IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 July 2023, accepted 17 August 2023, date of publication 25 August 2023, date of current version 31 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3308820

RESEARCH ARTICLE

Finding Partial Periodic and Rare Periodic Patterns in Temporal Databases

K. JYOTHI UPADHYA[1], AMAN PALEJA[1], M. GEETHA[1], (Member, IEEE), B. DINESH RAO[2], AND MINI SHAIL CHHABRA[1]

[1]Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India
[2]Manipal School of Information Sciences, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

Corresponding author: B. Dinesh Rao (dinesh.rao@manipal.edu)

ABSTRACT Most of the periodic pattern mining algorithms extract fully periodic patterns by strictly monitoring the cyclic behaviour of patterns in transactional as well as temporal databases. The most recent and preferred method for discarding non-periodic uninteresting patterns is partial periodic pattern mining, which has control over the strictness measure on cyclic repetitions of patterns. Recently, a variety of industries, including fraud detection, telecommunications, retail marketing, research, and medical have found applications for rare association rule mining, which uncovers unusual or unexpected combinations. A limited amount of literature demonstrated how periodicity is essential in mining low-support rare patterns. However, time of occurrence is also a vital phrase that is ignored which further aids in significant information retrieval. With this inspiration, a novel depth-first search framework named *3P-BitVectorMiner,* is proposed to extract entire partial periodic patterns from a temporal database. Experiments are carried out by varying support and periodicity thresholds for a variety of datasets. It is found that *3P-BitVectorMiner* consistently displays greater performance over the state-of-the-art algorithm *3P-Growth.* Further, the scalability of the *3P-BitVectorMiner* algorithm is also presented to demonstrate the efficiency over the *3P-Growth* algorithm on large temporal databases. In addition, two variations named *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* are proposed to mine rare fully periodic patterns and rare partial periodic patterns from temporal databases respectively. Different experiments carried out show that these proposed frameworks successfully capture periodic rare patterns in temporal databases.

INDEX TERMS Periodic pattern mining, partial periodic pattern mining, rare periodic pattern mining, partial periodic patterns, bit-vector representation.

I. INTRODUCTION

The aim of Pattern mining, a key component of data mining, is to extract valuable information from a vast volume of data. The most researched area of pattern mining is Frequent Pattern Mining(FPM), which extracts often recurring patterns from a dataset. Rare Pattern Mining(RPM), on the other hand, extracts hidden, unusual, yet valuable information from the set of transactions. However, one significant drawback of these techniques is that the pattern's occurrence behaviour is not taken into account. Additionally, the information regarding the time at transaction occurred is also completely ignored. ''Periodic Frequent Pattern Mining''(PFPM), has come up as a promising area that studies the occurrence characteristics of the itemsets. An inter-arrival time of an itemset is said to be periodic (or cyclic) if it is not greater than a periodic measure considered. Tanbeer et al. [1] have first demonstrated the significance of taking regularity into account in a static database. With its expansion, it is now employed in a variety of applications, including the analysis of gene and medical data [2], [3], mobility intention analysis [4], website user behaviour analysis [5] and so on. The periodic behaviour is closely checked in the early *PFPM* works using any of the user-specified periodicity measures like maximum periodicity [1], [6], [7], variance [8], [9], multiple periodicity measures [10] and lability [11], [12],

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Fiumara.

among others. The patterns extracted by these methods are called full periodic patterns. These patterns are selected only when all the periodicities confirm the threshold measure considered. However, in real-life applications, mining partial periodic patterns [13], [14] is also vital. Even though some events may happen regularly, some events occur only on weekends or at a specific instant of the day or maybe a particular day of the month. For example, heavy traffic is observed during weekends than weekdays. In supermarkets, customers can purchase milk and butter regularly. Whereas, rice is purchased monthly once. Partial periodic patterns relax the strictness measure by having a count on the requirement of the minimum number of cyclic repetitions. In addition, maintaining the temporal information gives still more insight into the knowledge discovery. For example, during weekends traffic congestion is observed more from 10 a.m. to 2 p.m. or 6 p.m. to 9 p.m. than at other timings. Therefore, it is important to preserve the occurrence time of transactions in temporal databases. Distinguished characteristics of temporal databases compared to transactional databases are:

- The transactions are sorted concerning their arrival time in ascending order.
- The arrival time of every transaction is not uniform.
- Multiple transactions may arrive at the same time.

It should also be noted that the conversion of temporal information to transactional information by merging the transactions with common time stamps should be avoided. As it leads to the following issues:

- The actual support of a pattern is lost when transactions with common time stamps are merged. In some cases, this may miss the required partial periodic patterns. For example, consider the sample temporal database presented in Table 1. It comprises 8 transactions with 5 distinct items. Here transactions T3 are T4 are having common time stamps. The merging of these transactions results in itemset {m,n,o,p,q}. This will cause a loss of actual support of items {n}, {o} and {p}. Further, as a result, these items may not be selected as partial periodic patterns.
- On the contrary, the merging of transactions may also create false associations leading to the generation of uninteresting partial periodic patterns. For example, merging transactions T3 and T4 will create an invalid association between items {m} and {q}.

Few studies have focused on extracting partial periodic frequent patterns from columnar databases [15], [16], partial frequent patterns [17], [18], partial frequent as well rare patterns [19] from temporal row databases.

Periodic Rare Pattern Mining has been emerging as a new promising area to discover hidden unexpected or unusual activities with their occurrence behaviour. The primary focus behind periodic rare pattern mining is the ability to discover uncommon or unexpected combinations that are missed by *PFPM* algorithms. If the rare patterns are evenly spread throughout the transaction dataset, they are periodic and significant.Very few algorithms have been designed to mine these patterns [20], [21]. However, studying the time stamp information will further enhance the knowledge discovered. For example, traffic congestion may be more at particular times during festival days. As festivals are happening rarely but throughout the year hence it is important to capture the information. Unlike periodic frequent patterns, these are the patterns comprising low support and larger periodicities. The occurrence of these events is not captured with a majority of the *PFPM* algorithms. With this motivation, the following major contributions are included in this paper:

- A novel algorithm named *3P-BitVectorMiner* is proposed to capture all required partial periodic patterns from a temporal dataset. When the pattern satisfies a user-given periodicity measure called *maxPer* it is treated as periodic(or cyclic). Here, the number of cyclic repetitions is counted and based upon the user-specified periodic support measure called *minPS* the partial patterns are selected.
- The temporal database is converted to a bit-vector form. *3PTSList* structure is created, which plays an important role in producing the one-length partial periodic patterns. Subsequently, it eliminates the non-periodic one-length patterns which reduces the huge search space. In contrast to the pattern growth method used in *3P-Growth* algorithm, here subsequent partial periodic patterns are generated with simple logical operations.
- As in real-life it is necessary to capture periodic rare patterns from the temporal database. In this paper, *3P-BitVectorMiner* is modified and two variations are proposed. First, *RFPP-BitVectorMiner* is proposed to mine rare fully periodic patterns. Second, *R3P-BitVectorMiner* extracts rare partial periodic patterns. These are the first algorithms, to our knowledge, that successfully capture periodic uncommon patterns in temporal databases. When the periodic support *minPS* threshold is set too low to extract periodic rare patterns, this results in numerous periodic patterns including both frequent as well as rare patterns. In addition, it will also generate a lot of spurious patterns. If *minPS* is set high, then it is unable to extract many rare periodic patterns. To overcome this issue, two different support thresholds *minFreqPS* and *minRarePS* thresholds are used along with *maxPer* to control the required number of cyclic repetitions. Here *minRarePS* assists in discarding the uncommon patterns that are associated by chance and are considered to be noisy itemsets.
- Many real-life as well as synthetic, sparse and dense type datasets are used for experimentation. Results show that *3P-BitVectorMiner* is faster, highly scalable and uses less memory compared to *3P-Growth.* Additionally, several analyses using different periodicities and support thresholds are shown for *RFPP-BitVectorMiner* and *R3P-BitVectorMiner.*

The remainder of the paper is arranged as follows: Section II discusses the related work done in the field

TABLE 1. Sample temporal database - TD.

TID	Time-stamp	Items
T1	1	m, n, o, p
T2	2	n, q
T3	4	m, n, o, p
T4	4	n, o, p, q
T5	5	m, o, p
T6	7	n, p
T7	7	m, n, o, p
T8	9	m, q

of periodic pattern mining. The basic definitions of the proposed algorithms are defined in section III. Different modules and illustrations of *3P-BitVectorMiner* are shown in Section IV. The two variations *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* are exhibited in Section V. Experimental evaluation and result analysis are presented in section VI. Section VII discusses the time complexity of proposed algorithms. Section VIII highlights the conclusion and future directions.

II. LITERATURE WORK

A. PERIODIC FREQUENT PATTERN MINING(PFPM)

A generalization of *FPM,* Periodic Frequent Pattern Mining(PFPM), addresses periodicity(occurrence behaviour or regularity). The literature review is carried out based on whether the time of occurrence is considered along with the periodicity threshold value.

1) RELATED WORK IN STATIC/STREAM DATASET

FP-Tree is a prominent data structure designed by Han et al. [22] that focuses on the enumeration of frequent patterns using the support count measure. The importance of considering the periodic behaviour of patterns is shown first in the work of Tanbeer et al. [1]. Here regularity behaviour is controlled by *maxPer* threshold measure. To support the regularity computation Regular-Pattern tree is constructed, where the transaction ids are stored only in the leaf nodes. The work is enhanced further to find regular patterns from stream data [6] and body sensor networks [7]. These algorithms discard itemsets even when a single periodicity fails to satisfy the *maxPer* threshold measure. To overcome this drawback, several other models are designed with different periodic measures. Rashid et al. [8] extracted regularly frequent patterns from static data by utilizing both support count as well as variance measures for finding frequency and regularity respectively. This work is modeled in the field of wireless sensor networks to find regular frequent sensor patterns [9]. The ''rare item problem'' is addressed by Kiran et al. [23], [24], [25] with multiple support and periodicity thresholds to extract frequent as well as rare frequent regular patterns from a static database. Fournier-Viger et al. [11] proposed a novel measure named lability to mine the stable periodic patterns from the database. A flexible method named *"Periodic Frequent Pattern Miner"* is designed by Fournier-Viger et al. [10]. The combination of minimum, maximum and average periodicity threshold measures is used here to discover entire frequent periodic patterns from the given set of transactions. To avoid setting the tedious task of occurrence frequency measures several models have come up with discovering top-k regular frequent patterns. Amphawan et al. [26] developed a single scan algorithm to discover top-k frequent regular patterns based on partition and estimation techniques. To deal with the stream data, this work is improvised and named *TFRIM-DS.* This is a single-pass algorithm that makes use of the sliding window technique to mine top-k regular itemsets having the highest support in the stream data. *TSPIN,* a model designed by Fournier-Viger et al. [12] constructs stable periodic-frequent tree and employs pattern growth approach to mine topmost-k frequent stable periodic patterns. The regularity concept is also introduced in mining high-utility itemsets. An efficient single-scan algorithm called *MHUIRA,* is the contribution of Amphawan et al. [27]. This algorithm finds regularly co-occurring items with high utility values. In addition, *HUIIs-Miner* [28] is designed to extract infrequently purchased itemsets with high profits. *PHM* a contribution of Fournier-Viger et al. [29], uses the minimum and average periodicity measures to extract high-utility periodic patterns from the static dataset.

These methods are found to be too strict as patterns are pruned even if one of the periodicity is not satisfying the considered periodicity measure. To overcome this strict behaviour, partial periodic pattern mining algorithms are developed. The partial periodic mining algorithms take the itemsets into account even though some of the periodicities do not satisfy the periodicity measure. The number of cyclic repetitions is controlled by the minimum periodic support count threshold. Kiran et al. [13] designed *GPF-growth* algorithm to extract a complete collection of partial periodic frequent patterns from a dataset. A novel periodic-ratio measure is utilized which considers the proportion of cyclic repetitions of frequent itemsets in databases. Venkatesh et al. [14] found a solution to the rare-item problem by discovering new measures named all-confidence and periodic-all-confidence. A pattern growth method *"Extended Periodic-Frequent pattern-growth"* is proposed to enumerate frequent patterns involving both frequent as well as rare periodicity.

2) RELATED WORK IN TEMPORAL DATASET

The characteristics of temporal transactions are non-uniform arrival time and multiple occurrences of transactions at a common time stamp. Initially, these features are handled by *3P-Growth* algorithm designed by Kiran et al. [17], [18]. Here, 3P-list and 3P-tree data structures are designed which are used to store temporal information instead of storing tid information. *3P-Growth* mining method enumerates all the partial periodic patterns by considering inter-arrival time information. To extract the patterns comprising both rare as well as frequent items in non-uniform temporal databases, Kiran et al. [19] proposed a novel measure named relative

periodic support. In the initial phase, the temporal dataset is compressed into *G3P-tree* and further from this tree *G3P-growth* algorithm recursively extracts an entire set of partial periodic patterns. To extract periodic patterns with minimum cyclic repetitions while showing non-uniform periodic nature, Kiran et al. [30] introduced a relative periodic-support measure. The periodic pattern growth method is used which mines the periodic pattern tree to capture periodic patterns in the non-uniform temporal database. The rare-item problem solution used for transactional data is enhanced by Venkatesh et al. [31] and *"Extended Periodic-Correlated pattern-growth"* method proposed which is able to mine frequent patterns that are correlated periodically. Few algorithms are developed to deal with columnar databases. Given a columnar temporal database, *"Frequent-Equivalence CLass Transformation"* is a run-time and memory-efficient method presented by Ravikumar et al. [16] to enumerate periodic-frequent patterns. Further, *3P-ECLAT* a depth-first search framework is designed by Ravikumar et al. [15] where initially, one-length partial periodic patterns are generated by storing time-stamp in a list named TS-list. Next, an intersection operation is performed on the TS-list and entire partial periodic patterns existing in the temporal database are generated. To tackle memory, runtime and energy, Likitha et al. [32] developed *max3P-Growth* to extract maximal partial periodic patterns from the temporal dataset. In addition, to avoid the tedious task of setting *minSup* threshold value, Likhitha et al. [33] designed *"Top-k Periodic-Frequent Pattern Miner"*. This model accepts a threshold value k and it presents all k frequent periodic patterns having the least periodicity value in a temporal dataset. Further, Likhitha et al. [33] contributed *SPP-ECLAT* method to extract the periodic-frequent patterns that are stable in a temporal dataset represented in vertical format.

B. RARE PATTERN MINING

1) RELATED WORK WITHOUT CONSIDERING PERIODICITY MEASURE

To mine rare itemsets and non-present itemsets, Adda et al. [34] developed *ARANIM* algorithm. It follows a top-down approach by starting from a k-itemset consisting of all items in the database. Subsequently, the subsets of the k-itemset are repeatedly produced and at every level, many non-existent patterns are generated and pruned. Rarity algorithm, a novel level-wise top-down strategy created by Troiano et al. [35], [36]. In the beginning, the longest itemset is found and its level-wise subsets are found subsequently. The advantage of this approach is that repeated database scanning is prevented. On the contrary, the memory requirements are increased to maintain different list structures. To extract the minimal rare itemsets(MRIs) which lie in the negative border of the frequent itemset zone, Szathmary et al. [37] designed *Apriori-Rare* and *MRG-Exp*. Further, *ARIMA* algorithm, [38] mines all rare itemsets from already discovered MRIs. The database is scanned multiple times to compute the support which degrades the performance. Bhasker and Yahia [39], [40], utilized the *bond* threshold along with support measure to handle the spurious rare itemsets extracted during mining of low support threshold value. At first, *CORI* algorithm transforms the input database into its vertical bit-wise format. This helps in performing simple logical operations to compute disjunctive as well as conjunctive support of the itemsets. Further, entire rare correlated patterns are mined in a bottom-up fashion with the help of this metric. To handle both frequent as well as rare patterns Borah and Nath [41] constructed *SSP*(Single Scan Pattern Tree). At first, the transactions are sorted in non-ascending order of support count and a compact tree is built by inserting it into the tree. During the tree building, if any path of the tree deviates from the support count descending order then the path is re-arranged. To store the static data in the main memory, Rai et al. [42], proposed a compact tree structure called the Binary count tree(BIN-Tree). An efficient mining technique is proposed to extract both rare and frequent itemsets. Lu et al. [43] contributed *NII-Miner,* the first tree-based method to discover all rare itemsets using a top-down depth-first strategy. This method considers the dual perspective of the original database by the representation of negative items.

To minimize the search space, some of the existing *RPM* algorithms mine subsets of rare patterns in a bottom-up fashion. These algorithms found rare 1-itemsets along with their supersets from those transactions which comprise one rare item at least. Rare Pattern Tree(RP-Tree), a compact tree structure, is a contribution by Tsang et al. [44] which is similar to the FP-Tree structure [22]. Pattern growth approach is utilized to extract only those rare itemsets that fall in between *minFrepSup* and *minRareSup* threshold values. Similarly, to mine subsets of rare patterns, Borah and Nath [45] proposed Hyper-Linked Rare Pattern Mining, a memory-based queue data structure with the hyper-linked pattern. Algorithms proposed in [46] and [47], discovered minimal rare itemsets using the bottom-up approach which traverses through several frequent itemsets to reach the minimal rare itemsets in the lattice.

2) RELATED WORK CONSIDERING PERIODICITY THRESHOLD

Periodicity plays a vital role in discovering significant rare patterns in a wide variety of applications. *MRCPPS* is a contribution by Fournier-Viger et al. [20] to extract periodic rare correlated itemsets from multiple sequences. With the support threshold, the standard deviation of periods is used as the periodicity measure. Along with these thresholds, the bond measure is utilized to filter the bundle of spurious patterns generated in the process of extracting useful periodic rare correlated itemsets. *PRCPMiner,* is our novel contribution which is able to discover periodic rare correlated patterns [21]. Here *CORI* algorithm is enhanced by using three different threshold measures regarding support, periodicity and bond measures.

As the literature shows very less *RPM* algorithms have concentrated on finding the occurrence behaviour of rare patterns. In addition, the temporal information is not at all considered in most of the existing *RPM* works. To examine the periodic behaviour of partial patterns in the temporal database, *PRCPMiner* is modified and a novel algorithm called *3P-BitVectorMiner* is proposed. Further, two variations named *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* were presented to study the occurrence behaviour of rare patterns in the temporal dataset.

III. PERIODIC PATTERN MODEL

A complete collection of k unique data items is represented as D = {d_1, d_2,…., d_k}. A sample collection of data items $P \subseteq D$ is called a pattern. A pattern comprising of c unique items, where $c \geq 1$ is named as a c-pattern. A temporal dataset *TD* over *D* is an ordered group of transactions, i.e., TD = {t_1, t_2,…..,t_x} where $x \geq 1$ represents total number of transactions and the database size is represented as |TD|. A temporal transaction comprising three fields, t_x = (tid,ts,P) where *tid* presents unique transaction identifier, *ts* represents time-stamp and *P* denotes a pattern. Let ts_{min} and ts_{max} represent the lower and upper time-stamp values in TD respectively. It can be observed from Table 1, two transactions can occur in the same time-stamp and there can be a delay between the two consecutive time-stamps. As a result (ts_{min} - ts_{max} + 1) may not represent |TD|. This shows that the temporal dataset may represent transactional dataset but not vice versa. The time-stamp of a pattern $Q \in P$ can be expressed as ts^Q if it appears in a transaction t_x = (tid,ts,P). Let $TS^Q = \{ts_i^Q, ts_j^Q, \ldots, ts_n^Q\}$ where i≤j≤n is used to signify the ordered time-stamps in which Q appears in TD. Support of Q is the number of transactions in which Q appears in TD and is indicated as Sup(Q)= $|TS^Q|$.

Example 1: Table 1 comprises total 8 transactions. Hence x and |TD| is 8. The data items set D = {m,n,o,p,q}. The first transaction t_1 = (T1, 1, mnop) where *T1* represents *tid.* Time-stamp *ts* is 1 and the set {m,n,o,p} is having 4 items termed as a 4-pattern. Here ts_{min} and ts_{max} ranges from 1 to 9. Assume that the pattern {mo} appears in transactions with time-stamp values 1, 4, 5 and 7. Therefore TS^{mo} = {1,4,5,7} and Sup(mo) is $|TS^{mo}|$ which results in 4.

Definition 1 (Periodicity of pattern $\boldsymbol{Q}$*):* Let (ts_i^Q, ts_j^Q) be a pair of consecutive time-stamp in TS^Q. An inter-arrival time is the time difference between ts_i^Q and ts_j^Q and it is denoted as $iat^Q = ts_j^Q - ts_i^Q$. Let list of inter-arrival times of Q in TD is represented as $IAT^Q = \{iat_1^Q, iat_2^Q, \ldots, iat_s^Q\}$ where s = Sup(Q) - 1. When the inter-arrival time of Q is within the user-given maximum periodicity threshold i.e. $iat_i^Q \leq maxPer$ then it is considered as **periodic**.

Definition 2 (Periodic support of pattern $\boldsymbol{Q}$*):* Let $\widehat{IAT^Q}$ denote the group of all inter-arrival times that are periodic. Therefore, $\widehat{IAT^Q} \subseteq IAT^Q$ such that when $\exists iat_i^Q \in IAT^Q$ and $iat_i^Q \leq maxPer$, then $iat_i^Q \in \widehat{IAT^Q}$. The periodic support of Q is denoted as PS(Q) and it is equal to $|\widehat{IAT^Q}|$. The proposed measure considers inter-arrival time as well as support in the database while selecting the pattern.

Example 2: The initial time-stamps of pattern {mo} is 1 and 4 resulting in its first inter-arrival time iat_1^{mo} = 3(4-1). Similarly, the remaining inter-arrival times are computed which results in IAT^{mo} = {3,1,2}. If the user specifies *maxPer* threshold value as 2 then $\widehat{IAT^{mo}}$ ={1,2} resulting PS(mo) = 2.

Definition 3 (Partial periodic pattern $\boldsymbol{Q}$*):* Given the user-given minimum period support threshold *minPS,* an itemset Q is considered to be partial periodic if PS(Q) ≥ *minPS.*

Example 3: For the user specified *minPS* value 2, itemset {mo} results as a partial periodic pattern.

Problem Definition Given a temporal database *TD,* a support threshold *minPS* and a periodicity threshold *maxPer,* the task of extracting partial periodic pattern is to find the entire collection of patterns with periodic support not less than *minPS.*

The partial periodic patterns(PPPs) enumerated by the proposed model satisfy the downward closure property. The correctness is illustrated with the help of Lemma 1 and is based on Property 1.

Property 1: If $R \subset Q$, then $TS^R \supseteq TS^Q$ and it results in PS(R) ≥ PS(Q).

Lemma 1: If Q is a partial periodic pattern, then ∀R ⊂ Q and R ≠ ∅, is also a partial periodic pattern.

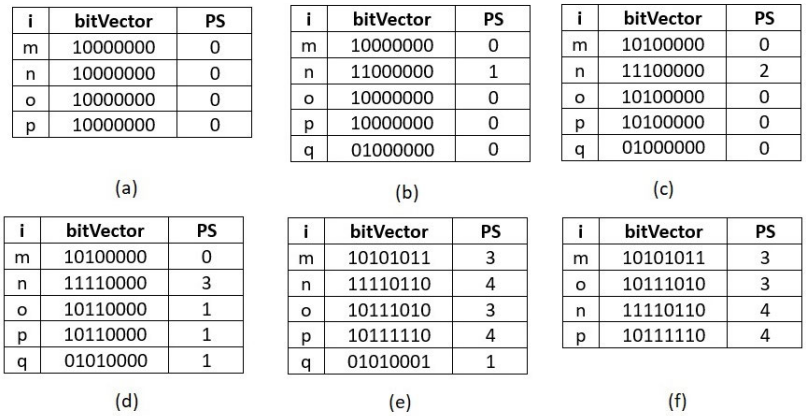
Proof: If PS(Q) ≥ *minPS* then Q is a partial periodic pattern, with respect to Definition 3. Then PS(R)≥PS(Q)≥minPS based on Property 1. Therefore, R is also a partial periodic pattern.

IV. 3P-BitVectorMiner(PARTIAL PERIODIC PATTERN BIT VECTOR MINER): THE PROPOSED ALGORITHM

3P-BitVectorMiner is a novel method to extract an entire collection of partial periodic patterns(PPPs) from the given temporal dataset *TD.* Complete set of *PPPs* are discovered in two steps: (i) Transform *TD* into a bit-vector form and produce one length PPPs which are maintained in a *3PTSList* structure (ii) Construct *3PTSTree* and recursively traverse *3PTSTree* in Depth First Search(DFS) method to extract complete set of *PPPs* by discarding non-periodic patterns during the mining task.

A. TRANSFORM THE DATASET INTO BIT-VECTOR FORM AND GENERATE ONE LENGTH PPPs

In the initial stage, the temporal database *TD* is scanned and each item is converted into a bit-vector form. Each bit in the bit-vector represents consecutive temporal transactions where the presence of an item is indicated by '1' and absence by '0'. A *3PTSList* structure which stores the bit-vector of every item is simultaneously created. *3PTSList* maintains two fields related to an item *i* : bit-vector of item i - *bitVector(i)* and periodic support of item *i* - *PS(i).* As every transaction is transformed into a bit-vector form, the *3PTSList* structure is updated for all the items appearing in the transactions. Along with the modification of bit-vector, the periodic



i	bitVector	PS
m	10000000	0
n	10000000	0
o	10000000	0
p	10000000	0

(a)

i	bitVector	PS
m	10000000	0
n	11000000	1
o	10000000	0
p	10000000	0
q	01000000	0

(b)

i	bitVector	PS
m	10100000	0
n	11100000	2
o	10100000	0
p	10100000	0
q	01000000	0

(c)

i	bitVector	PS
m	10100000	0
n	11110000	3
o	10110000	1
p	10110000	1
q	01010000	1

(d)

i	bitVector	PS
m	10101011	3
n	11110110	4
o	10111010	3
p	10111110	4
q	01010001	1

(e)

i	bitVector	PS
m	10101011	3
o	10111010	3
n	11110110	4
p	10111110	4

(f)

FIGURE 1. (a) After scanning tid = 1 (b) After scanning tid = 2 (c) After scanning tid = 3 (d) After scanning tid = 4 (e) After scanning all transactions (f) After discarding non-periodic one length items and final sorted *3PTSList*.

support values are updated in the *3PTSList* as shown in Algorithm 1. The transaction id, tid_{cur}, is considered to have continuous values beginning from 1. As shown in line 3, the current time-stamp information ts_{cur} is stored in an array *TSList* and it is used as common time-stamp information for all the items. Using the bit-vector representation all the occurrence information along with the previous occurrence information can be retrieved. And also the Sup(i) can be calculated any time using *bitVector(i)*. This shows that like *3P-Growth* [17] support and previous time-stamp information need not be maintained for each item. Lines 6 and 7 show how the time-stamp value can be acquired by extracting the information from the *TSList* array for the required bit of *bitVector(i)*. As observed in line 8, the current periodicity is computed by subtracting the current time-stamp from the previous time-stamp obtained from array *TSList*. The periodic support *PS* value of the current item *i* is incremented by one if the resultant periodicity value is not greater than *maxPer* threshold. Finally, once the entire database is scanned, the non-periodic one length items *i* with PS(i) not greater than *minPS* value are eliminated from *3PTSList*. For the database shown in Table 1, the *3PTSList* creation and updation are as shown in Fig. 1. Initially, after reading the first transaction the bit-vectors created for items *m, n, o,* and *p* are shown in Fig. 1(a). As this is the first appearance of items the *PS* values are zero. Fig. 1(b) shows the updated bit-vectors after reading the second transaction. As it can be observed only item *n* has occurred in consecutive transactions, its period difference is calculated. From the bitVector of item *n* the last bit set to 1 was for tid 1 which is the resultant value extracted by *lastSetTid*. Now from the common time-stamp array *TS* the previous time-stamp for item *n* is extracted and used in the current periodicity calculation. Here *curPrd* results in 1(2-1). According to the *maxPer* this is periodic and therefore *PS* value is incremented by 1. These steps are shown in lines 6 to 10 of Algorithm 1. Similar way the remaining transactions are read and the *PS* values are updated as shown in Fig. 1(c) and (d). After the complete database has been scanned, the non-periodic items of one length are removed

Algorithm 1 GetPartialPeriodicOneLengthItems(*TD* - a Temporal Database, *maxPer* - Maximum Periodicity Threshold and *minPS* - Minimum Period support)

1: Initialize bitVector(i) and PS(i) to 0 where *i* represents an item
2: **for each** transaction T ∈ TD with transaction id tid_{cur} and time-stamp ts_{cur} **do**
3: Set TS[tid_{cur}] ← ts_{cur}
4: **for each** item i ∈ T **do**
5: Set tid_{cur} bit of bitVector(i) as 1
6: Let *lastSetTid* represent the tid of last bit set of bitVector(i)
7: Set *prevTS* ← TS[lastSetTid]
8: Compute current periodicity *curPrd* by subtracting the *prevTS* with ts_{cur}.
9: **if** *curPrd* ≤ *maxPer* **then**
10: Increment periodic support PS(i) by 1
11: **end if**
12: **end for each**
13: **end for each**
14: Remove all non-periodic item *i* from *3PTSList* having PS(i) < *minPS* and Sort remaining items in *3PTSList* in ascending order of Support Count
15: Save all one length *PPPs* to *3POutputList*

and the remaining partial periodic 1-itemsets are sorted by their support value into ascending order. Fig. 1(e) shows the final *3PTSList* after discarding the non-periodic 1-itemset {q} and sorting the remaining 1-itemsets in ascending order with respect to the periodic support *PS* . These one length *PPPs* are also saved into the *3POutputList*.

B. CONSTRUCT 3PTSTree *AND RECURSIVELY TRAVERSE* 3PTSTree *IN DEPTH FIRST SEARCH(DFS) METHOD TO EXTRACT COMPLETE SET OF* PPPs

All the one length *PPPs* generated after the first scanning are sorted in ascending order. The first layer of *3PTSTree* is constructed with the sorted *PPPs*. Further, the supersets

Algorithm 2 3P-BitVectorMiner(*3PTSList*)

for each item *i* in *3PTSList* **do**
  Set *pi* ← ∅ and k ← *i*
  for each item *j* that comes after item *i* in *3PTSList* **do**
    Set δ ← k ∪ j and Generate bitVector(δ) ← bitVector(k) ∧ bitVector(j)
    Set PS(δ) = CalculatePeriodSupport(bitVector(δ))
    if PS(δ) ≥ *minPS* **then**
      Store PS(δ) to *3POutputList* and Add δ to *pi*
    end if
  end for each
  3P-BitVectorMiner(pi)
end for each
Procedure CalculatePeriodSupport(bitVector)
Set PS = ∅
for each bit *b* in bitVector **do**
  Let x and y represent the tid of ordered bits that are 1 in bitVector
  Compute current periodicity *curPrd* by subtracting the *TS[x]* with *TS[y]*.
  if *curPrd* ≤ *maxPer* **then**
    Increment PS by 1
  end if
end for each
Return PS
end procedure

of one length *PPPs* are generated as shown in Algorithm 2. Every item *i* will be considered and in each iteration item *j* that has a higher *PS* value than *i* will be considered. The initial stage generates a superset of item *i* and *j* named as δ. Next, *an AND* operation is performed on the *bitVectors* of *i* and *j* to generate *bitVector(δ)*. Further, the period support of δ is calculated as shown in Procedure *CalculatePeriodSupport*. Let *x* and *y* represent tid's of two continuous bits that are 1 in *bitVector(δ)*. The *curPrd* is computed by subtracting the timestamps corresponding to *x* and *y*. If the *curPrd* is ≤ *maxPer* then *PS* is incremented by 1. This task is repeated for all the bits of *bitVector(δ)*. Finally, if the resultant *PS* value is ≥ *minPS* threshold value then itemset δ is partially periodic and is saved in to *3POutputList*. Further, when the child nodes produce partial periodic itemsets, the DFS method is continued in that path to generate all the remaining *PPPs*. In every other case, non-periodic itemsets are pruned. Their children will not be traversed as there is no chance of generating any *PPPs*. The *PPPs* generated for item {m} are shown in the Figure 2. Since itemset {mo} and itemset {mp} both results as *PPPs,* they are saved into *3POutputList*. The DFS traversal continues further to extract superset {mop} which is also found to be a partial periodic pattern and saved into *3POutputList*. Whereas, Itemset {mn} is pruned because it is non-periodic and traversal along that path is not commenced. Since item {m} doesn't have any more supersets, the process is then repeated by considering item {o}. This task is repeated for all the pending one length

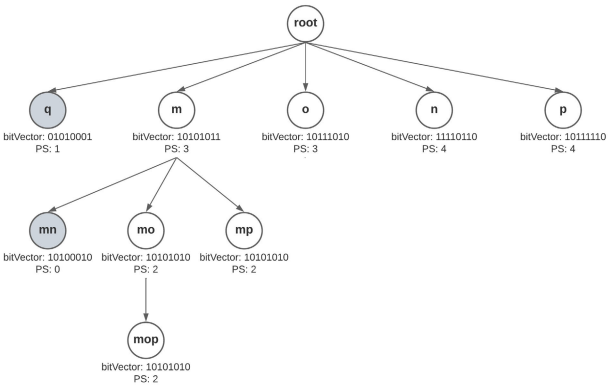


FIGURE 2. **Resultant *3PTSTree*** after the DFS traversal of item {m}.

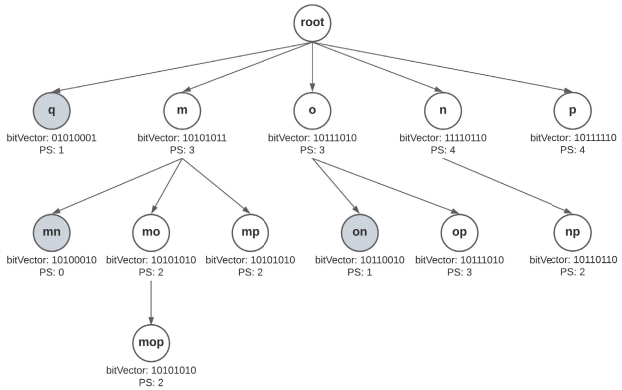


FIGURE 3. **Resultant *3PTSTree*** after the DFS traversal of all items.

PPPs in the first layer of *3PTSTree*. The final resultant *3PTSTree* after the DFS traversal of all items is presented in Figure 3. As the proposed method of finding *PPPs* follows downward closure property, this reduces the search area and results in improving the mining performance.

V. RARE PERIODIC PATTERN MINING

In many real-life situations, it is necessary to extract rare patterns whose presence is throughout the database. When the period support *minPS* threshold is kept too low to extract rare patterns, this results in numerous periodic patterns including both frequent as well as rare patterns. In addition, it will also generate a lot of spurious patterns. If *minPS* is set high, then it is unable to extract many rare patterns. To overcome this issue, two different support thresholds *minFreqPS* and *minRarePS* thresholds are used along with *maxPer* threshold to control the number of cyclic repetitions. Here *minRarePS* assist in discarding the uncommon patterns that are associated by chance and are considered to be noisy itemsets. Based on the strictness of periodicity measure rare periodic patterns can be classified as full and partial periodic patterns.

*Definition 4 (Rare fully periodic pattern **Q**):* Given the user-specified minimum period support thresholds *minFreqPS* and *minRarePS,* a pattern *Q* is said be rare fully periodic pattern(RFPP) if ((PS(Q) < *minFreqPS* ∧ PS(Q) ≥ *minRarePS)* ∧ (PS(Q) = Sup(Q) - 1)).

The Rare full periodic pattern measure is too strict and a pattern is discarded even when one inter-arrival time is also exceeding the *maxPer* threshold. As rare patterns show the tendency to behave non-periodic in certain time-period there is a need to propose a relaxed measure.

*Definition 5 (Rare partial periodic pattern **Q**):* Given the user-specified minimum period support thresholds *minFreqPS* and *minRarePS,* a pattern Q is said be rare partial periodic pattern(R3P) if (PS(Q) $<$ *minFreqPS* $\wedge$ PS(Q) $\geq$ *minRarePS).*

To handle Rare full and partial periodic patterns, *3P-BitVectorMiner* is modified and two variations are proposed. *RFPP-BitVectorMiner* is proposed to mine rare fully periodic patterns and *R3P-BitVectorMiner* extracts rare partial periodic patterns. Algorithm 1 is modified as follows in order to deal with rare patterns: When discarding non-periodic itemsets from the *3PTSList,* item Q with PS(Q) $<$ *minRarePS* are also discarded. Removing noise itemsets aids in minimizing the search space. All item Q with PS(Q) $\geq$ *minFreqPS* are retained as their supersets may have PS value $<$ *minFreqPS* $\wedge \geq$ *minRarePS.* Further, in Algorithm 2, line 6 is modified according to Definitions 4 and 5 to extract *RFPPs* and *R3Ps* respectively.

VI. EXPERIMENTAL RESULTS

A. EXPERIMENTAL SETUP

The proposed framework *3P-BitVectorMiner* accepts *minPS* and *maxPer* threshold values from the user and discovers all partial periodic patterns from the temporal database. As *3P-Growth* is the state-of-the-art algorithm which considers row temporal information, accepts the same inputs and generates *same number of PPPs* as *3P-BitVectorMiner.* Therefore, *3P-BitVectorMiner* is compared with *3P-Growth* [17] which is the state-of-the-art algorithm in mining *PPPs.* This section analyses how the approaches differ in terms of memory usage and execution time. The scalability test is also presented to demonstrate the performance of the *3P-BitVectorMiner* algorithm over the *3P-Growth* on large temporal datasets. Further, the two variations *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* is proposed to mine rare fully and partial periodic patterns respectively. To extract rare full and partial periodic patterns, the algorithms ask the user for two support thresholds *minFreqPS* and *minRarePS* as well as a periodicity threshold *maxPer.* Since these are the first algorithms for mining rare periodic patterns from temporal databases, the algorithms are tested on different datasets with varying threshold values for *maxPer, minFreqPS* and *minRarePS.* All the proposed algorithms are implemented in the Java platform and are tested in the system with configuration Intel(R) Core(TM) i5-7400 CPU@3.00GHz with 8GB RAM running Windows 10 Enterprise.

B. DATASETS

For the experimentation, three real and one synthetic datasets with a varying number of transactions are downloaded from the "frequent itemset mining dataset repository" (http://fimi.ua.ac.be/data/). *Mushroom* is a dataset with total transactions of about 8k whereas *Accidents* is a large dataset with more than 340k transactions. *Chess* is a small real-world dataset with about 3k transactions. *T20I6D100K* is the sparse synthetic dataset generated by the IBM data generator. *Pollution* is a real-world dense high dimensional dataset with long transactions which is taken from https://u-aizu.ac.jp/ udayrage/datasets.html. Both sparse and dense types of datasets are chosen and the descriptions of the dataset are given in Table 2.

C. EVALUATION OF 3P-BitVectorMiner AND 3P-GROWTH ALGORITHMS

By taking into account various datasets for varied maximum periodicity and minimum support threshold values, the run-time performance of the algorithms is determined. Fig. 4 displays the run-time comparison of *3P-BitVectorMiner* and *3P-Growth* for various thresholds. Here X-axis represents *minPS* threshold value keeping *maxPer* as constant. On the other side, if X-axis represents *maxPer* threshold value then *minPS* is kept as constant. Y-axis represents the run-time in seconds in these figures. It has been observed that *3P-BitVectorMiner* outperforms *3P-Growth* in all the cases.

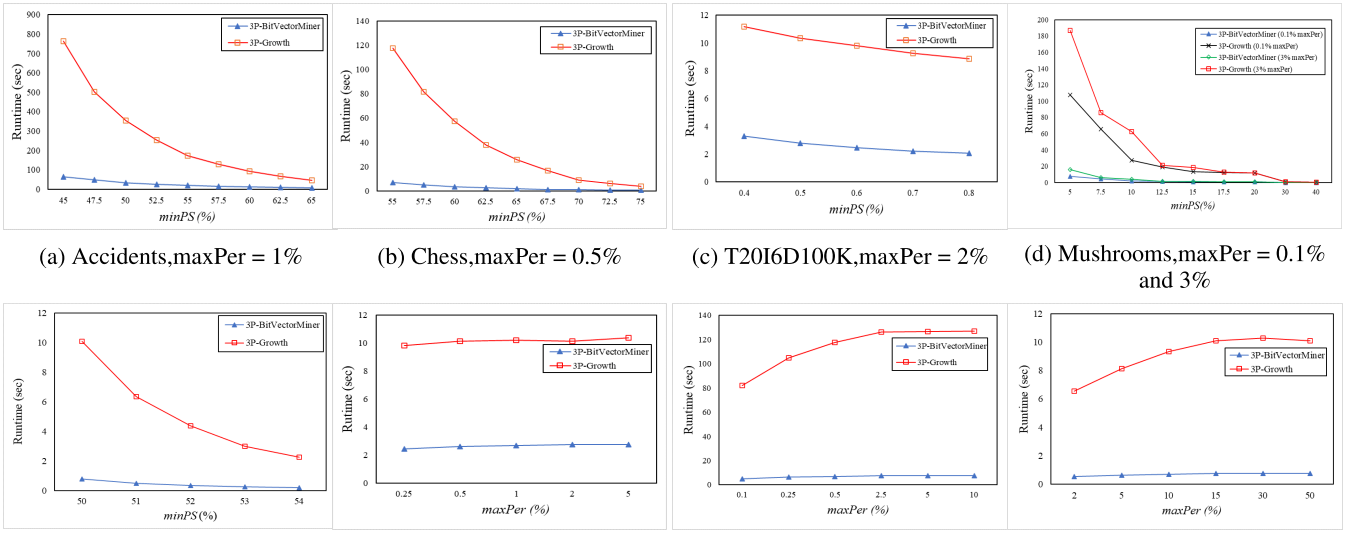
1) RUN-TIME PERFORMANCE

For the *Accidents* dataset, the *minPS* values are varied from 45% to 65% while keeping *maxPer* constant as 1% as shown in Fig. 4(a). Fig. 4(b) depicts the run-time performance of *Chess* dataset, where *maxPer* is set constant as 0.5% and *minPS* values are varied in the range of 55% to 75%. Compared to *3P-Growth, 3P-BitVectorMiner* has demonstrated a performance improvement of 88.87% and 92.38% in the case of *Accidents* and *Chess* dataset respectively. As shown in Fig. 4(c), for *T20I6D100K* dataset, *3P-BitVectorMiner* depicts a performance improvement of 74.2% when *minPS* values are changed from 0.4% to 0.8% and *maxPer* is kept constant as 2%. It is observed in Fig. 4(d), *3P-BitVectorMiner* shows a performance gain of 91.33% and 91.05% for *Mushrooms* dataset when *maxPer* thresholds set as 0.1% and 3% respectively. Here *minPS* is varied in the range of 5% to 40%. Fig. 4(e) shows the case of *Pollution* database where the *maxPer* is set constant as 15% and *minPS* is changed from 50% to 54%. In this case *3P-BitVectorMiner* has achieved a performance improvement of 91.98%. Also, it is noticed that algorithm takes a long time to run as *minPS* values are reduced further, especially when the dataset is large.

Fig. 4(g) depicts the total execution time taken by *Chess* dataset where the *maxPer* is varied from 0.1% to 10% while the *minPS* is set constant as 55%. It is observed that compared to *3P-Growth, 3P-BitVectorMiner* performance is improved by 94.04%. Similarly, *3P-BitVectorMiner* also depicted a performance improvement of 73.88% in the case of *T20I6D100K* as shown in Fig. 4(f). Here the *maxPer* is changed from 0.25% to 5% while the *minPS* is set constant as

TABLE 2. Statistics of Datasets.

S. No	Database	Type	Nature	Transaction Length			Database Size
				Min.	Avg.	Max.	
1	Accidents	Real	Dense	18	33.8	51	3,40,183
2	T20I6D100K	Synthetic	Sparse	1	19.8	47	99,922
3	Mushroom	Real	Dense	23	23	23	8,124
4	Chess	Real	Dense	37	37	37	3,196
5	Pollution	Real	Dense	11	459.2	971	717



(a) Accidents,maxPer = 1% (b) Chess,maxPer = 0.5% (c) T20I6D100K,maxPer = 2% (d) Mushrooms,maxPer = 0.1% and 3%

(e) Pollution, maxPer = 15% (f) T20I6D100K,minPS = 0.5% (g) Chess,minPS = 55% (h) Pollution,minPS = 50%

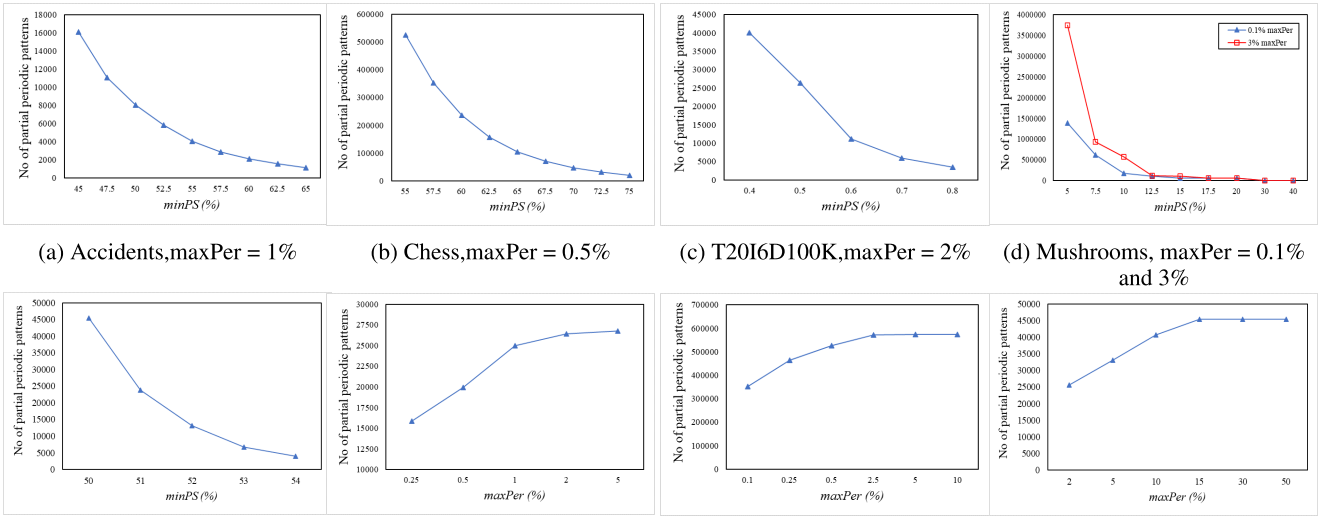
FIGURE 4. Runtime comparison on different datasets for varying ***minPS*** or *maxPer* threshold.

0.5%. Fig. 4(h) presents the case of *Pollution* database where the *minPS* is set constant as 50% and *maxPer* is changed from 2% to 50%. In this case *3P-BitVectorMiner* have achieved a performance improvement of 92.23%. However, increasing the *maxPer* value or even decreasing does not give any significant changes in the number of patterns generated in other cases.

Influence of *minPS* and *maxPer* threshold values: The number of *PPPs* generated for the different datasets and thresholds considered in Fig. 4 are shown in Fig. 5. The Figures emerged in the following key points: (i) It is evident that *minPS* threshold has a negative effect on the generation of *PPPs.*That is as *minPS* threshold is increased the collection of *PPPs* decreases and vice-versa. The reason behind this is more number of 1-itemsets fail to satisfy the increased *minPS* value. This further declines the number of *PPPs* generated. Obviously, execution time taken reduces as a lesser number of *PPPs* are generated. (ii) On the contrary, the *maxPer* has a positive effect on the generation of *PPPs.* As *maxPer* is increased, most of the non-periodic 1-itemsets will turn into partial periodic 1-itemsets. This further becomes the reason to increase the resultant *PPPs.* As Fig. 4 depicts, the run-time performance of *3P-BitVectorMiner* improves compared to *3P-Growth* with a increase in the number of *PPPs.* The major difference in the case of *3P-Growth* is the tree size grows when non-periodic 1-itemsets turn into partial periodic 1-itemsets. This further results in more recursive tree creation and increased mining time. Whereas, in *3P-BitVectorMiner,* these complex operations are replaced with the simple logical bit-wise operations playing the main contribution towards performance gain. iii) It is also noted that, in comparison to *maxPer, minPS's* alteration has a greater impact on the generation of *PPPs.*

2) MEMORY CONSUMPTION

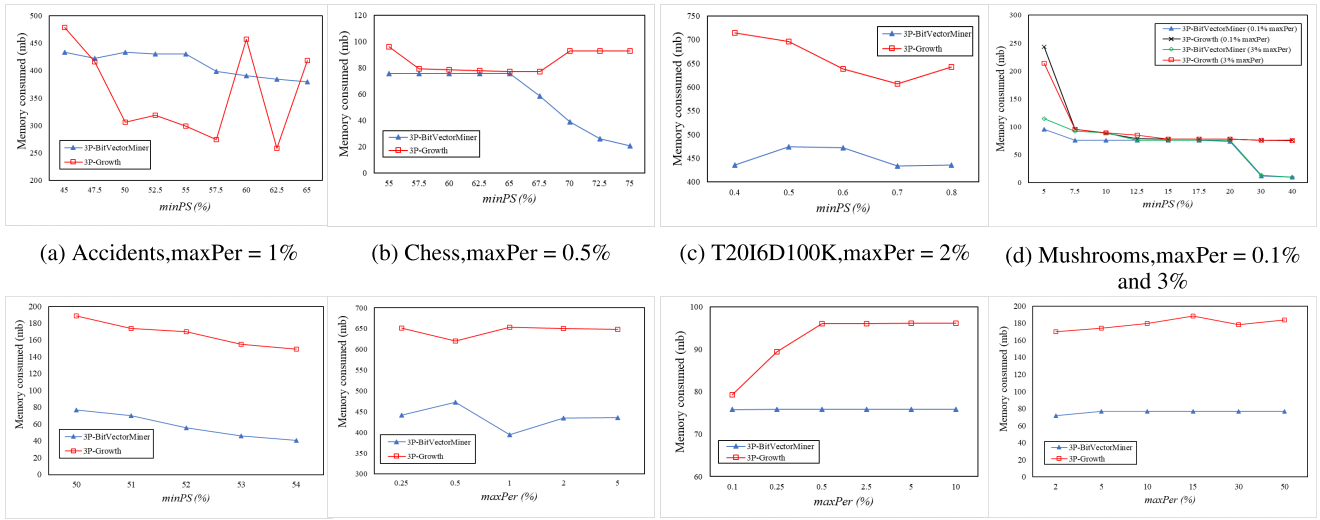
Fig. 6 displays the memory consumption details of both algorithms considering various datasets with different threshold values set as exhibited in Fig. 4. Fig. 6(b),(c),(d),(e) presents the memory utilized by both the algorithms when *minPS* is varied for *Chess, T20I6D100K, Mushroom* and *Pollution* datasets respectively. The memory usage details of *Chess, T20I6D100K* and *Pollution* datasets for *maxPer* variation is shown in Fig. 6(f),(g) and (h) respectively. In all these cases *3P-BitVectorMiner* consumes lesser memory compared to *3P-Growth.* The following observations are noted from the Figures: (i) The memory consumption reduces with the increase in *minPS* threshold value and vice versa. (ii) Conversely, as *maxPer* threshold increases memory usage is also increased. It demonstrates that when more number of non-periodic 1-itemsets change into partial periodic 1-itemsets, causes rise in the number of *PPPs* formed, increasing the need for memory. In *3P-BitVectorMiner* Partial periodic 1-itemsets are represented using bit-vectors and further the *PPPs* are extracted using logical operations consuming the same number of bits. Consequently, in *3P-Growth,* when



(a) Accidents,maxPer = 1% (b) Chess,maxPer = 0.5% (c) T20I6D100K,maxPer = 2% (d) Mushrooms, maxPer = 0.1% and 3%

(e) Pollution, maxPer = 15% (f) T20I6D100K,minPS = 0.5% (g) Chess,minPS = 55% (h) Pollution,minPS = 50%

FIGURE 5. **Number of *PPPs*** generated on different datasets for varying *minPS* or *maxPer* threshold.



(a) Accidents,maxPer = 1% (b) Chess,maxPer = 0.5% (c) T20I6D100K,maxPer = 2% (d) Mushrooms,maxPer = 0.1% and 3%

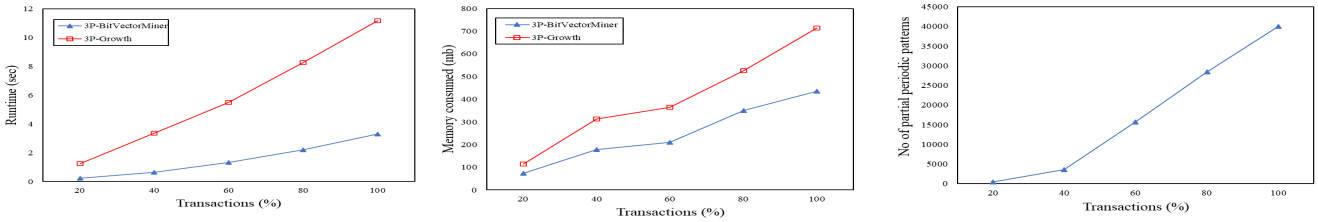
(e) Pollution, maxPer = 15% (f) T20I6D100K,minPS = 0.5% (g) Chess,minPS = 55% (h) Pollution,minPS = 50%

FIGURE 6. **Memory usage by different datasets for varying *minPS*** or *maxPer* threshold.

non-periodic 1-itemsets transform into partial periodic 1-itemsets, the tree size increases and further increases the memory need. This adds to the load of memory needed for the increased recursive tree constructions. *3P-BitVectorMiner* uses 31.52% less memory and 32.32% less memory for *T20I6D100K* dataset when *minPS* and *maxPer* are varied, respectively, as compared to *3P-Growth.* In the case of *Mushroom* dataset, 31.11% and 26.59% lesser memory is consumed by *3P-BitVectorMiner* when the *maxPer* is set as 0.1% and 3% respectively. The *Chess* dataset has the same characteristics as the *Mushroom* dataset. The memory consumption of *3P-BitVectorMiner* is reduced by 29.48% and 17.32% when *minPS* and *maxPer* is varied respectively. In the case of *Pollution* dataset, similar behaviour is observed. When *minPS* and *maxPer* is varied, *3P-BitVectorMiner* consumed 65.76% and 57.56% lesser memory compared to *3P-Growth*respectively. However, as shown in Fig. 6(a), only for *Accidents* dataset, *3P-BitVectorMiner* consumes 20.22% more memory space than *3P-Growth.* Even though *Accidents* is a large dataset, the number of *PPPs* produced is very less. Memory is consumed in finding and discarding non-periodic partial patterns.

3) SCALABILITY

The scalability operation performed here determines the effectiveness and productivity of *3P-BitVectorMiner* over *3P-Growth* on large temporal datasets. The *T20I6D100K* database, a sparse real-world large dataset, was employed in this experiment to carry out the scalability testing. The database is split into five equal sections for this experiment, each portion comprising around 20K transactions. The performance at each iteration is carried out by accumulating the



(a) Run-time (b) Memory Consumption (c) Number of *PPPs*

FIGURE 7. **Scalability of 3P-BitVectorMiner** and *3P-Growth* algorithms.

previous portion of transactions. The sparse dataset chosen shows the performance of algorithm as every portion consists of a different number of items. When *minPS* is set to 0.4% and *maxPer* is set to 2%, Fig. 7(a) and (b) displays the results for the runtime-performance and memory consumption levels of both the methods for various database sizes respectively. Fig. 7(c) depicts the corresponding resultant number of *PPPs* extracted by both the algorithms. The following are some significant points observed from these figures: (i) The resulting number of *PPPs* also rises as the database size increase. As a result, both algorithms' runtime and memory needs increase linearly. (ii) *3P-BitVectorMiner* shows a run-time performance improvement of 76.6% compared to *3P-Growth* algorithm. (iii) Compared to *3P-Growth, 3P-BitVectorMiner* consumes 38.77% lesser memory. The scalability test demonstrates that, with less runtime and memory needs, *3P-BitVectorMiner* could extract partial periodic patterns from massive temporal databases.

D. EVALUATION OF RFPP-BitVectorMiner AND R3P-BitVectorMiner

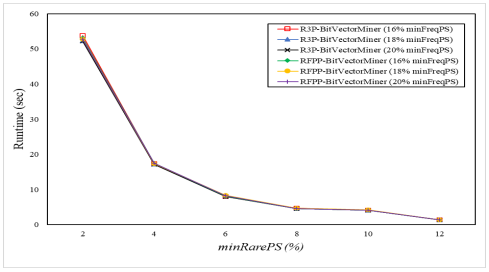
The two variations *RFPP-BitVectorMiner* and *R3P-BitVectorMiner,* uses *minFreqPS* and *minRarePS* thresholds, in order to restrict the retrieved patterns to rare patterns. While *maxPer* threshold makes sure that only periodic patterns are enumerated. As rare periodic patterns are patterns with low support and larger periodicity in comparison with frequent periodic patterns, the thresholds are set accordingly.

1) RUN-TIME COMPARISON ON DIFFERENT DATASETS FOR VARYING *minRarePS* KEEPING *minFreqPS* AND *maxPer* CONSTANT:

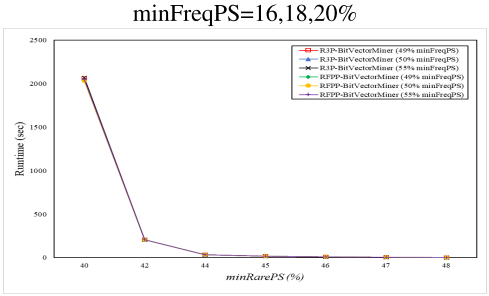
Fig. 8(a),(b) and (c) shows the run-time performance of *Mushroom, Pollution* and *T20I6D100K* dataset respectively. The Figures depict a significant impact on the execution time when *minFreqPS* and *maxPer* threshold kept constant while varying the *minRarePS* threshold. Threshold *minRarePS* is varied from 2 to 12%, 40 to 48% and 0.1 to 0.5% as shown in Fig. 8(a),(b) and (c) respectively. Whereas, *minFreqPS* is kept constant as 16,18 and 20% in Fig. 8(a), 49,50 and 55% in Fig. 8(b) and 0.6,0.7 and 1% in Fig. 8(c) respectively. The threshold *maxPer* is set as 2% in Fig. 8(c) and 15% in both Fig. 8(a) and (b). Fig. 10(a),(b) and (c) presents the corresponding *RFPPs* and *R3Ps* generated for these execution setups. The experiments resulted in the following significant outcomes: (i) As it can be observed from Fig. 9(a) and (b) the *minRarePS* variation has shown a negative effect on the count of resultant *RFPPs* and *R3Ps.* The decrease in *minRarePS* especially the low threshold accelerates the transformation of noisy itemsets to rare 1-itemsets. This increase in rare 1-itemsets results in a rise in the number of *RFPPs* and *R3Ps* produced for low *minRarePS* threshold values. As the number of *RFPPs* and *R3Ps* rises, the execution time also increases as shown in Fig. 8(a),(b) and (c). It is also observed that the number of *RFPPs* and *R3Ps* start varying for low *minRarePS* thresholds. However, as *T20I6D100K* is a sparse dataset, the number of *R3Ps* extracted are more compared to *RFPPs* as depicted in Fig. 9(c). (ii) On the other hand, the number of *RFPPs* and *R3Ps* increases very slowly with the rise in *minFreqPS* threshold value in the case of *T20I6D100K* sparse dataset as shown in Fig 8(c).

2) RUN-TIME COMPARISON ON DIFFERENT DATASETS FOR VARYING *minFreqPS* VALUE KEEPING *minRarePS* AND *maxPer* CONSTANT:

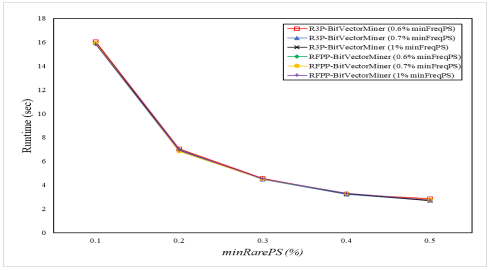
Fig. 10(a),(b) and (c) shows the total execution time taken by *Mushroom, Pollution* and *T20I6D100K* dataset respectively. The Figures present a significant impact on the run-time when *minFreqPS* is varied by keeping *minRarePS* and *maxPer* constant. Threshold *minFreqPS* is varied from 10 to 18%, 50 to 75% and 0.4 to 0.8% as shown in Fig. 10(a),(b) and (c) respectively. Whereas, *minRarePS* is kept constant as 4,6 and 8% in Fig. 10(a), 46,47 and 48% in Fig. 10(b) and 0.2,0.25 and 0.3% in Fig. 10(c). The threshold *maxPer* is set as 2% Fig. 10(c) and 15% in both Fig. 10(a) and (b). Fig. 11(a),(b) and (c) presents the corresponding *RFPPs* and *R3Ps* generated for these execution setups. The experiments resulted in the following significant outcomes: (i) As it can be observed from Fig. 11(a) and (b) the *minRarePS* variation has shown a negative effect on the count of resultant *RFPPs* and *R3Ps.* The decrease in *minRarePS* threshold accelerates the count of rare 1-itemsets. In addition, this rise in the count of rare 1-itemsets increases the number of *RFPPs* and *R3Ps* produced. A similar observation is noted during the generation of *R3Ps* as depicted in Fig. 11(c) for *T20I6D100K* dataset. Additionally, there is a slight variation where a decrease in *minRarePS* has not much effect on



(a) Mushroom, maxPer=15% and minFreqPS=16,18,20%

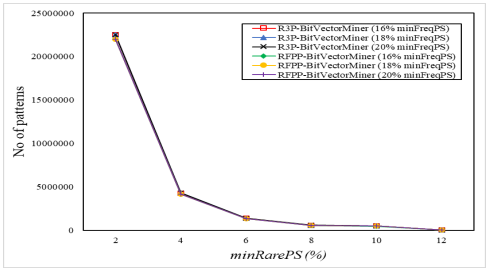


(b) Pollution, maxPer=15% and minFreqPS=49,50,55%

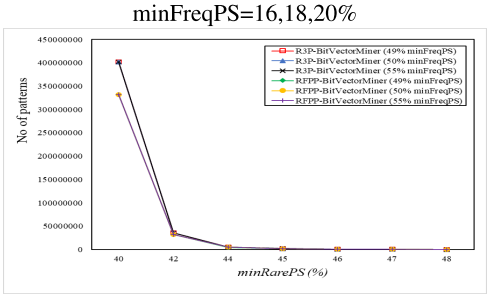


(c) T20I6D100K, maxPer=2% and minFreqPS=0.6,0.7,1%

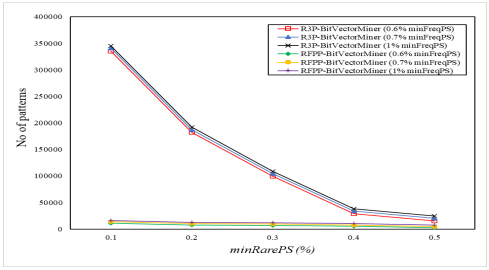
FIGURE 8. **Runtime comparison on different datasets for varying *minRarePS*** keeping *minFreqPS* and *maxPer* constant.



(a) Mushroom, maxPer=15% and minFreqPS=16,18,20%



(b) Pollution, maxPer=15% and minFreqPS=49,50,55%



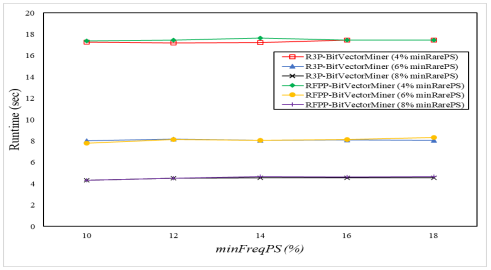
(c) T20I6D100K, maxPer=2% and minFreqPS=0.6,0.7,1%

FIGURE 9. **Number of *RFPPs*** *R3Ps* generated by different datasets for varying *minRarePS* keeping *minFreqPS* and *maxPer* constant.

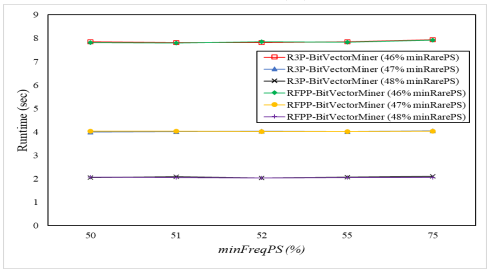
the extraction of *RFPPs* as shown in Fig. 11(c). (ii) An increase in *minRarePS* declines the number of *RFPPs* and *R3Ps* generated which results in a lesser time of execution. As the number of *RFPPs* and *R3Ps* rises the execution time also increases and vice versa. The run-time performance is shown in Fig. 10(a),(b) and (c). (iii) On the other side, the number of *RFPPs* and *R3Ps* increases very slowly with the rise in *minFreqPS* threshold value. Accordingly, even the total execution time also increases very slowly as *minFreqPS* threshold value is increased.

3) RUN-TIME COMPARISON ON DIFFERENT DATASETS FOR VARYING *maxPer* VALUE KEEPING *minFreqPS* AND *minRarePS* CONSTANT:

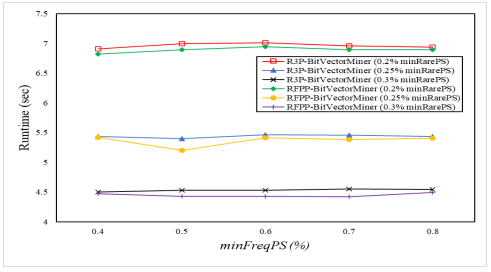
Total execution time taken by *Mushroom, Pollution* and *T20I6D100K* dataset respectively presented in Fig. 12(a),(b) and (c). The Figures present a significant impact on the total execution time when *maxPer* is changed by keeping *minRarePS* and *minFreqPS* constant. Threshold *maxPer* is varied from 0.1 to 20%, 2 to 50% and 0.25 to 20% in a different range as depicted in Fig. 12(a),(b) and (c) respectively. Whereas, *minRarePS* is kept constant as 4,6 and 8% in Fig. 12(a), 46,47 and 48% in Fig. 12(b) and 0.2,0.25 and 0.3% in Fig. 12(c). The threshold *minFreqPS* is kept as 10%, 50% and 1% in Fig. 12(a),(b) and (c) respectively. Corresponding *RFPPs* and *R3Ps* generated for these execution setups are shown in Fig. 13(a),(b) and (c) respectively. Different tests conducted have resulted in the following major outcomes: (i) For very low *maxPer* threshold value, either *RFPPs* not generated (Fig. 13(a),(b)) or very less *RFPPs* are generated (Fig. 13(c)). However, *R3Ps* are generated because itemsets are still considered even when some inter-arrival time exceeds *maxPer* threshold. An increase in *maxPer* threshold, increases the *PS* count which in turn changes more number of aperiodic 1-itemsets into partial periodic 1-itemsets. Consequently, the number of *RFPPs* and *R3Ps* produced also increases. It can be observed that when *maxPer* around 20, the same number of *RFPPs* and *R3Ps* is generated and further they will increase very slowly. This is because of the reduction in further transformation of non-periodic to periodic 1-itemsets. (ii) The *minRarePS* threshold value has a negative effect



(a) Mushroom, maxPer=15% and minRarePS=4,6,8%

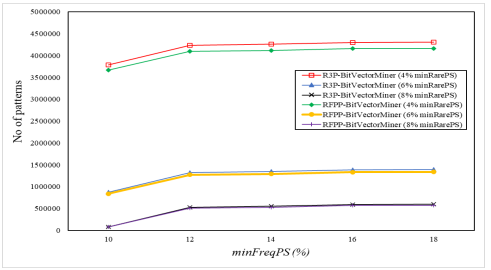


(b) Pollution, maxPer=15% and minRarePS=46,47,48%

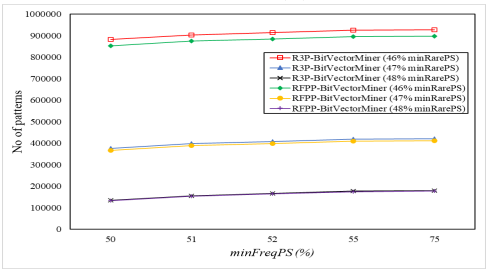


(c) T20I6D100K, maxPer=2% and minRarePS=0.2,0.25,0.3%

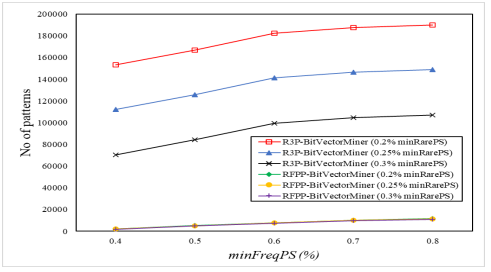
FIGURE 10. **Runtime comparison on different datasets for varying *minFreqPS*** keeping *minRarePS* and *maxPer* constant.



(a) Mushroom, maxPer=15% and minRarePS=4,6,8%



(b) Pollution, maxPer=15% and minRarePS=46,47,48%



(c) T20I6D100K, maxPer=2% and minRarePS=0.2,0.25,0.3%

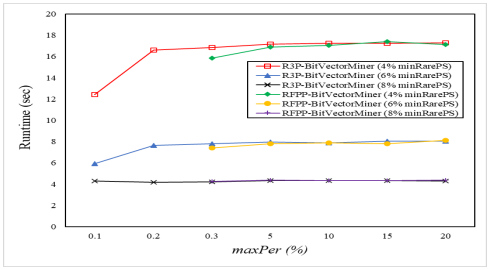
FIGURE 11. **Number of *RFPPs*** *R3Ps* generated by different datasets keeping *minRarePS* and *maxPer* constant.

on the number of *RFPPs* and *R3Ps* produced. An increase in *minRarePS* value, decreases the noisy itemsets while increasing the rare 1-itemsets. In addition, this reduction in noisy itemsets increases the number of *RFPPs* and *R3Ps* produced. (iii) Obviously, increase in the number of *RFPPs* and *R3Ps* demands more execution time as shown in Fig. 12(a),(b) and (c).

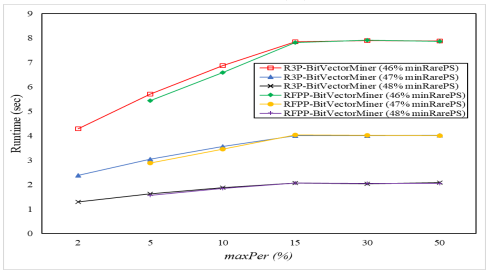
VII. DISCUSSION SECTION

This section compares the time complexity analysis of *3P-BitVectorMiner* and *3P-Growth* algorithms. *3P-Growth* [17], [18] is the state-of-the-art method that handles multiple arrivals of transactions simultaneously considering the row temporal database. The time complexity analysis of *3P-Growth:* In this model the two components of *3P-tree* named a *3P-list* and a prefix tree are constructed. The operations performed in *3P-Growth* algorithm are: i)Initially, the entire dataset is scanned and all the one-length periodic items are retained in the *3P-list.* Consider a temporal dataset comprising 'C' number of unique items and the database size depicted as 'S'. Here if 'C' number of items is considered interesting(periodic) and present in each transaction *S* then the generation of *3P-list* is in O(C × S). ii) The dataset is scanned once again and the items in every transaction are sorted and saved in the prefix tree. In the worst case, the prefix tree creation operation is in O(C × S). iii) Next, the prefix tree is recursively mined in a *dfs* manner to extract *PPPs.* The collection of possible itemsets generated is $N = 2^C - 1$. Finally, to generate the conditional pattern base, *3P-list* and prefix-tree of α for every considered itemset α that extends an itemset β, *3P-Growth* traverses the node-links of the *3P-list* of β. As these structures of β are only visited once, this construction is completed in linear time. Hence the *3P-Growth*'s overall time complexity is O(C × S × N). The count of itemsets taken into account in real-world applications relies on the features of the database and the considered threshold measures of the algorithm. Due to the usage of search space pruning techniques, fewer itemsets may be taken into account when *minPS* is increased and *maxPer* threshold value is decreased.

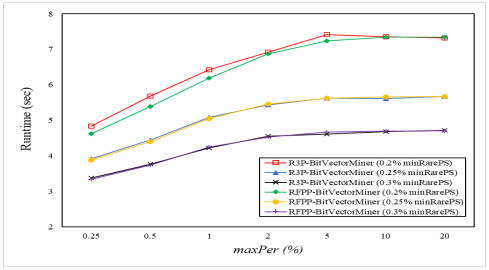
Resultant *PPPs* are generated using two algorithms in *3P-BitVectorMiner.* In Algorithm 1, the entire database is scanned and transformed into a bit-vector representation.



(a) Mushroom, minFreqPS=10% and minRarePS=4,6,8%

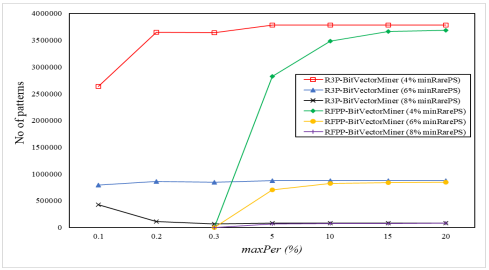


(b) Pollution, minFreqPS=50% and minRarePS=46,47,48%

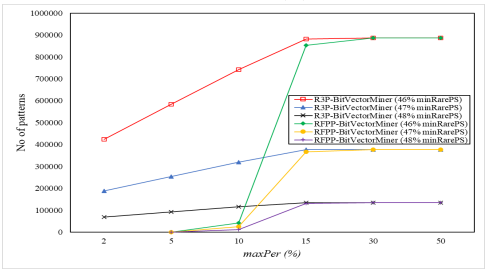


(c) T20I6D100K, minFreqPS=1% and minRarePS=0.2,0.25,0.3%

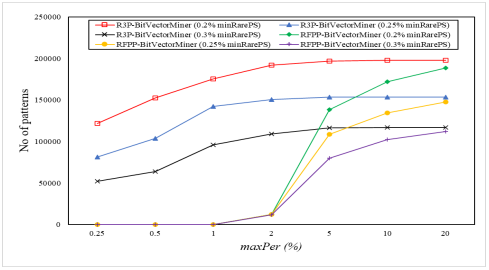
FIGURE 12. **Runtime comparison on different datasets for varying *maxPer*** keeping *minFreqPS* and *minRarePS* constant.



(a) Mushroom, minFreqPS=10% and minRarePS=4,6,8%



(b) Pollution, minFreqPS=50% and minRarePS=46,47,48%



(c) T20I6D100K, minFreqPS=1% and minRarePS=0.2,0.25,0.3%

FIGURE 13. **Number of *RFPPs*** *R3Ps* generated by different datasets keeping *minFreqPS* and *minRarePS* constant.

Simultaneously one length *PPPs* are generated and stored in *3PTSlist* data structure. In the worst case, the time complexity of *3PTSlist* creation is O(C × S). Initially, Algorithm 2, performs the logical AND operation by considering the two current length itemsets to generate the larger length itemsets. Every item is represented by *S* number of bits. Irrespective of the number of bits the time complexity for performing the logical AND operation results in O(1). Furthermore, the period support calculation considers every bit of each item that requires a maximum 'S' number of operations. On the itemset lattice, this algorithm applies the *DFS* method. There are $N = 2^C - 1$ itemsets that are possible. As a result, it takes O(N × S) time to create all potential interesting itemsets. Hence *3P-BitVectorMiner* has a total time complexity of O(N × S). Since *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* are variations of *3P-BitVectorMiner,* the time complexity remains same as that of *3P-BitVectorMiner.* The entire effectiveness of *3P-BitVectorMiner* ultimately depends on the real-world values of the provided parameters, such as *C, S,* and *N.* To prove that the proposed *3P-BitVectorMiner* method outperforms the state-of-the-art *3P-Growth* algorithm, extensive experiments on a variety of real-world databases are carried out in this paper.

VIII. CONCLUSION

The proposed efficient and novel algorithm named *3P-BitVectorMiner,* is a depth-first search method to capture entire partial periodic patterns from the temporal database. The thresholds *maxPer* and *minPS* successfully control the periodicity and the number of cyclic repetitions respectively. Here, the input temporal database is converted into a bit-vector form. *3PTSList* structure plays a vital role in producing the one-length partial periodic patterns by eliminating the non-periodic one-length patterns which reduce the huge search space. In contrast to the pattern growth method used in *3P-Growth* algorithm, here subsequent partial periodic patterns are generated with simple logical operations. An in-depth study of the novel approach *3P-BitVectorMiner* on various real-world, synthetic, sparse as well as dense datasets exhibited that it is run-time efficient, highly scalable and consumes less memory relative to the state-of-the-art *3P-Growth* algorithm. As in real-life it is necessary to

capture periodic rare patterns from the temporal database. In addition, two variations named *RFPP-BitVectorMiner* and *R3P-BitVectorMiner* is proposed to mine rare fully periodic pattern and rare partial periodic patterns from temporal databases respectively. Along with *maxPer,* two different support thresholds *minFreqPS* and *minRarePS* thresholds are used to control the number of cyclic repetitions and to prune the noisy itemsets. Different experiments carried out show that these proposed frameworks successfully capture periodic rare patterns in temporal databases.

The proposed *3P-BitVectorMiner* framework and its variants were limited to extracting partial and rare periodic patterns from static databases based on *maxPer* and *minPS* threshold values. However, alternative periodic support metrics can be applied as per the user requirements. The technique can also eventually be expanded to include incremental mining of rare periodic patterns, including partial periodic patterns. Further, the proposed algorithms may be enhanced to extract partial and rare periodic patterns from stream data. In addition, suitable real-world situations can be considered and the proposed frameworks may be employed to extract the uncommon periodic patterns. Further, a step may be used for an in-depth study of the patterns discovered by the proposed methods to uncover the hidden knowledge.

ACKNOWLEDGMENT

The authors would like to thank the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, for providing the laboratory facilities to conduct the experiments.

REFERENCES

[1] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Mining regular patterns in transactional databases," *IEICE Trans. Inf. Syst.*, vols. E91–D, no. 11, pp. 2568–2577, Nov. 2008.

[2] E. F. Glynn, J. Chen, and A. R. Mushegian, "Detecting periodic patterns in unevenly spaced gene expression time series using Lomb–Scargle periodograms," *Bioinformatics*, vol. 22, no. 3, pp. 310–316, Feb. 2006.

[3] M. A. Khaleel, G. N. Dash, K. S. Choudhury, and M. A. Khan, "Medical data mining for discovering periodically frequent diseases from transactional databases," in *Computational Intelligence in Data Mining*, vol. 1. Cham, Switzerland: Springer, 2015, pp. 87–96.

[4] F. Yi, L. Yin, H. Wen, H. Zhu, L. Sun, and G. Li, "Mining human periodic behaviors using mobility intention and relative entropy," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, 2018, pp. 488–499.

[5] A. C. M. Fong, B. Zhou, S. C. Hui, G. Y. Hong, and T. A. Do, "Web content recommender system based on consumer behavior modeling," *IEEE Trans. Consum. Electron.*, vol. 57, no. 2, pp. 962–969, May 2011.

[6] S. K. Tanbeer, C. F. Ahmed, and B.-S. Jeong, "Mining regular patterns in data streams," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2010, pp. 399–413.

[7] S. K. Tanbeer, M. M. Hassan, A. Almogren, M. Zuair, and B.-S. Jeong, "Scalable regular pattern mining in evolving body sensor data," *Future Gener. Comput. Syst.*, vol. 75, pp. 172–186, Oct. 2017.

[8] M. M. Rashid, M. R. Karim, B.-S. Jeong, and H.-J. Choi, "Efficient mining regularly frequent patterns in transactional databases," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2012, pp. 258–271.

[9] M. M. Rashid, I. Gondal, and J. Kamruzzaman, "Regularly frequent patterns mining from sensor data stream," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2013, pp. 417–424.

[10] P. Fournier-Viger, C.-W. Lin, Q.-H. Duong, T.-L. Dam, L. Ševčík, D. Uhrin, and M. Voznak, "PFPM: Discovering periodic frequent patterns with novel periodicity measures," in *Proc. 2nd Czech-China Sci. Conf.* London, U.K.: IntechOpen, 2017.

[11] P. Fournier-Viger, P. Yang, J. C.-W. Lin, and R. U. Kiran, "Discovering stable periodic-frequent patterns in transactional data," in *Proc. Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst.* Cham, Switzerland: Springer, 2019, pp. 230–244.

[12] P. Fournier-Viger, Y. Wang, P. Yang, and J. C. Lin, "TSPIN: Mining top-K stable periodic patterns," *Appl. Intell.*, vol. 52, pp. 6917–6938, Feb. 2021.

[13] R. U. Kiran, J. N. Venkatesh, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering partial periodic-frequent patterns in a transactional database," *J. Syst. Softw.*, vol. 125, pp. 170–182, Mar. 2017.

[14] J. N. Venkatesh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic-frequent patterns in transactional databases using all-confidence and periodic-all-confidence," in *Proc. 27th Int. Conf. (DEXA)*, in Lecture Notes in Computer Science, vol. 9827, S. Hartmann and H. Ma, Eds. Porto, Portugal: Springer, 2016, pp. 55–70.

[15] P. Ravikumar, V. V. Raj, P. Likhitha, R. U. Kiran, Y. Watanobe, S. Ito, K. Zettsu, and M. Toyoda, "Towards efficient discovery of partial periodic patterns in columnar temporal databases," in *Proc. Intell. Inf. Database Syst., 14th Asian Conf. (ACIIDS)*. Ho Chi Minh City, Vietnam: Springer, 2022, pp. 141–154.

[16] P. Ravikumar, P. Likhitha, B. Venus Vikranth Raj, R. Uday Kiran, Y. Watanobe, and K. Zettsu, "Efficient discovery of periodic-frequent patterns in columnar temporal databases," *Electronics*, vol. 10, no. 12, p. 1478, Jun. 2021.

[17] R. U. Kiran, P. Veena, P. Ravikumar, C. Saideep, K. Zettsu, H. Shang, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of partial periodic patterns in large temporal databases," *Electronics*, vol. 11, no. 10, p. 1523, May 2022.

[18] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering partial periodic itemsets in temporal databases," in *Proc. 29th Int. Conf. Sci. Stat. Database Manage.*, Jun. 2017, pp. 1–6.

[19] R. U. Kiran, V. Chhabra, S. Chennupati, P. K. Reddy, M.-S. Dao, and K. Zettsu, "A novel null-invariant temporal measure to discover partial periodic patterns in non-uniform temporal databases," in *Database Systems for Advanced Applications*. A. Bhattacharya, J. L. M. Li, D. Agrawal, P. K. Reddy, M. Mohania, A. Mondal, V. Goyal, and R. U. Kiran, Eds. Cham, Switzerland: Springer, 2022, pp. 569–577.

[20] P. Fournier-Viger, P. Yang, Z. Li, J. C.-W. Lin, and R. U. Kiran, "Discovering rare correlated periodic patterns in multiple sequences," *Data Knowl. Eng.*, vol. 126, Mar. 2020, Art. no. 101733.

[21] U. K. Jyothi, B. D. Rao, M. Geetha, and H. K. Vora, "Discovery of periodic rare correlated patterns from static database," in *Proc. 6th Int. Conf. Advance Comput. Intell. Eng.*, B. Pati, C. R. Panigrahi, P. Mohapatra, and K.-C. Li, Eds. Singapore: Springer Nature, 2023, pp. 649–660.

[22] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.

[23] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of periodic-frequent patterns in very large databases," *J. Syst. Softw.*, vol. 112, pp. 110–121, Feb. 2016.

[24] R. U. Kiran and M. Kitsuregawa, "Novel techniques to reduce search space in periodic-frequent pattern mining," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2014, pp. 377–391.

[25] 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*. Cham, Switzerland: Springer, 2018, pp. 146–172.

[26] K. Amphawan, P. Lenca, and A. Surarerks, "Efficient mining top-k regular-frequent itemset using compressed Tidsets," in *New Frontiers in Applied Data Mining*. Berlin, Germany: Springer, 2012, pp. 124–135.

[27] K. Amphawan, P. Lenca, A. Jitpattanakul, and A. Surarerks, "Mining high utility itemsets with regular occurrence," *J. ICT Res. Appl.*, vol. 10, no. 2, pp. 153–176, 2016.

[28] S. Laoviboon and K. Amphawan, "Mining high-utility itemsets with irregular occurrence," in *Proc. 9th Int. Conf. Knowl. Smart Technol. (KST)*, Feb. 2017, pp. 89–94.

[29] P. Fournier-Viger, J. C.-W. Lin, Q.-H. Duong, and T.-L. Dam, "PHM: Mining periodic high-utility itemsets," in *Proc. Ind. Conf. Data Mining*. Cham, Switzerland: Springer, 2016, pp. 64–79.

[30] R. U. Kiran, J. N. Venkatesh, P. Fournier-Viger, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic patterns in non-uniform temporal databases," in *Proc. Adv. Knowl. Discovery Data Mining 21st Pacific–Asia Conf.*, in Lecture Notes in Computer Science, vol. 10235, J. Kim, K. Shim, L. Cao, J. Lee, X. Lin, and Y. Moon, Eds. Jeju-do, South Korea, May 2017, pp. 604–617.

[31] J. N. Venkatesh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic-correlated patterns in temporal databases," *Trans. Large Scale Data Knowl. Centered Syst.*, vol. 38, pp. 146–172, Nov. 2018.

[32] P. Likitha, P. Veena, R. U. Kiran, Y. Watanobe, and K. Zettsu, "Discovering maximal partial periodic patterns in very large temporal databases," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2021, pp. 1460–1469.

[33] P. Likhitha, P. Ravikumar, R. U. Kiran, and Y. Watanobe, "Discovering top-k periodic-frequent patterns in very large temporal databases," in *Proc. Big Data Anal., 10th Int. Conf. (BDA)*. Hyderabad, India: Springer, Dec. 2023, pp. 200–210.

[34] M. Adda, L. Wu, S. White, and Y. Feng, "Pattern detection with rare item-set mining," 2012, *arXiv:1209.3089*.

[35] L. Troiano, G. Scibelli, and C. Birtolo, "A fast algorithm for mining rare itemsets," in *Proc. 9th Int. Conf. Intell. Syst. Design Appl.*, Nov. 2009, pp. 1149–1155.

[36] L. Troiano and G. Scibelli, "A time-efficient breadth-first level-wise lattice-traversal algorithm to discover rare itemsets," *Data Mining Knowl. Discovery*, vol. 28, no. 3, pp. 773–807, May 2014.

[37] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *Proc. 19th IEEE Int. Conf. Tools with Artif. Intelligence (ICTAI)*, Oct. 2007, pp. 305–312.

[38] L. Szathmary, P. Valtchev, and A. Napoli, "Generating rare association rules using the minimal rare itemsets family," *Int. J. Softw. Informat.*, vol. 4, pp. 219–238, Jan. 2010.

[39] S. Bouasker and S. Ben Yahia, "Key correlation mining by simultaneous monotone and anti-monotone constraints checking," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, Apr. 2015, pp. 851–856.

[40] S. Bouasker, T. Hamrouni, and S. B. Yahia, "New exact concise representation of rare correlated patterns: Application to intrusion detection," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, 2012, pp. 61–72.

[41] A. Borah and B. Nath, "Identifying risk factors for adverse diseases using dynamic rare association rule mining," *Exp. Syst. Appl.*, vol. 113, pp. 233–263, Dec. 2018.

[42] S. Rai, M. Geetha, P. Kumar, and B. Giridhar, "Binary count tree: An efficient and compact structure for mining rare and frequent itemsets," *Engineered Sci.*, vol. 17, pp. 185–194, Oct. 2022.

[43] Y. Lu, F. Richter, and T. Seidl, "Efficient infrequent pattern mining using negative itemset tree," in *Complex Pattern Mining*. Cham, Switzerland: Springer, 2020, pp. 1–16.

[44] S. Tsang, Y. S. Koh, and G. Dobbie, "RP-tree: Rare pattern tree mining," in *Proc. Int. Conf. Data Warehousing Knowl. Discovery*. Cham, Switzerland: Springer, 2011, pp. 277–288.

[45] A. Borah and B. Nath, "Mining rare patterns using hyper-linked data structure," in *Proc. Int. Conf. Pattern Recognit. Mach. Intell.*, Dec. 2017, pp. 467–472.

[46] L. Szathmary, P. Valtchev, A. Napoli, and R. Godin, "Efficient vertical mining of minimal rare itemsets," in *Proc. CLA*, 2012, pp. 269–280.

[47] A. Gupta, A. Mittal, and A. Bhattacharya, "Minimally infrequent itemset mining using pattern-growth paradigm and residual trees," 2012, *arXiv:1207.4958*.

K. JYOTHI UPADHYA is currently pursuing the Ph.D. degree with the Manipal Academy of Higher Education, Manipal, India. She is an Assistant Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education. Her research interests include rare pattern mining, periodic pattern mining in static, and stream data.

AMAN PALEJA is currently pursuing the bachelor's degree with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.

M. GEETHA (Member, IEEE) received the Ph.D. degree from NITK Surathkal, in 2010. She is currently a Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India. She has presented several papers in national and international conferences and her work has been published in several international journals. Her research interests include data mining, text mining in healthcare, and financial sectors.

B. DINESH RAO received the M.Tech. degree from the Indian Institute of Technology Kanpur, Kanpur, in 1993, and the Ph.D. degree from the Manipal Academy of Higher Education, Manipal, in 2014. He is currently a Professor with the Manipal School of Information Sciences, Manipal Academy of Higher Education. He has around 30 years of teaching experience. He has published many research papers in national/international journals/conferences. His research interests include fractals and theoretical computer science.

MINI SHAIL CHHABRA is currently pursuing the bachelor's degree with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.