Hindawi
Scientific Programming
Volume 2021, Article ID 8841188, 14 pages
https://doi.org/10.1155/2021/8841188

Research Article

HOVA-FPPM: Flexible Periodic Pattern Mining in Time Series Databases Using Hashed Occurrence Vectors and Apriori Approach

Muhammad Fasih Javed,[1] Waqas Nawaz ,[2] and Kifayat Ullah Khan [1]

[1]*IKMA Lab, Department of Computer Science, National University of Computer and Emerging Sciences, Islamabad, Pakistan*
[2]*Department of Computer and Information Systems, Islamic University of Madinah, Al-Madinah, Saudi Arabia*

Correspondence should be addressed to Waqas Nawaz; wnawaz@iu.edu.sa

Received 25 August 2020; Revised 26 November 2020; Accepted 21 December 2020; Published 4 January 2021

Academic Editor: Jiwei Huang

Finding flexible periodic patterns in a time series database is nontrivial due to irregular occurrence of unimportant events, which makes it intractable or computationally intensive for large datasets. There exist various solutions based on Apriori, projection, tree, and other techniques to mine these patterns. However, the existence of constant size tree structure, i.e., suffix tree, with extra information in memory throughout the mining process, redundant and invalid pattern generation, limited types of mined flexible periodic patterns, and repeated traversal over tree data structure for pattern discovery, results in unacceptable space and time complexity. In order to overcome these issues, we introduce an efficient approach called HOVA-FPPM based on Apriori approach with hashed occurrence vectors to find all types of flexible periodic patterns. We do not rely on complex tree structure rather manage necessary information in a hash table for efficient lookup during the mining process. We measured the performance of our proposed approach and compared the results with the baseline approach, i.e., FPPM. The results show that our approach requires lesser time and space, regardless of the data size or period value.

1. Introduction

Data mining allows us to discover useful information from the data that would otherwise be very hard to uncover. The process of data mining involves various tasks including classification, clustering, pattern mining, and several others [1]. Pattern mining is a subfield of data mining that focuses on discovering useful patterns within a given dataset [2]. It finds either a single item or a set of items that appear more than a certain threshold set by the user. A classic example would be the patterns of frequently bought items together. It is common for companies to store the transactions, containing the type and name of the items, in their databases. The data of these transactions are used to mine the patterns of frequently bought items together [3]. Traditionally, these patterns help the stores in deciding which set of items is required to place together for customers.

Time series datasets are common these days, having some of the application areas such as economics, social sciences, epidemiology, medicine, and physical sciences, for instance, measuring a person's heart rate after every minute, readings of air temperature or wind after every hour, stock rates at mid and end of every day, and so on. Mining the patterns from time series data is usually referred to as periodic pattern mining [4–7]. The periodic patterns in the mining process are usually categorized as symbol, sequence (partial), and full-cycle (segment) periodic patterns [8–11]. Symbol periodicity happens when there is a single event that reappears in the time series after a specific time period. Sequence periodicity is the reoccurrence of multiple symbols after a specific time period. Full-cycle periodicity is the repetition of the whole data in a given period. Recently, research focus has been shifted towards the flexible periodic pattern mining [9, 12–15]. The idea behind flexible pattern mining is to discover a set of events that do not

necessarily appear together, but those events are accompanied by random events in between them periodically, i.e., {$event_{imp}$, $event_{unimp}$, $event_{imp}$, $event_{imp}$}. The term flexibility refers to the existence of irregular occurrence of unimportant events between frequently occurring events at different time periods. Let us consider the example of bank transactions [12] denoted as {*a*, *b*, *c*, *d*} for amounts in millions for 0–5, 5–10, 10–15, and ≥15, respectively. Given a set of transactions {*abd*, *accd*, *ad*, *abbccd*} and a manager is interested to analyze the patterns of transactions having lower *a* and higher *d* amounts. We observe that there is no exact number of transaction types between *a* and *d*. Such a scenario can be formally represented as $a\{*\}d$, where $^{0}*^{0}$ shows flexibility in the number of options (transaction types) to consider during mining and we consider it as a flexible pattern mining problem.

The use of suffix tree (prefix trie or trie) data structure is prevalent among state-of-the-art approaches towards flexible periodic pattern mining. Rasheed et al. [9] constructed suffix tree from time series data to mine flexible periodic patterns. Similarly, Chanda et al. [12] introduced an improved approach based on suffix tree data structure. There exists in-memory linear time algorithm to construct the suffix tree [16] and pruning strategy suggested in [12]; however, repeated traversal over such a tree structure makes the overall mining process computation intensive. We observe various redundant (unnecessary) patterns kept for a significant amount of time, and tree size remains constant throughout the mining process. In our understanding, a better strategy is required to be space and time efficient while keeping necessary information and producing the same patterns.

In this paper, we introduce an efficient strategy based on hashed occurrence vectors and Apriori approach, which does not rely on suffix tree data structure [9, 12] rather on hash table for efficient lookup, to mine periodic patterns from time series data without compromising on the results. Our approach comprises of discretization of time series data, periodicity detection, and pattern mining modules. We maintain uniquely variable length periodic patterns along with their occurrence vectors as key-value pairs in a hash table, where key represents pattern and occurrence vector is stored as corresponding value. We use arithmetic on occurrence vectors of discretized events to compute periodicity of the underlying patterns. The proposed periodicity detection algorithm is different from existing algorithms in terms of removing the redundant occurrence vector values and computing periodicity of all patterns at same level in one pass. We tailor the incremental pattern mining process to use occurrence vectors of existing patterns in hash table to explore new patterns and update the hash table once patterns found at each level.

The contributions of our work are as follows.

(i) We propose HOVA-FPPM to simplify flexible periodic pattern mining process through an efficient key-value pair data structure.

(ii) We maintain necessary information (patterns and their occurrence vectors) and compute efficiently (through basic arithmetic on occurrence vectors, reducing redundant and invalid patterns) to significantly improve time and space complexity of our approach.

(iii) Our approach requires single pass to discover symbol, partial, and full-cycle periodic patterns at any periodic level.

(iv) Comprehensive performance analysis on real-world datasets with the baseline algorithm shows the run time effectiveness and space efficiency of our approach while maintaining the same accuracy.

The rest of the paper is organized as follows. We discuss related works followed by preliminaries section. The proposed methodology with detailed explanation of the algorithm is discussed afterwards. Subsequently, we outline the experimental analysis of the proposed methodology in comparison to the baseline approach on real-world datasets. We also provide a detailed discussion at the end. Lastly, we conclude the paper along with its future directions.

2. Literature Review

In this section, we briefly discuss various approaches in the context of periodicity mining, which is subfield of pattern mining. Pattern mining has other subfields depending on the type of mined patterns such as frequent item sets [3], sequential patterns [8, 11, 17, 18], high utility patterns [19–22], periodic patterns [23–25], as well as flexible periodic patterns [9, 12, 26]. Chanda et al. [12] put forth an algorithm recently that finds flexible periodic patterns in a time series by using suffix tree data structure. In their approach, they first made a suffix tree from the time series and then used that suffix tree to mine patterns of various lengths.

After an extensive literature review of the periodic pattern mining algorithms, we noticed that all of these algorithms can be categorized based on the approaches they used in their algorithmic solutions (refer to the taxonomy of frequent pattern mining in Figure 1). For instance, some algorithms use the Apriori approach to find patterns in an increasing length, whereas others use a tree-like structure to avoid the memory requirement and mine the patterns from it. Each approach has its own benefits and drawbacks. Therefore, instead of categorizing algorithms based on the types of patterns they were mining, we group them based on their underlying approaches. The key approaches were Apriori, tree based, projection based, and others. The other category contains algorithms that follow a uncommon approach.

2.1. Apriori-Based Approaches. The earlier works based on Apriori approaches were not able to mine all types of patterns at once. However, some important properties, monotone and anti-monotone, were introduced later that helped in improving the performance of mining algorithms [17, 18]. The most recent algorithm that could detect flexible periodic patterns of all types was presented by Nishi et al. [19]. The algorithm used Apriori-based approach to mine all the valid patterns by using the occurrence vectors.

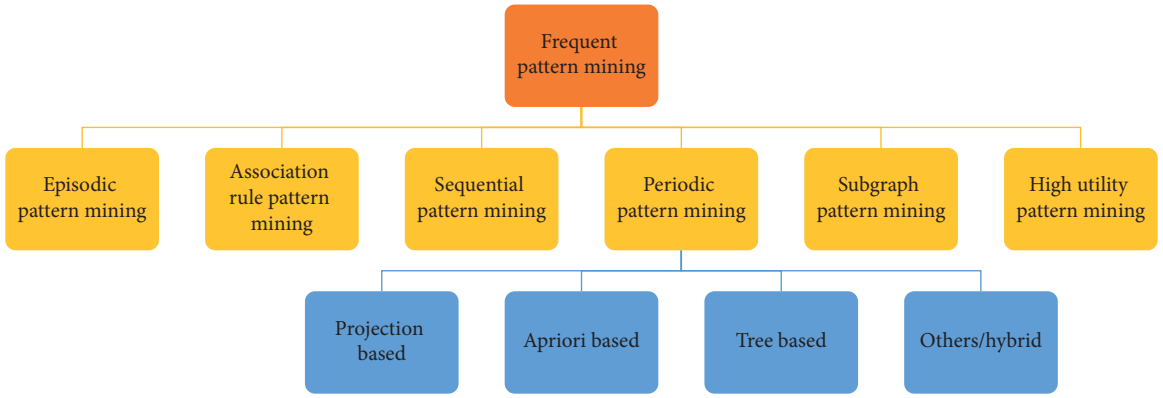

Figure 1: Taxonomy of research in frequent pattern mining.

Algorithm started by mining one-length patterns and then moved to the two length patterns and so on. The found patterns are made flexible by inserting "∗" as special character. As per the authors' claim, their algorithm is capable of finding some interesting patterns which was not possible with previous algorithms. They also claim that it finds all types of periodic patterns; however, the mining strategy is computationally expensive because of generating a lot of excessive patterns.

2.2. Tree-Based Methods. Many researchers were utilizing an important notion called "closed patterns" in the context of periodic pattern mining that has reduced search space towards mining redundant patterns [20, 21, 23]. Majority of these algorithms involved the usage of tree structures for pattern mining and ended up reducing the time and space complexity of the decreased number of patterns with reduced tree size. The research trend shifted towards mining a different type of patterns known as the high utility patterns [24–26]. Rasheed et al. [9] put forth an algorithm that finds flexible periodic patterns in time series databases using suffix tree. In this approach, suffix tree is created from the time series data and then used to mine patterns of various lengths. Although the authors used a technique of generating patterns incrementally from one-length patterns to two length patterns and so on, unlike other algorithms, they did not follow Apriori approach completely of making excessive and all possible combinations. The suffix tree allowed them to make the combinations with the events that appeared in the suffix tree. Another distinguishing aspect of their approach was that the patterns were not made flexible at the end of the process, rather, it kept attaching the unimportant event symbols with every mined pattern hoping to become a pattern at successive depth of the tree. Recently, Chanda et al. [12] introduced another efficient approach in the same direction for flexible pattern mining using pruned suffix tree.

2.3. Projection-Based Techniques. In this category, unlike other approaches, researchers used the projection database to mine the patterns with multiple pruning techniques. They introduced the concepts of flexible periodic pattern mining in the domain of closed pattern mining [14, 27]. For instance, Akther et al. [14] presented an algorithm to mining closed flexible patterns, which was an improvement over the previously presented paper on flexible closed patterns [27]. There were some other works on discovering closed patterns using projection database and tree structures [28–30].

2.4. Other Approaches. There are other approaches in literature for pattern mining that includes dynamic time warping [31], information gain [32], and manipulation of bitmap representation of data [33, 34]. The dynamic time warping algorithm [31] used a matrix to warp the time dimension of data for insertion or deletion of noise. The WARP algorithm is not capable of detecting all types of periodic patterns; however, handling noise in the data makes it prominent among other approaches. In another work, the authors proposed an extended version of information gain measure [32] to detect periodicity. It is focused on flexible pattern mining with gap penalties, but it only worked for symbol and partial periodic patterns. Another study [35] detected partial periodic patterns with the aim of eliminating the need to know the period lengths before mining. Ayres et al. and Lucchese et al. [33, 34] worked on long sequential pattern mining. The data are represented in bitmaps where depth-first search strategy is used to mine the long sequential patterns.

It is evident from the above discussion that the existing approaches either employ complex data structure for pattern mining process or generate excessive redundant or invalid patterns. For instance, in Apriori-based approaches, excessive generation of redundant or invalid patterns reduces time and space efficiency. The effect of excessive pattern generation on tree- and projection-based approaches results in complex tree structure, which further increases the memory usage as well as traversal times. The approach with an effective pattern generation strategy and efficient repeated traversal over data has the potential to overcome the aforementioned limitations of existing studies.

2.5. Preliminaries. In this section, we explain existing terminologies to understand the problem and its solutions.

Definition 1. (periodicity). In a time series T, $T = \{a_0, a_1, a_2, a_3, \ldots, \ldots a_n\}$, with n events, finding how many times an event

reoccurs within a specified period of time is known as the periodicity of that event.

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$ based on daily activities schedule of a working individual in office, as shown in Figure 2, event "a" is periodic for period value 5. This means if T is divided into pieces of length 5, "a" will always appear at the same place within those pieces. In other words, there will always be four events in between two occurrences of "a." However, if we look at the period value 3 and divide T in pieces of length 3, "a" would not be considered as periodic because "a" does not appear after every two events.

Definition 2. (perfect periodicity). In a time series T, $T = \{a_0, a_1, a_2, a_3, \ldots, a_n\}$, with n events, an event reoccurring within a specified period of time for the entirety of the time series is known as perfect periodicity.

Perfect periodicity is denoted by PP and defined as follows:

$$\mathrm{PP}(\mathrm{period}, \mathrm{start}, \mathrm{end}, \mathrm{Pattern}) = \left[\frac{\mathrm{end} - \mathrm{start} + 1}{\mathrm{period}}\right]. \quad (1)$$

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$, PP (5, 0, 19, "$abd$") = $(19 - 0 + 1)/5$ results in 4 that means "abd" would have to appear 4 times in the time series in order to have perfect periodicity. However, it has an actual periodicity of 3 in the data.

Definition 3. (periodic pattern). When a pattern repeatedly occurs for a specific period of time and satisfies the support threshold, we call it periodic pattern.

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$, pattern "$abd$" is periodic with a support of 3 for period value 5 when the threshold support value is 3. In short, event or a sequence of events will be considered a periodic pattern if they satisfy the minimum support threshold for the given period value. This should not be confused by the support measure. For example, consider the events "ca" in the time series T. If we look at the events in T, we notice that "ca" appears a total of 3 times, which means it satisfies the support threshold considering the whole time series. However, "ca" only appears once if we look at T with a period value of 5 if we start at the index 0 of T.

Definition 4. (flexible periodic pattern (FPP)). Any periodic pattern that contains "do not care" events is called flexible periodic pattern.

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$, the pattern "$a * d$" is a periodic pattern with support 4 and period value 5. It means that event "a" and event "d" occur after every 4 values with a random event in between them. Flexible periodic patterns are difficult to mine through regular pattern mining approaches due to the occurrences of unimportant events.

Definition 5. (occurrence vector). An occurrence vector is a list that represents the index positions of a unique element (event) in the data. Every unique element (event) has one occurrence vector.

Daily schedule	Discretized events	Discretized sequence
Coffee	a	$\{a, b, d, b, c\}$
Work	b	
Lunch	d	
Work	b	
Talk break	c	
Coffee	a	$\{a, b, d, b, a\}$
Work	b	
Lunch	d	
Work	b	
Coffee	a	
Coffee	a	$\{a, b, d, a, c\}$
Work	b	
Lunch	d	
Coffee	a	
Talk break	c	
Coffee	a	$\{a, c, d, c, a\}$
Talk break	c	
Lunch	d	
Talk break	c	
Coffee	a	

Figure 2: Daily activity schedule of a working individual at office.

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$, the occurrence vector of the element "a" is [0, 5, 9, 10, 13, 15, 19], which represents the index positions of the occurrences of event "a" in T.

Definition 6. (difference vector). Given a pair of occurrence vectors, the difference vector generates a pattern along with its frequency count.

The objective of a difference vector is to obtain a pattern along with its frequency. It is computed by comparing each value of the first occurrence vector with every value of the second occurrence vector. For a given transaction in Figure 3, the pattern ab occurs three times. To generate this pattern, we need to compute the difference $a - b$, which provides us three occurrences of value -1 in Figure 4 in a table titled "Differences." Similarly, the pattern $a * * * * b$ appears two times; hence, we find the value -5 two times. The difference vector can be positive or negative. A negative value denotes that an element appears before the other element, i.e., a appears before b while computing the difference $a - b$. A positive value for the said difference denotes that a appears after b.

Definition 7. (confidence). Confidence of a pattern is the ratio of actual periodicity and perfect periodicity, as shown in the following equation:

$$\mathrm{conf}(p, \mathrm{strt}, \mathrm{pattrn}) \frac{\mathrm{AP}(p, \mathrm{strt}, \mathrm{end}, \mathrm{pattrn})}{\mathrm{PP}(p, \mathrm{strt}, \mathrm{end}, \mathrm{pattrn})}. \quad (2)$$

For example, in time series $T = \{abdbc\ abdba\ abdac\ acdca\}$, the perfect periodicity of pattern "abd" is 4 and the actual periodicity is 3. Therefore, the confidence of this pattern is 75%, i.e., confidence (5, 0, "abd") = 3/4.

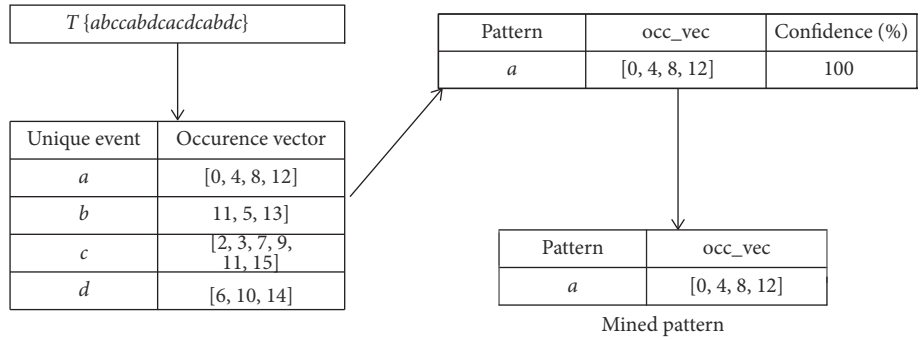

Figure 3: The proposed mining process for one length flexible periodic patterns.

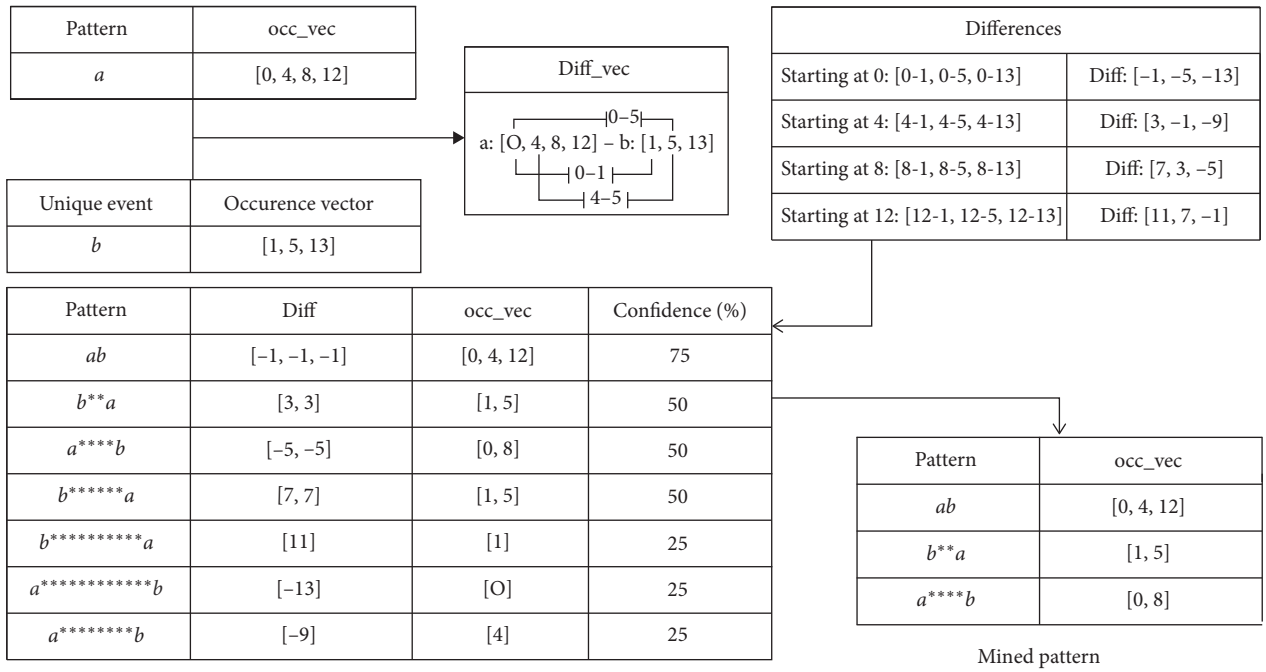

Figure 4: The proposed mining process for two length flexible periodic patterns.

3. HOVA-FPPM: Hashed Occurrence Vectors and Apriori Approach for Flexible Periodic Pattern Mining

The goal of HOVA-FPPM is to overcome the limitations of existing approaches, by eliminating the need of complex data structure such as suffix tree, through Apriori-like approach over occurrence vectors maintained in a hash table to mine flexible periodic patterns. It helps us not only to reduce the space requirements but also to avoid temporary patterns with limited calls to periodicity detection module, which eventually reduces the overall time requirements of the mining process. In the following passages, we discuss the components of our strategy and the proposed algorithm for pattern mining.

3.1. Components of HOVA-FPPM Strategy. The key components involved in our proposed approach are discretization, event lookup table construction, periodicity detection, and pattern mining as depicted in Figure 5.

3.1.1. Discretization. The discretization process converts each unique element from the data series to a simplified unique element. The sole purpose of the discretization process is to make the process of pattern mining easier, since it is easier to work with a series of data with unique characters than to work with a series of alphanumeric product IDs. For instance, if we have a data series $t = \{asyt2345, kgjhu6789, enhy4521, enhy4521\}$ containing product IDs, then the discretizing makes our input as $t = \{a, b, c, c\}$. Once we perform the discretization over actual time series data, then it produces simplified representation of the same data.

3.1.2. Events Table Construction. The events table consists of all unique events and their occurrence vectors. Occurrence vectors are of significant importance because they hold the index position of each unique element in the input data. They are even more important because our proposed algorithm relies heavily on the occurrence vectors. Each

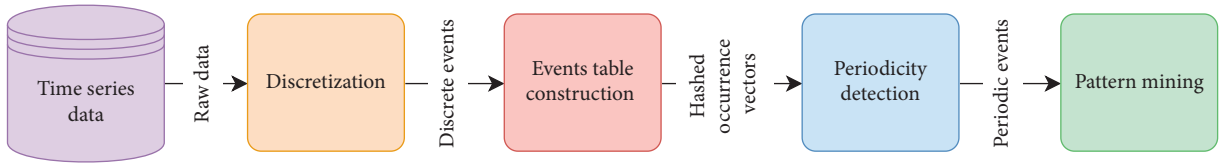

Figure 5: The sequentially dependent components of our proposed strategy.

unique element in the input data has its own occurrence vector. The values in the occurrence vector are the index positions of that element where it is located in the input data. For example, if we have an input data $t = \{abcdabddabccabdc\}$, then the occurrence vectors would be $a = [0, 4, 8, 12]$, $b = [1, 5, 9, 13]$, $c = [2, 10, 11, 15]$, and $d = [3, 6, 7, 14]$. Occurrence vectors are calculated while scanning the input data once. During the scanning process, when we find an element that does not exist in the events table, then we add it to our hashing-based event table and allocate the corresponding list. If the element already exists in the events table, then we add the current index position to its occurrence vector.

3.1.3. Periodicity Detection. Once we have gone through the input data, we should have the occurrence vectors of all the elements in the data. We now calculate the periodicity of each element by passing the element and its occurrence vector to the periodicity detection algorithm. Our periodicity detection algorithm is slightly modified compared with the FPPM's [12] periodicity detection algorithm. The original periodicity detection algorithm goes through the occurrence vector of a particular event n times. It then makes another iteration for $n - 1$ times and compares all the values from the $n - 1$ values of the occurrence vector to check their periodicity. At the end, the periodicity detection algorithm returns a list of occurrence vectors of the periods for which the event is periodic, where each starting position has separate list. For example, the output of the periodicity detection algorithm for $a = [0, 4, 8, 12]$ would be starting position $0 = [0, 4, 8, 12]$, $[0, 8]$, $[0, 12]$, starting position $4 = [4, 8, 12]$, $[4, 12]$, and starting position $8 = [8, 12]$. We can see that the occurrence vector $[0, 4, 8, 12]$ for starting position 0 and occurrence vector $[4, 8, 12]$ for starting position 4 are redundant because both point to the same occurrence vector. Mining and returning such redundant occurrence vector is unnecessary because superperiodic occurrence vector mines suboccurrence vectors in successive iteration at next starting position. We do not compute such redundant occurrence vector in our modified periodicity detection algorithm, while rest of the steps remains intact.

3.1.4. Pattern Mining. The pattern mining module of our proposed strategy reduces the time required to discover flexible periodic patterns compared with its counterpart. The mining process is based on Apriori approach and involves basic arithmetic operations among occurrence vectors, i.e., difference vector. The occurrence vectors along with the events are managed effectively using key-value pairs with inherent hashing strategy to provide quick lookups for time efficiency. The pattern enumeration step is guided through unique events and polarity of the difference vector values, while frequencies help in deciding whether the pattern is frequent or not given the threshold.

3.2. Illustration of HOVA-FPPM with an Example. We illustrate the process of our pattern mining algorithm with the help of a toy example. For this illustration, we assume the data $T = \{abccabdcacdcabdc\}$ with a confidence threshold of 50% and maximum allowable unimportant events as 3. The mining process behaves similarly for all sizes of the datasets. The illustration of the process in Figures 3, 4, and 6 does not reflect complete patterns and combinations due to space limitation. After the discretization process, our data should be in the form as given above.

Since we have already passed through the data once, to discretize it, we have the occurrence vectors of all the unique elements in the data. The unique elements are a, b, c, and d. The occurrence vectors of all these unique elements are available, as shown in Figure 4. Our mining process goes through each unique element and their occurrence vectors to mine the patterns. First, we consider the element a and take its occurrence vector to check the periodicity. It turns out to be periodic for periods of length 4 and 8; therefore, we consider it as one length mined pattern.

For two length patterns, we compare the occurrence vector of one length mined pattern with the occurrence vectors of all other unique elements in our events table. It may generate all the patterns involving elements ab, ac, ad, ba, ca, and da. The way our algorithm works allows us to mine patterns starting from all the elements involved in one iteration, which almost halves the computation of the iteration. We perform difference operation between the occurrence vectors of both elements, one is mined pattern a and second is b, as shown in Figure 5. We compare first value of the occurrence vector of a element with all the values of the occurrence vector of b. We repeat this process for all the values of occurrence vector of a. As a result, we maintain lists of difference values for each position.

The difference operation reveals important information related to frequency, position, and occurrence of patterns. In each difference vector, we have positive and negative values. Since we are performing "$a - b$" operation, the negative numbers give us the difference between the positions of the a and b, whereas a positive difference value indicates the distance between b and a. For instance, if we have 4 values from the a occurrence vector and we take the difference with 1 value of the b occurrence vector, then it gives us the difference of 3. The difference value is positive, so we know

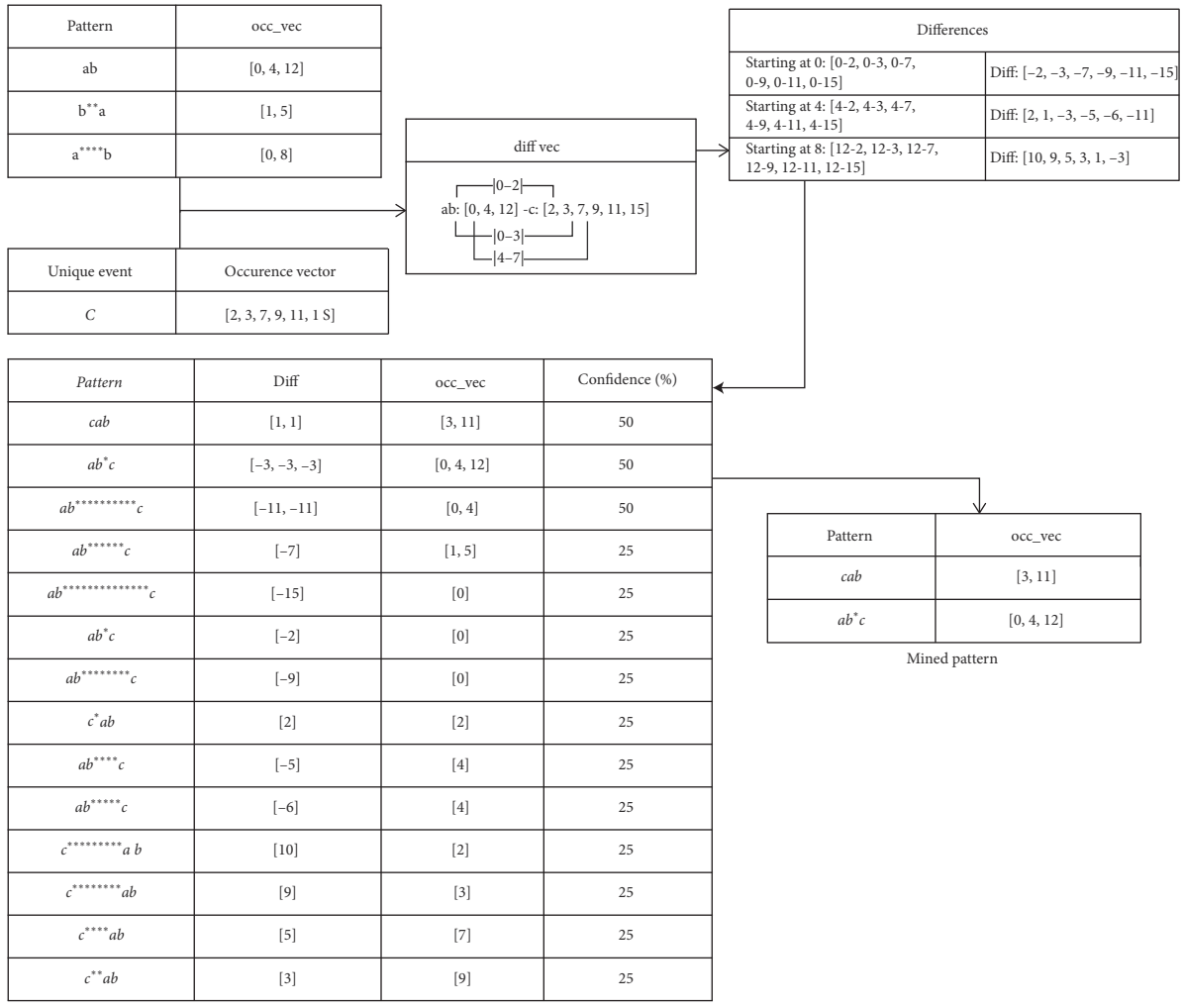

Figure 6: The proposed mining process for three length flexible periodic patterns.

that this is the distance between b and a. In this particular instance, b has occurred prior to a.

If we find 1 positive value 4 times in the difference vectors obtained through the difference operation, then we can conclude that b occurred 4 times before a with a distance of 1. The frequency value 4 of 1 tells us the confidence of that pattern, distance of 1 reveals event distance between b and a, and the positive sign shows that b occurred before a instance. Therefore, the mined pattern from this operation is ba with a frequency of 4. If we assume the distance value is 2, then it means a occurred with distance value of 2 after b, which results in generated pattern $b * a$.

The above process is repeated for all successive patterns to be mined from data without redundant and unnecessary computations (see Figure 6 for three length patterns). We get separate difference vectors after the difference operation, where we count the frequencies of all the unique numbers to check whether the pattern is frequent or not. The orientation of the pattern is determined through polarity of the numbers, i.e., ab or ba pattern. The difference tells us how many other events are between a pair of events. Since the maximum allowable unimportant event limit is 3, any number greater than 3 is not considered for the pattern generation. We are able to generate next level patterns based on these difference vectors. In the current scenario, the patterns ab and $b * * a$ with occurrence vectors [0, 4, 12] and [1, 5], respectively, are the only mined patterns. It is important to notice that we did not perform any explicit iteration to get the patterns involving b and a because we already mined these patterns. Therefore, when we mine patterns starting from the b element, we do not compare its occurrence vectors with the a element rather with c element. The process is same and generates patterns involving ac and ca. Similarly, in successive iterations, we do not mine patterns ca when considering the c and a elements. In this way, we are able to reduce computations especially when we have a huge amount of unique elements. We have a hash table similar to the original one but includes variable length patterns along with their occurrence vectors. For next level patterns, we take the ab and $b * * a$ patterns and their occurrence vectors to compute the difference vectors. The remaining process is the same as that of one length elements (Algorithm 1).

3.3. Proposed Algorithm. In this section, we discuss the details of our proposed algorithm (see Algorithm 1) for

```
Input: event E, Occ_vec All Events S, Max_Pattern_Length, Prev_Keys
Output: a list of mined patterns
NP = {}, New_Events = slice_even (S, Prev_Keys) Periodic = periodicity (E, Occ_vec,lengthofOcc_vec,Confidence) for i in {O1, ..., On}
of Period do
    for Key in {K1, ..., Kn} of New_Events do for Item
                in {I1, ..., In} of
                New_Events(Keys) do
                Difference ← Periodic – Item if
        continue
      end        Difference > Max_Pattern_Length
                then
                if Item > Periodic then if
                Difference > KeySize then
                star_count = Difference–
                KeySize
      end
              if star_count > star_limit then
      end
                continue
                end
      end        stars = calculate_stars(star_count) pattern_key = E + stars + Key
      end        patterns_with_stars = Periodic[i] NP ∪ pattern_with_star
else
                pattern_key = Key + E patterns_without_stars = Periodic[i]
                NP ∪ pattern_without_star_count
      end      if Difference > KeySize then
                star_count = Difference – KeySize
else
                if star_count > star_limit then
                continue
      end        end
      end        stars = calculate_stars (star_count) pattern_key = Key + stars + E pattern_with_star
    end          =Periodic[i] NP ∪ pattern_with_star
    pattern_key = Key + Epattern_without_star = Periodic[i] NP ∪
    pattern_without_star
  end for j in {Key1, ..., Keyn} of New_Events do
  NL_Patterns = calc_next (New_Events, NP, Max_Pattern_Length) NP ∪
    update (NL_Patterns)
      end
    end
ReturnNP
```

Algorithm 1: The proposed algorithm.

mining the flexible periodic patterns. The prerequisite for this algorithm is the discretization of the data, and it is executed initially for each unique element of the dataset. It takes a unique element along with its occurrence vector, the hash table containing all the unique elements, maximum pattern length, and previous keys as input. The previous keys refer to a list that holds already mined events. It is critical to maintain such information because, as we discussed earlier, once we mine the patterns starting from a, we expect to have inverse patterns. The previous keys list prevents us from remining the patterns. The rest of the input is self-explanatory. The output of the algorithm is a list of mined patterns. The key operations involved in our proposed algorithm are as follows.

(i) Discretize the events while passing through the time series data and count the total occurrences, occurrence vectors, and maximum pattern length for the entire time series. We also make pairs and hash their values in the same pass.

(ii) Find the differences in occurrence vectors of each unique event.

(iii) Make two length pairs of the frequent pairs that have surpassed the user given support threshold.

(iv) Add the unimportant symbol for each pattern based on the difference of occurrences.

(v) Find the differences in occurrence vectors of each consecutive event pair.

(vi) The frequent events with highest difference count values are considered in the successive levels.

(vii) Add the unimportant symbol for each pattern based on the difference of occurrences.

(viii) Repeat for the next level of patterns.

In the subsequent paragraphs, we explain the steps our proposed algorithm. In line 1, we initialize an empty list to save the mined patterns, and line 2 generates a list of events that we have to mine. Line 2 calls the slice_events function that takes the hash table and previous keys as an input and slices the previous key hashes along with their occurrence vectors. It helps us in reducing mining time. The repetition construct at lines 4–6 scans entire hash-table and its occurrence vectors to compare the events' occurrences. Line 7 computes the difference of elements, and lines 8 and 9 perform a check on the difference value and maximum pattern length. At this stage, we eliminate any difference that is greater than the maximum allowable skippable unimportant event.

Since we expect different patterns depending the polarity of the difference value, we need separate pattern name making sections for it. We perform the operation $a - b$ meaning that the second element in the difference operation is bigger if the result of the difference is positive; therefore, we perform checks on lines 10 and 23. These sections enable us to make patterns depending on the polarity of the difference value. Line 11 is a check on the difference and the key size. It is important because all our mined patterns are represented by the index of the first element of the pattern. It means that if we have a pattern *abc*, then this pattern will have the same starting position as the *a* element. If we have to mine a 4 length pattern from this, then we need to consider elements with a starting position of current key size away. In this case, the pattern *abc* has a key size 3. Any element that is not 3 steps ahead of the starting position of the current pattern would make it an invalid pattern, which is enforced at line 11. Line 12 calculates the difference from the current key to the next key under consideration and gives us the star count that we would have to add if we mine this pattern. Lines 13 and 14 are a simple check on the maximum skippable event and continue action, respectively. Line 15 calls a function that returns a string containing the number of stars we have from line 12. Line 16 makes a new pattern, and line 17 inserts the occurrence vector value in it. Line 18 saves the newly generated pattern in the pattern list. Lines 20–22 perform the same operation but for single-length events. Lines 23–35 are similar to the ones mentioned above but this section mines the reverse patterns. Lines 36–38 send the newly mined pattern along with the full hash table to next level generation function, which works on more than 2 length patterns. We did not check for periodicity of the newly generated patterns in this function yet. It is because we send these to the next level function, which automatically checks the periodicity itself. Once the patterns are mined and returned, the return list will have all the mined patterns related to the specific event that was passed on to our algorithm function. This whole process will be repeated for every individual unique element.

3.4. Complexity Analysis. We discuss the algorithmic complexity of both approaches based on their key steps. The baseline approach involves three steps that includes suffix tree construction, periodicity check, and pattern generation towards mining. The authors of FPPM algorithm claim that the pruned suffix tree construction has same time complexity as the suffix tree contraction, i.e., $O(n)$, where n is the size of the time series. The periodicity detection relies on the occurrence vector and length of the periodic pattern, i.e., $O(k * n\hat{2})$, where k is the maximum length of periodic pattern and n is size of the time series. The pattern generation involves the levelwise traversal of the pruned suffix tree rooted at corresponding events. The average depth of the traversal is determined by the ladder factor computed prior to the mining step for each node. The overall complexity of the pattern generation step is logarithmic to the number of nodes in a subtree rooted at a particular event, i.e., $O(m * \log(n)\hat{2})$, where n is the number of nodes of a subtree and m is the number of rooted subtrees.

On the other hand, the proposed approach consists of events table construction, periodicity detection, and pattern mining. The event table construction scans the input data and generates events with corresponding occurrence vectors, linear in time complexity $O(n)$. The periodicity detection algorithm remains the same as the baseline approach so does the complexity of this module. However, our approach deviates in determining periodicity of the patterns early from successive iterations and avoiding it later. The overall complexity remains the same as that of the baseline approach while minimizing the total number of computations. The proposed approach has superiority in pattern mining stage, where we explicitly avoid the traversal of the suffix tree by hashing key-value pairs for quick lookup for pattern generation, i.e., from log-scale to constant time. We process the available set of frequent patterns and their occurrence vectors to generate patterns at next level; therefore, complexity depends on total number of patterns p and associated occurrence vectors, i.e., $O(k * h(p)\hat{2})$, where k is the length of the pattern and $h(p)$ represents hashed patterns. We did not introduce a new strategy for discretization, which is similar for both approaches, so we neglected it in our complexity analysis.

3.5. Experiments. In this section, we perform experiments to evaluate the efficiency of our proposed approach against baseline approach, i.e., FPPM. In the experimental analysis, we focus on time and space efficiency of both algorithms (the proposed HOVA-FPPM and baseline) by varying period value and data size. Since we use the same settings/parameters (i.e., confidence 60% and support 50% inspired from baseline approach) during our experiments, obtained results are justified as per our claim in this article to prove the superiority of the proposed approach. However, we understand that different values of these parameters will affect the mining results. For instance, increasing the support threshold will reduce the number of mined patterns. We assume that both algorithms produce the same results (no difference in result accuracy), rather, have different requirements for time and space during execution.

We briefly discuss the datasets used in our experiments and describe the results followed by discussion.

3.5.1. Datasets and System. In our experiments, we use two datasets namely diabetes and bike sharing datasets. Both datasets are publicly available on the UCL Machine Learning Repository with the same names. The diabetes dataset contains a total of roughly 8000 records. It contains records of diabetic patients and their health related results. A total of 20 unique codes are used throughout the dataset to calculate the health of the patients. All of these 20 codes represent measurement for the patient. Each of these codes will have a value for each patient along with a date column to represent time of the patient history record. On the other hand, the bike sharing dataset had a total record of 17000 (roughly) for the hours that the bikes were shared and 730 records for the same bikes but on a day level. The specification of the system used for the experiments is an i5-3320M 2.6 GHz processor with 8 GB RAM and Windows 10 operating system. The algorithms were implemented in Python.

3.6. Performance Based on Time and Space. We analyze the performance of the proposed algorithm with the baseline approach, i.e., FPPM, by measuring execution time and memory space usage.

3.6.1. Performance Based on Varying Data Size. We try to understand the performance aspect of the proposed algorithm with the baseline approach by varying the size of the data. If we look at the performance of our proposed algorithm (Figure 7), it is evident that our algorithm outperforms the FPPM in both time and space requirements. It is worth noting that the time difference between algorithms is quite significant as compared to the space difference. This is because of the fact that the number of patterns increases significantly with more data. The output (mined patterns) of both of the algorithms is the same. Since our proposed algorithm does not rely on the suffix tree-like data structure, we do not need to allocate the memory and store the data in a tree. In our algorithm, the memory requirement increases with the number of patterns. The dataset under consideration is not big; therefore, space requirements did not grow much for the proposed algorithm. In contrast, the FPPM algorithm's memory requirement increases almost linear to the data growth.

3.6.2. Performance Based on Varying Period Value. Now we look at the data from varying period value perspective (see Figure 8). This analysis helps us to uncover the effects of changing the period value while keeping the data size fixed for both algorithms. The patterns mined from the data are from all starting positions. As we can see, the time performance of our propose algorithm is significantly better than the FPPM. However, the space requirements start to favor the FPPM in varying period values. This is because of the fact that the space requirement of our algorithm grows with the periods unlike the FPPM, whose space requirement remains fixed.

3.6.3. Performance Analysis on Bike Sharing Dataset. We also performed experiments on the second dataset, i.e., bike sharing dataset, which is comparatively bigger than the first one. It contains 17000 rows of data containing hourly updates of bike sharing information. We performed experiments on this dataset to understand the scalability aspect of both algorithms. Figure 9 depicts the time performance of both algorithms on a maximum of 6000 rows of data. The proposed algorithm outperformed the FPPM by cutting the time into almost half of what is required for FPPM approach. We performed these experiments with the same settings as before, i.e., confidence 60% and period value was ≤6, and we only varied the data size.

The results of space allocated to both algorithms are quite interesting in the case of varying dataset, as shown in Figure 9. The settings of the experiment were the same on bike sharing dataset. It is evident that the space requirements of FPPM are quite stable and increase slowly with respect to the input dataset size. However, our proposed approach has a very variable space allocation. It is also worth noting that the space used by the FPPM does not change, at least significantly, during the mining process, and the space used by the FPPM is allocated at the start of the algorithm when the suffix tree is generated. On the other hand, the data shown in Figure 9 contain the maximum amount of space allocated to our proposed algorithm during the mining process. This is because our algorithm dynamically changes the space during the mining process, which usually depends on the number of patterns per iteration. This explains the unstable space used during the mining process of our proposed approach. Based on our observations, the number of new patterns generated is not directly proportional to the new rows of data added to the input dataset.

4. Discussion

Our discussion revolves around selected related questions. In those questions, we aim to analyze the effects of Apriori approach over flexible periodic pattern generation with varying starting position to identify the factors affecting the generation of invalid or redundant patterns and to understand the relationship between varying dataset length and result's accuracy.

4.1. Effects of Apriori Approach over Flexible Periodic Pattern Generation with Varying Starting Positions. In order to analyze the effects of Apriori approach, we performed experiments on two datasets with varying properties and compared the results with the FPPM performance, which is a tree-based approach for flexible periodic pattern mining. As per our knowledge, no other algorithm is designed to mine flexible periodic patterns with variable starting position while using an Apriori-based approach. We already discussed that it is possible to mine patterns just from the occurrence vectors of the events without using a tree structure to simplify the process. As per the experiments and analysis of the results, we concluded that Apriori approach is better when (i) the available space for mining patterns is not

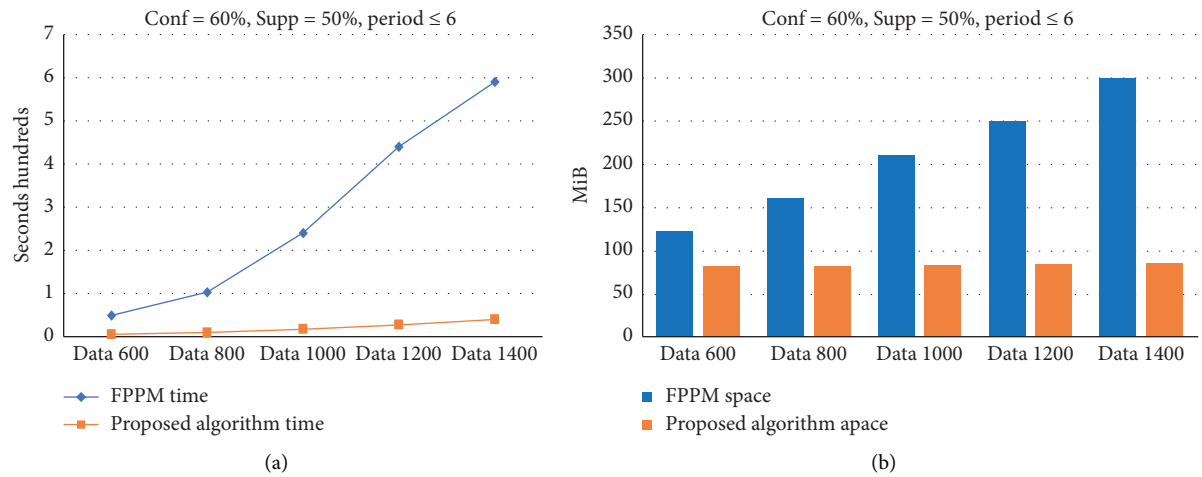

Figure 7: FPPM vs. proposed algorithm: time and space results on diabetes dataset of varying size.

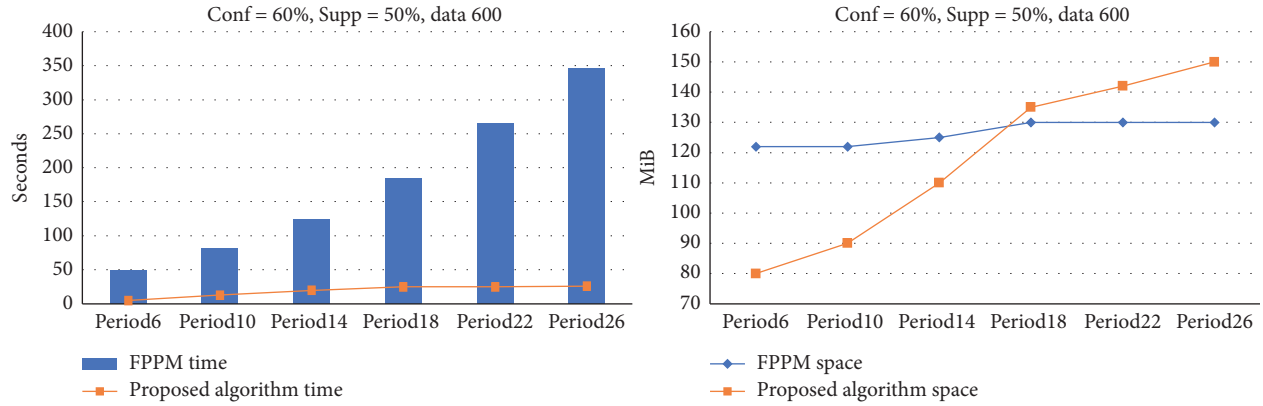

Figure 8: FPPM vs. proposed algorithm: time and space results on diabetes dataset of varying period value.

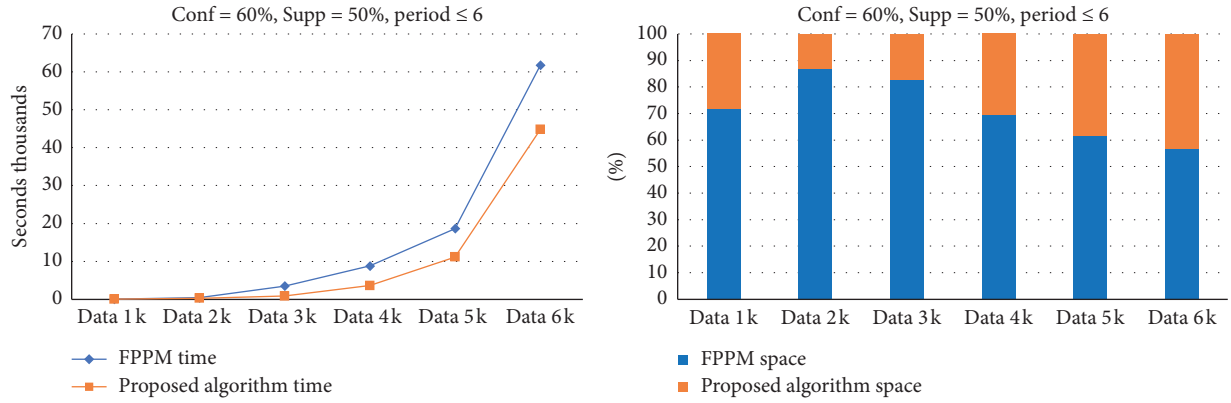

Figure 9: FPPM vs. proposed algorithm: time and space results on bike sharing dataset with varying data size.

an issue and (2) there are no strict restrictions on the space usage. An Apriori-based approach keeps the occurrence vectors and patterns in the memory, which can fluctuate depending on the data and unique elements in the data.

There is prominent positive effect of Apriori approach on the time required to mine patterns because it lacks tree traversal process. The existing Apriori-based algorithms, being capable of mining the flexible periodic patterns, were costly in space and time requirements. We overcome the excessive time requirements of Apriori-based approaches by introducing an efficient algorithm to mine the patterns. We achieve this by tweaking our algorithm to take advantage of

the pattern generation process, which can generate patterns from multiple passes. Therefore, our proposed algorithm is able to mine both *ab* and its inverse patterns in a single pass. The pattern generation process also relies on the occurrence vectors to generate necessary patterns, which reduces the redundant and invalid patterns.

4.2. Factors Affecting the Generation of Invalid or Redundant Patterns. In order to identify the factors affecting the generation of invalid or redundant patterns, we first analyzed the suffix tree behavior of the FPPM and its effect on pattern generation. We achieve this by running the FPPM on varying data size and varying unique values. The reason for focusing on the suffix tree was the way the FPPM mining process works. FPPM mines patterns level by level by visiting each immediate descendent nodes from root node. We used dataset of varying size and number of unique values to change the size of the suffix tree for analyzing its effect on the performance of the algorithm and redundant pattern generation. Varying the data size increases the depth of the suffix tree, whereas large total number of unique elements increases the breadth of the suffix tree. The breadth of the suffix tree has no significant effect on the redundant pattern generation; however, more redundant patterns were generated with the increased length of the data with more depth of the tree. Since FPPM holds the redundant patterns for each branch, the cost of redundant patterns is a lot higher with more depth of the suffix trees. On the other hand, more unique values did not affect the performance of redundant pattern generation because FPPM keeps patterns in memory for short period of time. One of the possible reasons is that the shallow tree allows FPPM to switch to a newer branch and discard the previous nonfrequent patterns.

4.3. Varying Dataset Length and Results Accuracy. It is evident from our experiments that there is a relationship between varying data size and the accuracy of results. The results obtained have accuracy of 33–50% reaching the lower limit on bigger dataset. It proves that the algorithms generate large patterns based on the initially generated patterns. Any mismatched pattern at the beginning leads to an increased reduction of accuracy. After careful observation, it was revealed that the unmatched patterns were produced by both algorithms. Any unmatched pattern at initial phase results in lowering the accuracy at successive phases. However, in very few cases, the patterns generated in the later phases did match. It reveals that in order to improve the accuracy of the proposed algorithm, the patterns generated in the initial phase require more attention. If the initial patterns of both algorithms are similar, then it would increase the accuracy of the mined patterns significantly because both algorithms use the initial patterns to generate the next phase patterns.

5. Conclusion and Future Directions

We presented an efficient strategy, i.e., HOVA-FPPM, to mine flexible periodic patterns from time series database without using complex data structures. We identified the limitations of tree-based approaches and discovered various factors that cause the redundant or invalid pattern generation during the mining process. We surpassed those issues with the help of hashing-based data structure while minimizing the number of redundant and invalid patterns through manipulation of occurrence vectors. The proposed solution outperformed the baseline algorithm in terms of time needed to mine the same patterns. We empirically justified that Apriori-based approaches are effective for the mining process without excessive pattern generation. For small dataset, our algorithm is space efficient compared with the FPPM. On a larger dataset, the space requirement of our strategy fluctuates due to incremental growth of accumulated data in hash table in contrast to baseline approach. We aim to improve the space complexity of our proposed algorithm on vary large datasets as an extension to this work. The analysis of combining hashing strategy with tree-like structure is another aspect to discover in future.

Data Availability

The datasets used in this research could be downloaded freely at UCL Machine Learning Repository (diabetes: https://archive.ics.uci.edu/ml/datasets/diabetes; bike sharing: https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset). However, the authors are willing to share the used dataset on request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Authors' Contributions

MFJ was solely responsible for the data curation, software development, and initial report. WN handled the original draft preparation, project administration, and funding acquisition. MFJ and KUK were responsible for the conceptualization of the idea, methodology, and formal analysis. WN and KUK reviewed and edited the manuscript.

Acknowledgments

This study was partially supported by Deanship of Research at Islamic University of Madinah (IUM), Saudi Arabia (Tamayuz-1 program of academic year 1439–1440 AH; research project number: 24/40).

References

[1] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[2] C. St-Onge, N. Kara, O. A. Wahab, C. Edstrom, and Y. Lemieux, "Detection of time series patterns and periodicity of cloud computing workloads," *Future Generation Computer Systems*, 2020.

[3] T.-c. Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[4] M. Patel and N. Modi, "A comprehensive study on periodicity mining algorithms," in *Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pp. 567–575, IEEE, Jalgaon, India, December 2016.

[5] Q. Yuan, J. Shang, X. Cao, C. Zhang, X. Geng, and J. Han, "Detecting multiple periods and periodic patterns in event time sequences," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 617–626, Singapore, November 2017.

[6] H. Yuan, Y. Qian, and M. Bai, "Efficient mining of event periodicity in data series," in *Proceedings of the International Conference on Database Systems for Advanced Applications*, pp. 124–139, Springer, Chiang Mai, Thailand, April 2019.

[7] R. A. Rizvee, M. S. H. Shahin, C. F. Ahmed, C. K. Leung, D. Deng, and J. J. Mai, *Sliding Window Based Weighted Periodic Pattern Mining over Time Series Data*, IBAI Publishing, Fockendorf, Germany, 2019.

[8] P. Indyk, N. Koudas, and S. Muthukrishnan, "Identifying representative trends in massive time series data sets using sketches," in *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 2000*, pp. 363–372, Cairo, Egypt, September 2000.

[9] F. Rasheed, M. Alshalalfa, and R. Alhajj, "Efficient periodicity mining in time series databases using suffix trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 79–94, 2010.

[10] S.-S. Chen, T. C.-K. Huang, and Z.-M. Lin, "New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1638–1651, 2011.

[11] K. Xylogiannopoulos, *Data structures, algorithms and applications for big data analytics: single, multiple and all repeated patterns detection in discrete sequences*, Ph.D. dissertation, University of Calgary, Calgary, AB, USA, 2017.

[12] A. K. Chanda, S. Saha, M. A. Nishi, M. Samiullah, and C. F. Ahmed, "An efficient approach to mine flexible periodic patterns in time series databases," *Engineering Applications of Artificial Intelligence*, vol. 44, pp. 46–63, 2015.

[13] A. K. Chanda, C. F. Ahmed, M. Samiullah, and C. K. Leung, "A new framework for mining weighted periodic patterns in time series databases," *Expert Systems with Applications*, vol. 79, pp. 207–224, 2017.

[14] S. Akther, M. R. Karim, M. Samiullah, and C. F. Ahmed, "Mining non-redundant closed flexible periodic patterns," *Engineering Applications of Artificial Intelligence*, vol. 69, pp. 1–23, 2018.

[15] J. Chen, K. Li, H. Rong, K. Bilal, K. Li, and S. Y. Philip, "A periodicity-based parallel time series prediction algorithm in cloud computing environments," *Information Sciences*, vol. 496, pp. 506–537, 2019.

[16] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.

[17] L. Ma and Y. Qi, "An efficient algorithm for frequent closed itemsets mining,"vol. 4, pp. 259–262, in *Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, vol. 4, pp. 259–262, IEEE, Wuhan, China, December 2008.

[18] H. Duong, T. Truong, and B. Vo, "An efficient method for mining frequent itemsets with double constraints," *Engineering Applications of Artificial Intelligence*, vol. 27, pp. 148–154, 2014.

[19] M. A. Nishi, C. F. Ahmed, M. Samiullah, and B.-S. Jeong, "Effective periodic pattern mining in time series databases," *Expert Systems with Applications*, vol. 40, no. 8, pp. 3015–3027, 2013.

[20] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han, "Comine: efficient mining of correlated patterns," in *Proceedings of the ICDM*, vol. 3, pp. 581–584, Melbourne, FL, USA, November 2003.

[21] S. Ma and J. L. Hellerstein, "Mining partially periodic event patterns with unknown periods," in *Proceedings 17th International Conference on Data Engineering*, pp. 205–214, IEEE, Heidelberg, Germany, April 2001.

[22] U. Suvarna and Y. Srinivas, "Efficient high-utility itemset mining over variety of databases: a survey,," in *Soft Computing in Data Analytics*, pp. 803–816, Springer, Berlin, Germany, 2019.

[23] M. J. Zaki and C.-J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 462–478, 2005.

[24] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and H.-J. Choi, "A framework for mining interesting high utility patterns with a strong frequency affinity," *Information Sciences*, vol. 181, no. 21, pp. 4878–4894, 2011.

[25] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and H.-J. Choi, "Interactive mining of high utility patterns over data streams," *Expert Systems with Applications*, vol. 39, no. 15, pp. 11979–11991, 2012.

[26] J. Venkatesh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic-correlated patterns in temporal databases," in *Transactions on Large-Scale Data-And Knowledge-Centered Systems XXXVIII*, pp. 146–172, Springer, Berlin, Germany, 2018.

[27] H.-W. Wu and A. J. Lee, "Mining closed flexible patterns in time-series databases," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2098–2107, 2010.

[28] P. Tzvetkov, X. Yan, and J. Han, "Tsp: mining top-k closed sequential patterns," *Knowledge and Information Systems*, vol. 7, no. 4, pp. 438–457, 2005.

[29] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering partial periodic itemsets in temporal databases," in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, pp. 1–6, Chicago IL USA, June 2017.

[30] P. Fournier-Viger, C.-W. Lin, Q.-H. Duong et al., "Pfpm: discovering periodic frequent patterns with novel periodicity measures," in *Proceedings of the 2nd Czech-China Scientific Conference 2016*, pp. 27–38, IntechOpen, Ostrava, Czech Republic, June 2017.

[31] M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, "Warp: time warping for periodicity detection," in *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, p. 8, IEEE, Houston, TX, USA, November 2005.

[32] R. Yang, W. Wang, and P. S. Yu, "Infominer+: mining partial periodic patterns with gap penalties,," in *Proceedings of the 2002 IEEE International Conference on Data Mining*, pp. 725–728, IEEE, Maebashi City, Japan, December 2002.

[33] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 21–36, 2005.

[34] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proceedings of the Eighth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, pp. 429–435, Edmonton, Canada, July 2002.

[35] C. Berberidis, W. G. Aref, M. Atallah, I. Vlahavas, and A. K. Elmagarmid, "Multiple and partial periodicity mining in time series databases," in *Proceedings of the 15th Eureopean Conference on Artificial Intelligence*, vol. 2, pp. 370–374, Lyon, France, July 2002.