IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 September 2023, accepted 9 October 2023, date of publication 20 October 2023, date of current version 31 October 2023.
Digital Object Identifier 10.1109/ACCESS.2023.3326419

RESEARCH ARTICLE

Mining Periodic-Frequent Patterns in Irregular Dense Temporal Databases Using Set Complements

PAMALLA VEENA[1], TARUN SREEPADA[2], (Member, IEEE), RAGE UDAY KIRAN[2], (Senior Member, IEEE), MINH-SON DAO[3], KOJI ZETTSU[3], YUTAKA WATANOBE[2], (Member, IEEE), AND JI ZHANG[4], (Senior Member, IEEE)
[1]Sri Balaji PG College, JNTU-Anantapur, Anantapur, Andhra Pradesh 515001, India
[2]Division of Information Systems, The University of Aizu, Aizuwakamatsu, Fukushima 965-0006, Japan
[3]National Institute of Information and Communications Technology, Koganei, Tokyo 184-8795, Japan
[4]School of Mathematics, Physics and Computing, the University of Southern Queensland, Toowoomba, QLD 4350, Australia

Corresponding author: Rage Uday Kiran (udayrage@u-aizu.ac.jp)

This work was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 21K12034.

ABSTRACT Periodic-frequent patterns are a vital class of regularities in a temporal database. Most previous studies followed the approach of finding these patterns by storing the temporal occurrence information of a pattern in a list. While this approach facilitates the existing algorithms to be practicable on sparse databases, it also makes them impracticable (or computationally expensive) on dense databases due to increased list sizes. A renowned concept in set theory is that the larger the set, the smaller its complement will be. Based on this conceptual fact, this paper explores the *complements,* redefines the periodic-frequent pattern and proposes an efficient depth-first search algorithm that finds all periodic-frequent patterns by storing only non-occurrence information of a pattern in a database. Experimental results on several databases demonstrate that our algorithm is efficient.

INDEX TERMS Data mining, pattern mining, periodic patterns, set complements, temporal databases.

I. INTRODUCTION

The big data generated by real-world applications naturally exists as a temporal database, an ordered set of transactions by timestamp. Beneficial patterns that can empower the users with competitive information to achieve socio-economic development lie hidden in this data. Tanbeer et al. [1] described a model to find periodically occurring frequent patterns in a uniform temporal database. Venkatesh et al. [2] generalized this model to (an irregular) temporal database. Since then, the problem of finding these patterns has received considerable attention [3], [4], [5], [6], [7], [8]. A classic use case of periodic-frequent patterns is market-basket analysis. It involves finding the regularly purchased itemsets in market-basket data. An example of a periodic-frequent pattern is:

$$\{Bread, Jam, Bag\}[support = 5\%, periodicity = 2\ hrs].$$

The above pattern says that five percent of the customers have purchased the items 'Bread,' 'Jam,' and 'Bag' at least once every two hours. The supermarket managers may find this information beneficial for campaigning, inventory management, and product placement.

The basic periodic-frequent pattern model is as follows [2]: Let $I = \bigcup_{x=1}^{m} i_x, m \geq 1$, be a set of items. Let $TS = \bigcup_{x=i}^{f} ts_x$, where $ts_x \in \mathbb{R}^+$ represents a timestamp, $ts_i = 1$ represents the initial timestamp in TS and ts_f represents the final timestamp in TS. The ts_i represents a hypothetical timestamp, which is crucial to determine the time taken for the first appearance of a pattern in a database. Let $P \subseteq I$ be a **pattern** (or an itemset). A pattern containing β, $\beta \geq 1$, number of items is called a **β-pattern**. A **transaction**, $tr =$

The associate editor coordinating the review of this manuscript and approving it for publication was Juan A. Lara.

(tid, ts_a, Q), $a \geq 1$, is a triplet, where $tid \in \mathbb{R}^+$ denotes the transaction-identifier, $ts_a \in \{TS - 0\}$ represents the timestamp, and Q is a pattern. A temporal database, denoted as *TDB*, over I and TS is an ordered set of transactions by timestamp. That is, $TDB = \bigcup_{x=1}^{n} tr_x$, $n = |TDB|$, where $|TDB|$ represents the number of transactions in *TDB*. A temporal database is said to be **regular** if $n = t_f$; otherwise, the database is said to be **irregular**. For a transaction $tr = (tid, ts_x, Q)$, if $P \subseteq Q$, it is said that P occurs in tr (or tr contains P) and that timestamp is denoted as ts_x^P. Let $TS^P = \bigcup_{x=j}^{k} ts_x^P$, $j, k \in [1, ts_f]$, be an **ordered set of timestamps** at which P has occurred in *TDB*. The *support* of P, denoted as $sup(P) = |TS^P|$, where $|TS^P|$ represents the number of transactions containing P. The pattern P is said to be a **frequent pattern** if $sup(P) \geq minSup$, where *minSup* refers to the user-specified *minimum support* value. Let ts_c^P and $ts_d^P, j \leq c < d \leq k$, be the two consecutive timestamps in TS^P. The time difference (or an inter-arrival time) between ts_d^P and ts_c^P is defined as a ***period*** of P, say per_e^P. That is, $per_e^P = ts_d^P - ts_c^P$. Let $PER^P = \{per_1^P, per_2^P, \cdots, per_d^P\}$ be the set of all *periods* for pattern P. The ***periodicity*** of P, denoted as $prd(P) = max(per_x^P | \forall per_x^P \in SP^P)$. The frequent pattern P is said to be a **periodic-frequent pattern** if $prd(P) \leq maxPRD$, where *maxPRD* refers to the user-specified *maximum periodicity* value. Given a temporal database (*TDB*) and the user-specified *minimum support* (*minSup*) and *maximum periodicity* (*maxPRD*) constraints, the **problem definition** of periodic-frequent pattern mining is finding the complete set of periodic-frequent patterns having *support* no less than *minSup* and *periodicity* no more than the *maxPRD*. Please note that a pattern's *support* and *periodicity* can be represented in the percentage of $|TDB|$ and t_f, respectively.

Example 1: Let $I = \{p, q, r, s, t, u, v\}$ be the set of items. Let $TS = \{0, 1, 2, \cdots, 12\}$ be the set of timestamps. A hypothetical temporal database generated from I is shown in Table 1. This database contains 10 transactions, i.e., $n = 10$. The initial timestamp of this database, i.e., $ts_i = 0$. The final timestamp of this database, i.e., $t_f = 12$. Since $n! = ts_f$, this database represents an irregular temporal database with no transaction occurring at the timestamps 2 and 10. The set of items p and q, i.e., $\{p, q\}$ (or pq, in short) is a pattern. It is a 2-pattern as it contains only two items. The pattern pq appears in the transactions whose timestamps are 1, 4, 5, 8, 9, and 12. Therefore, the list of timestamps containing pq, i.e., $TS^{pq} = \{1, 4, 5, 8, 9, 12\}$. The *support* of pq, i.e., $sup(pq) = |TS^{pq}| = 6$. If the user-specified $minSup = 5$, then pq is a frequent pattern as $sup(pq) \geq minSup$. The periods for this pattern are: $per_1^{pq} = 1\ (= 1 - ts_i)$, $per_2^{pq} = 3\ (= 4 - 1)$, $per_3^{pq} = 1\ (= 5 - 4)$, $per_4^{pq} = 3\ (= 8 - 5)$, $per_5^{pq} = 1\ (= 9 - 8)$, $per_6^{pq} = 3\ (= 12 - 9)$, and $per_7^{pq} = 0\ (= ts_f - 12)$. Thus, $SP^{pq} = \{1, 3, 1, 3, 1, 3, 0\}$. The *periodicity* of pq, i.e., $prd(pq) = maximum(1, 3, 1, 3, 1, 3, 0) = 3$. If the user-defined $maxPRD = 3$, then the frequent pattern pq is said to be a periodic-frequent pattern because $per(pq) \leq maxPRD$.

TABLE 1. An irregular temporal database.

tid	ts	items
1	1	*pqtu*
2	3	*rst*
3	4	*pqt*
4	5	*pqrstu*
5	6	*pv*
6	7	*rsu*
7	8	*pqt*
8	9	*pqrstv*
9	11	*qrs*
10	12	*pqrsu*

TABLE 2. All periodic-frequent patterns generated from Table 1 at *minSup* = 5 and *maxPRD* = 3.

Patterns	*sup*	*prd*
p	7	3
q	7	3
t	6	3
r	6	2
s	6	2
pt	5	3
qt	5	3
rs	6	2
pq	6	3
pqt	5	3

The complete set of periodic-frequent patterns generated from Table 1 is shown in Table 2.

Several algorithms [3], [9], [10], [11], [12], [13], [14] were described in the literature to find the periodic-frequent patterns in a database. The basic approach used in these algorithms has always been the same. It involved the following two steps:

1) Construct a list, say **ts-list**, containing the occurrence timestamps of a pattern. (The ts-list captures TS^P.)
2) Determine whether a pattern is periodic-frequent by performing an exhaustive search on its ts-list.

The time complexity to search a pattern's ts-list is $O(n)$, where n represents the length of a ts-list. Henceforth, the performance of a periodic-frequent pattern mining algorithm primarily depends on the lengths of ts-lists generated for the patterns in a database. The ts-lists in sparse databases are relatively small (or manageable) than those in dense databases. Consequently, the existing algorithms are practicable on sparse databases while impracticable (or computationally expensive) on dense databases. This paper tackles this challenging problem by exploring the concept of "*(set) complements*" and proposing an efficient algorithm to find the patterns in dense databases.

It has to be noted that discovering periodic-frequent patterns in dense databases is a challenging and non-trivial task for the following reasons:

1) Many algorithms [15], [16], [17], [18] were described in the literature to find frequent patterns in a dense transactional database. Since these algorithms completely disregard the temporal occurrence information of an item, they cannot be extended to discover periodic-frequent patterns in a dense temporal database.
2) Zaki and Gouda. [19] explored the concept of *set difference*[1] to calculate the *frequency* (or *support*) of a pattern in a database. They have not described any methodology to determine the *periodicity* of a pattern.

[1]The term *set difference* refers to relative complement, whereas the term *complement* typically refers to absolute complement.

This paper proposes a novel method to calculate a pattern's *periods* and *periodicity* using *complements*.

The contributions of this paper are as follows. First, we introduce the concept of *complement timestamp-list* for an item and a pattern. Second, we redefine the *support*, *period*, *periodicity*, and *periodic-frequent pattern* using the *complement timestamp-lists*. Third, we propose a novel depth-first search algorithm to find all periodic-frequent patterns in a dense temporal database. We call our algorithm Periodic-Frequent Pattern Miner with Complements (PFPM-C). We also present our algorithm's correctness and theoretical complexity. Fourth, a new synthetic database generator algorithm was proposed to create synthetic sparse and dense temporal databases. We conduct experiments on synthetic and real-world databases and show that our algorithm is efficient concerning memory and runtime and highly scalable. Fifth, a case study on air pollution analytics has been presented to demonstrate the usefulness of finding periodic-frequent patterns using PFPM-C. This analytics involves identifying the areas where people were regularly exposed to harmful pollution levels in Japan.

This study is a significantly expanded version of our previous work [20], which reported a preliminary version of PFPM-C. This study contributes significantly to the related work by completely comprehending the existing literature. This study also provides theoretical correctness of the extended model of periodic-frequent patterns based on set complements. The time complexity of PFPM-C has also been investigated in this paper. Furthermore, incorporating new databases significantly expands the experimental findings section (Section V), which is of the utmost importance. This study demonstrates that PFPM-C outperforms the state-of-the-art on dense databases, regardless of the *minSup* and *maxPRD* values.

The remainder of this study is organized as follows. Section II describes the literature on frequent pattern mining and periodic-frequent periodic mining. In Section III, we define the model of periodic-frequent patterns using complements. In Section IV, we introduce our algorithm. In Section V, we present our experimental results. Finally, in Section VI, we present our conclusions and future works.

II. RELATED WORK

A. FREQUENT PATTERN MINING

Agrawal et al. [21] introduced frequent pattern mining as a key intermediary step to discover interesting associations between the itemsets in a transactional database. Since then, several algorithms (e.g., Apriori [21], ECLAT [22], and Frequent Pattern-growth [23]) were described in the literature to find these patterns effectively. Most of these algorithms can discover frequent patterns effectively in a sparse transactional database; however, they suffer from performance issues while mining the patterns in a dense transactional database. Zaki and Gouda [19] first explored the concept of set difference and proposed a depth-first search algorithm, ECLAT-diffSets, to find all frequent patterns in a dense transactional database. This algorithm uses the **size of set difference list** to determine whether a pattern is frequent or infrequent in the database. This algorithm does not describe any methodology to determine the *periodicity* of a pattern from the set difference list. Henceforth, this algorithm cannot be directly extended to find periodic-frequent patterns. This paper proposed a novel methodology to calculate the *periodicity* of a pattern given its set difference information. Luna et al. [18] conducted a detailed survey on frequent pattern mining and presented the improvements in the past 25 years.

B. PERIODIC PATTERN MINING IN TIME SERIES, EVENT SEQUENCES, AND GRAPHS

A key limitation of frequent pattern mining studies is their inability to consider the temporal occurrence information of the items in a database. Ozden et al. [24] tried to solve this limitation by adding a time attribute to a transactional database, then splitting the database into non-overlapping subsets by time, and finding cyclic association rules by counting the number of subsets in which a pattern has occurred. This approach simplifies the mining algorithm but also raises a major limitation of missing the patterns that span multiple windows.

Inspired by Ozden's work [24], Han et al. [25] described a model to find partial periodic patterns in an evenly spaced binary time series. Later, the authors proposed an efficient algorithm [26] to discover the partial periodic patterns. In this model, a binary series is split into multiple sequences of a particular length specified by the user, and interesting patterns were discovered using only the *minSup* threshold value. Yang et al. [27] extended Han's model to multiple minimum supports to address the rare item problem. Yang et al. [28] extended the model to discover asynchronous periodic patterns in a time series. Xun et al. [29] proposed an ECLAT-variant to discover partial periodic patterns in multi-source time series data. A key limitation of these models is that they fail to discover patterns spanning multiple sequences. More importantly, the periodic patterns discovered from a time series are conceptually different from those in a temporal database. In particular, the periodic patterns generated in a time series are very close to frequent patterns generated in a transactional database (if the length of segment or *period* is set to 1) as these models only use the *minSup* constraint to determine the interestingness of a pattern.

An event sequence represents an ordered list of events, where each event has a distinct timestamp. Mannila et al. [30] introduced frequent episode mining to find all the episodes (or subsequences of events) that frequently appear in a sequence over time. An episode is said to be frequent if its *support* is no less than the user-specified *minSup* value. Huang and Chang [31] extended Mannila's model to discover frequent episodes in complex event sequences. Although temporal and event-sequence databases capture the temporal occurrence information of the items in a database, they differ in their underlying data models and how they handle temporal aspects. As a result, the knowledge discovered from frequent episodes is completely different from the partial periodic patterns discovered from temporal databases.

Lahiri and Berger-Wolf [32] proposed a model to discover periodic patterns in graphs. Zhang et al. [33] described a model to discover seasonal periodic subgraphs in a network. It has to be noted that the patterns discovered from graphs are completely different from the patterns generated in databases.

C. PERIODIC-FREQUENT PATTERN MINING

Tanbeer et al. [1] described the periodic-frequent pattern model that eliminated the need for splitting the database by time. This model involved discovering all patterns in a temporal database that satisfy the user-specified *minSup* and *maxPrd* constraints. A pattern growth technique was presented to generate all periodic-frequent patterns in a database. Amphawan et al. [11] designed an efficient depth-first search-based algorithm for mining top-K periodic-frequent patterns without using the user-specified *minSup* constraint. Kiran and Reddy [3] introduced a novel greedy approach to discover periodic-frequent patterns effectively. Anirudh et al. [9] introduced a novel concept of periodic summaries to find the periodic-frequent patterns in a temporal database. Anirudh et al. [10] also presented a distributed in-memory algorithm based on map-reduce and Spark environment. Ravikumar et al. [14] proposed an ECLAT-based [22] to find periodic-frequent patterns in columnar databases. Tarun et al. [5] described a CUDA-based GPU algorithm to find periodic-frequent patterns. Since these algorithms store a pattern's complete temporal occurrence information in a *list*, they suffer from computational issues while dealing with dense databases.

D. EXTENSIONS OF PERIODIC-FREQUENT PATTERN MINING

Recently, the periodic-frequent pattern model was extended to find fuzzy periodic-frequent patterns [34], stable periodic-frequent patterns [7], [35], non-redundant periodic-frequent patterns [13], periodic-frequent patterns in uncertain data [36], geo-referenced periodic-frequent patterns [37], periodic-correlated patterns [2], regular patterns [38], [39], [40], periodic high-utility patterns [41], fuzzy driven periodic mining [6], and maximal periodic-frequent patterns [42]. Unfortunately, most of these algorithms maintain the temporal occurrence information in a list structure and, thus, suffer from computational issues while dealing with dense databases. The solution presented in this paper can be extended to improve the performance of the above algorithms. However, in this paper, we confine ourselves to improving the performance of the basic periodic-frequent pattern mining algorithms for brevity.

III. REDEFINITION OF A PERIODIC-FREQUENT PATTERN USING COMPLEMENTS

In set theory, given a universe of elements U, a complement of a set A, denoted as A^c, is the set of elements not in A. That is, $A^c = U - A$. More important, it turns out that the larger the set A, the smaller its complement A^c will be, and vice-versa. This motivated us to discover periodic-frequent patterns in dense databases using *set complements*. However, a key challenge we encountered is the methodology to determine the *periodicity* of a pattern from a complement set, as no prior studies exist in the literature. In this section, we resolve this challenge and provide the correctness.

Definition 1: The cts-list of an item $i_j \in I$, denoted as $\widehat{TS^{i_j}} = \{TS - TS^{i_j}\}$.

Example 2: In Table 1, the item q occurred at the timestamps of 1, 4, 5, 8, 9, 11, and 12. Thus, $TS^q = \{1, 4, 5, 4, 8, 9, 11, 12\}$. The cts-list of q, denoted as $\widehat{TS^q} = TS - TS^q = \{1, 3, 4, 5, 6, 7, 8, 9, 11, 12\} - \{1, 4, 5, 8, 9, 11, 12\} = \{3, 6, 7\}$. Similarly, the cts-list of p, i.e., $\widehat{p} = \{3, 7, 11\}$.

Definition 2: The cts-list of a pattern P, denoted as $\widehat{TS^P}$, represents the union of complement timestamp-lists of its items. That is, $\widehat{TS^P} = \bigcup_{i_k \in P} \widehat{TS^{i_k}}$.

Example 3: Continuing with the previous example, the complement timestamp-list of the pattern qp, i.e., $\widehat{qp} = \widehat{TS^q} \cup \widehat{TS^p} = \{3, 6, 7\} \cup \{3, 7, 11\} = \{3, 6, 7, 11\}$.

The correctness of the above definition is based on Property 1 and shown in Theorem 1.

Property 1: The timestamp-list of a pattern P, i.e., $TS^P = \bigcap_{i_k \in P} TS^{i_k}$.

Theorem 1: Let $P = \{i_1, i_2, \cdots, i_x\}$, $1 \leq x \leq m$, be a pattern in TDB. The complement timestamp-list of P, i.e., $\widehat{TS^P} = \bigcup_{i_k \in P} \widehat{TS^{i_k}}$.

Proof: According to set theory, the complement of TS^P, i.e.,

$$\begin{aligned}
\widehat{TS^p} &= TS - TS^P \\
&= TS - \bigcap_{i_k \in P} TS^{i_k} \\
&= TS - (TS^{i_1} \cap TS^{i_2} \cap \cdots \cap TS^{i_x}) \\
&= (TS - TS^{i_1}) \cup (TS - TS^{i_2}) \cup \cdots \cup (TS - TS^{i_x}) \\
&= \widehat{TS^{i_1}} \cup \widehat{TS^{i_2}} \cup \cdots \cup \widehat{TS^{i_x}} \\
&= \cup_{i_k \in P} \widehat{TS^{i_k}} \qquad (1)
\end{aligned}$$

Hence proved. ■

Definition 3: The *support* of pattern P, denoted as $sup(P) = |TS| - |\widehat{TS^X}|$.

Example 4: The *support* of qp, i.e., $sup(qp) = 10 - 4 = 6$. The correctness of the above definition is shown in Theorem 2.

Property 2: If X, Y and Z be three patterns such that $Z = X \cup Y$ and $X \cap Y = \emptyset$, then $\widehat{TS^Z} = \widehat{TS^X} \cup \widehat{TS^Y}$.

Example 5: The $\widehat{TS^{qp}} = \widehat{TS^q} \cup \widehat{TS^p} = \{3, 7, 11\} \cup \{3, 6, 7\} = \{3, 6, 7, 11\}$. It means the pattern qp does not appear at 3, 6, 7, and 11 timestamps.

Definition 4 (The Periodicity of Pattern X): Let $\widehat{MTS_k^Z} \subseteq \widehat{TS^Z}$, where $k \geq 1$, be a maximal set of consequence timestamps in $\widehat{TS^Z}$ such that $\widehat{TS^Z} = \{\widehat{MTS_1^Z} \cup \widehat{MTS_2^Z} \cup \cdots \cup \widehat{MTS_k^Z}\} = \cup_{j=1}^{k}\widehat{MTS_j^Z}$ and $\widehat{MTS_i^Z} \cap \widehat{MTS_j^Z} = \emptyset, 1 \leq i \leq j \leq k$. The *periodicity* of Z given $\widehat{TS^Z}$, i.e., $prd(Z) = max(|\widehat{MTS_i^Z}|\forall\widehat{MTS_i^Z} \in \widehat{TS^Z}) + 1$.

Example 6: We can split the $\widehat{TS^{qp}}$ into three maximal consecutive subsets, $\widehat{MTS_1^{qp}} = \{3\}$, $\widehat{MTS_2^{qp}} = \{6, 7\}$, and $\widehat{MTS_3^{qp}} = \{11\}$. Thus, periodicity of qp, denoted as $prd(qp) = max(|\{3\}|, |\{6, 7\}|, |\{11\}|) + 1 = 3$.

Theorem 2: Let $P = \{i_1, i_2, \cdots, i_x\}$, $1 \leq x \leq m$, be a pattern in *TDB*. The *support* of pattern P, denoted as $sup(P) = |TS| - |\widehat{TS^P}|$.

Proof: The *support* of P according to the definition of *support* 3, the *support* of P, i.e.,

$$\begin{aligned} sup(P) &= |TS^P| \\ &= |TS - \widehat{TS^P}|. \end{aligned} \tag{2}$$

Hence proved. ∎

Theorem 3: Let $P = \{i_1, i_2, \cdots, i_x\}$, $1 \leq x \leq m$, be a pattern in *TDB*. The *periodicity* of P in *TDB*, i.e., $prd(P) = max(|\widehat{MTS_i^P}|\forall\widehat{MTS_i^P} \in \widehat{TS^Z}) + 1$.

Proof: According to the definition of the periodic-frequent pattern model, the *periodicity* of P, i.e.,

$$\begin{aligned} pd(P) &= max(\bigcup_{j=1}^{d} per_j^P|\forall per_j^P \in SP^P) \\ &= max(\bigcup_{j=1}^{d}|\widehat{MTS_j^P}) + 1| \\ &= |max(\bigcup_{j=1}^{d}\widehat{MTS_j^P})| + 1. \end{aligned} \tag{3}$$

Hence proved. ∎

Please note the definition of a periodic-frequent pattern remains unchanged in our approach. We consider a pattern periodic-frequent if it satisfies the user-specified *minSup* and *maxPRD* constraints. The following section presents our algorithm to find periodic-frequent patterns in a dense database.

IV. PROPOSED ALGORITHM

A. BASIC IDEA: CONSTRUCTION OF CTS-LIST DURING DEPTH-FIRST SEARCH

An itemset lattice represents the space of items in a database. This lattice represents the search space of periodic-frequent pattern mining. Henceforth, the search space size of periodic-frequent pattern mining is $2^n - 1$, where n represents the total number of items in a database. Reducing this huge search space is a challenging task in pattern mining. We try to reduce this huge search space by performing the depth-first search using the **downward closure property** of periodic-frequent patterns (see Properties 3 and 4).

A key challenge in performing the depth-first search on the itemset lattice is the construction of the correct cts-list for a child node. We tackle this challenge effectively by constructing the cts-list of a child node, say Q, by performing the union operation between the cts-lists of its parent node, say P and an item, say i_k, that exists in the child node but not exist in the parent node. That is, if $Q = \{P \cup i_k\}$, where $i_k \notin P$ and $i_k \in I$, then $\widehat{Q} = \widehat{P} \cup \widehat{i_k}$. The correctness of our idea is shown in Lemma 1.

Property 3: If $P \subset Q$, then $sup(P) \geq sup(Q)$ and $prd(P) \leq prd(Q)$ as $TS^P \supseteq TS^Q$ (or $\widehat{TS^P} \subseteq \widehat{TS^Q}$).

Property 4 (The Downward Closure Property): If P is not a periodic-frequent pattern, then $\forall Q \supset P$, Q is not a periodic-frequent pattern. The reason is, if $sup(P) \ngeq minSup$ and/or $prd(P) \nleq maxPrd$, then $sup(Q) \ngeq minSup$ and/or $prd(Q) \nleq maxPrd$ (see Property 3).

Lemma 1: Let $P = \{i_1, i_2, \cdots, i_{k-1}\}$, $k > 1$, be a pattern representing a parent node. If $Q = \{i_1, i_2, \cdots, i_{k-1}, i_k\}$ is a pattern representing the child node of P, then $\widehat{Q} = \widehat{P} \cup \widehat{i_k}$.

Proof: According to Definition 1, the cts-list of P, i.e.,

$$\widehat{P} = \widehat{i_1} \cup \widehat{i_2} \cup \cdots \cup \widehat{i_{k-1}}. \tag{4}$$

Similarly, the cts-list of Q, i.e.,

$$\widehat{Q} = \widehat{i_1} \cup \widehat{i_2} \cup \cdots \cup \widehat{i_{k-1}} \cup \widehat{i_k}. \tag{5}$$

Substituting Equation 4 in Equation 5, we get

$$\widehat{Q} = \widehat{P} \cup \widehat{i_k}. \tag{6}$$

Hence proved. ∎

B. CONSTRUCTION OF CTS-LIST

Our algorithm, PFPM-C, has the following three steps: (*i*) Find all periodic-frequent items (or 1-patterns) by scanning the database, (*ii*) construct the complement set, i.e., $\widehat{TS^{i_j}}$, for every periodic-frequent item i_j, and (*iii*) using the *downward closure property* (see Property 4), find all periodic-frequent patterns from the database by performing the depth-first search on the lattice. Algorithms 1 and 2 provide the procedures to find the complete set of periodic-frequent patterns in a database. Now, we illustrate the working of these algorithms using the database shown in Table 1. Let *minSup* = 5 and *maxPRD* = 3.

Construct the list of timestamps for every item by scanning the database (Lines 1 to 8 in Algorithm 1). Fig. 2(a)-(c) shows the TS-list generated after scanning the first, second, and every transaction in the database. Next, calculate the *support* and *periodicity* for each item in the list. Using the *downward closure property*, we prune the items having *support* is less than *minSup* or *periodicity* is more than *maxPRD* (Lines 12 to 18 in Algorithm 1). Consider the remaining items in the list as periodic-frequent items and sort them in *support* descending order (Line 22 in Algorithm 1). Fig. 2(d) shows the sorted list of all periodic-frequent items discovered from Table 1. Let the sorted list of all discovered periodic-frequent patterns be denoted as L. Let us update the list of timestamps of all

Algorithm 1 PeriodicFrequentItems(*TDB*: Temporal Database, *minSup*: Minimum Support, *maxPRD*: Maximum Periodicity

```
1: Let $TS\text{-}list = (i_j, ts\text{-}list(i_j))$ be a dictionary that records the temporal occurrence information of an item $i_j$ in a $TDB$. Let $TS_l$ be a temporary list to record the timestamp of the last occurrence of an item in the database. Let $Per$ be a temporary list to record an item's periodicity in the database.
2: for each transaction $t_{cur} \in TDB$ do
3:     Set $ts_{cur} = t_{cur}.ts$;
4:     for each item $i_j \in t_{cur}.X$ do
5:         if $i_j$ does not appear in TS-list then
6:             Insert $_j$ and its timestamp into the TS-list. Set $TS_l[i_j] = ts_{cur}$ and $Per[i_j] = (ts_{cur} - ts_{initial})$;
7:         else
8:             Add $i_j$'s timestamp in the TS-list. Update $TS_l[i_j] = ts_{cur}$ and $Per[i_j] = max(Per[i_j], (ts_{cur} - TS_l[i_j]))$;
9:         end if
10:    end for
11: end for
12: for every item $i_j$ in TS-list do
13:     if $len(ts\text{-}list(i_j)) < minSup$ then
14:         Remove $i_j$ from the TS-list;
15:     else
16:         Set $Per[i_j] = max(Per[i_j], (ts_{final} - TS_l[i_j]))$;
17:         if $Per[i_j] > maxPRD$ then
18:             Remove $i_j$ from the TS-list.
19:         end if
20:     end if
21: end for
22: Treat the remaining items in the TS-list as periodic-frequent 1-patterns (or items). Next, sort these periodic-frequent items in support descending order. Let $L$ denote this sorted list of items.
23: for every item $i_j$ in TS-list do
24:     Update the $i_j$'s TS-list with its complement information. That is, set $ts\text{-}list(i) = TS - ts\text{-}list$. Let us call this new TS-list a CTS-list for brevity.
25: end for
26: Call PFPM-C(CTS-List).
```

periodic-frequent items with their *complements* (Lines 23 and 24 in Algorithm 1). Fig. 2(e) shows the resultant TS-list produced after applying the set complement. We now call this list a CTS-list for brevity. Next, we find all periodic-frequent patterns by performing a depth-first search on the itemset lattice generated by L (Line 26 in Algorithm 1). The search space optimization during the depth-first search is achieved by preventing the search on child nodes if a parent node fails to be a periodic-frequent pattern (Algorithm 2). Fig. 3. shows the depth-first search performed on the lattice. We start with the first item in L, i.e., p. Since p is a periodic-frequent

Algorithm 2 PFPM-C(CTS-List)

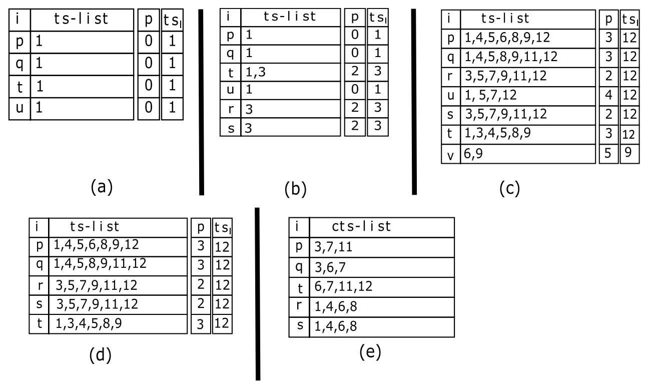
```
1: for each item $i$ in PFP-List do
2:     Set $pi = \emptyset$ and $X = i$;
3:     for each item $j$ that comes after $i$ in the CTS-list do
4:         Set $Y = X \cup j$. Determine $ts\text{-}list(Y) = ts\text{-}list(X) \cup ts\text{-}list(j)$; Calculate $sup(Y) = |TS| - |ts\text{-}list(Y)|$. Set $per(Y) = 0$, $periods = \emptyset$ and $tempPer = 0$;
5:         if ts-list(Y).size == 0 then
6:             $per(Y) = 1$
7:         else
8:             if $ts_{final}$ is in ts-list then
9:                 Remove $ts_{final}$ from ts-list
10:            end if
11:            for $(i = 0; i < ts\text{-}list(Y).size() - 1; ++i)$ do
12:                if $(ts\text{-}list(Y)[i+1] - ts\text{-}list(Y)[i]) == 1$ then
13:                    $tempPer += 1$;
14:                else
15:                    Append $tempPer$ in to periods. Set $tempPer = 0$;
16:                end if
17:            end for
18:        end if
19:        $per(Y) = max(periods) + 1$
20:        if $sup(Y) \geq minSup$ and $per(Y) \leq maxPRD$ then
21:            Add $Y$ to $pi$ and $Y$ is considered as periodic-frequent pattern;
22:        end if
23:    end for
24:    Call $PFPM\text{-}C(pi)$;
25: end for
```

pattern (or item), we concatenate p with the second item in L, i.e., q. The result is a new pattern pq. Using Property 2, we construct its complement list and determine whether it is a periodic-frequent pattern or not using Definition 4. We repeat this process until the child node fails to be a periodic-frequent pattern or lattice is traversed.

C. TIME COMPLEXITY ANALYSIS

Suppose we are examining a database that stores temporal information. This database contains a transactions, each corresponding to a specific time. Across all of these transactions, c unique items exist. Furthermore, the average transaction length is equal to b. In this database, all items are deemed of interest and, therefore, included in the analysis. Understanding the characteristics of the database, including the number of transactions, unique items count, and the length of transactions, is crucial for performing the complexity analysis.

The PFPM-C algorithm significantly contributes to PFPM by efficiently computing and identifying PFPs. The Algorithm 1 starts by scanning the entire database and calculating the *support*, and *periodicity* of each item. A list of items satisfying the *minSup* and *maxPRD* constraints are



(a)

i	ts-list	p	ts_l
p	1	0	1
q	1	0	1
t	1	0	1
u	1	0	1

(b)

i	ts-list	p	ts_l
p	1	0	1
q	1	0	1
t	1,3	2	3
u	1	0	1
r	3	2	3
s	3	2	3

(c)

i	ts-list	p	ts_l
p	1,4,5,6,8,9,12	3	12
q	1,4,5,8,9,11,12	3	12
r	3,5,7,9,11,12	2	12
u	1, 5,7,12	4	12
s	3,5,7,9,11,12	2	12
t	1,3,4,5,8,9	3	12
v	6,9	5	9

(d)

i	ts-list	p	ts_l
p	1,4,5,6,8,9,12	3	12
q	1,4,5,8,9,11,12	3	12
r	3,5,7,9,11,12	2	12
s	3,5,7,9,11,12	2	12
t	1,3,4,5,8,9	3	12

(e)

i	cts-list
p	3,7,11
q	3,6,7
t	6,7,11,12
r	1,4,6,8
s	1,4,6,8

FIGURE 1. The process of finding periodic-frequent items and constructing their cts-lists. (a) After scanning the first transaction. (b) After scanning the second transaction. (c) After scanning all of the transactions. (d) An unordered list of periodic-frequent items. (e) Sorted list of periodic-frequent items and their cts-lists.

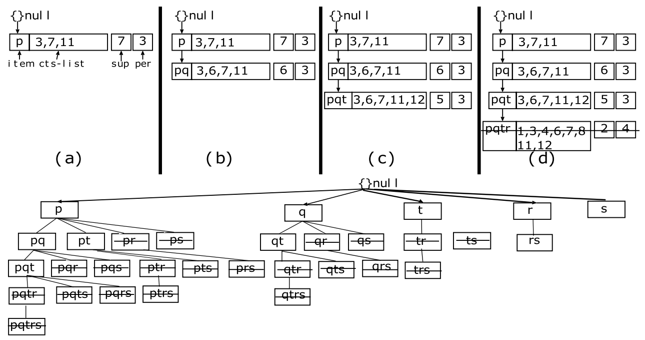


FIGURE 2. Finding periodic-frequent patterns in the itemset lattice using the depth-first search technique. (a) Let us start from the periodic-frequent item *p*. (b) After determining *p* as a periodic-frequent item, we visit its child node *pq*, construct its cts-list, and determine its *support* and *periodicity*. (c) As *pq* is a periodic-frequent pattern, PFPM-C visits its child node *pqt* and determines it as a periodic-frequent pattern. (d) The child node, *pqtr*, is later visited and pruned as it fails to be a periodic-frequent pattern. (e) The process of depth-first search performed n on the itemset lattice of *p*, *q*, *t*, *r*, and *s* items.

sorted based on their *support* in ascending order. In the final step of the algorithm, the timestamp information for each item is replaced with its complement information, which has a time complexity of $\mathcal{O}(a)$, where a is the total number of transactions (or worst case size of a TS-list). This process is repeated for all items in the CTS-List, resulting in a time complexity of $\mathcal{O}(c)$. Overall, the complexity of the initial Algorithm is $\mathcal{O}(ab) + \mathcal{O}(c) = \mathcal{O}(ab)$, where a represents the number of transactions and b represents the average transaction length.

Once the CTS-List (or one-length PFPs) has been identified, we generate combinations of items to form larger PFPs. This is accomplished using procedures outlined in Algorithm 2. This algorithm consists of two steps. In the first step, it accesses two items and compares their $(d-1)$ itemset complement TS-lists to generate a d-itemset complement TS-list. This step has a complexity of $\mathcal{O}(c^2)$, where c represents the number of unique items. Notably, in dense databases where the original TS-list size is large, the size of the complement TS-list is relatively small. As a result, the d-itemset construction process is faster, especially in dense databases. The second step involves calculating each itemset's *periodicity* and *support* and discarding uninteresting patterns based on user-specified *minSup* and *maxPRD* criteria. The overall complexity of Algorithm 2 is $\mathcal{O}(c^2)$.

In conclusion, the PFPM-C algorithm has a complexity of $\mathcal{O}(c^2)$ for finding all the PFPs. This efficiency makes PFPM-C a highly effective method for PFPM in dense temporal databases.

V. EXPERIMENTAL RESULTS

In this section, we show the results of conducted experimentation. The proposed algorithm PFPM-C is evaluated against the state-of-art algorithms PFP-growth [1], PFP-growth++ [3], and PFECLAT [14]) in terms of runtime requirements and memory consumption. We conducted experiments on various real-world dense databases by varying *minSup* and *maxPRD* thresholds.

A. EXPERIMENTAL SETUP

In this subsection, we explain the complete environment details of the experimentation. The configuration of the server machine (Gigabyte R282-z94 rack) is as follows: equipped with two AMD EPIC 7542 CPUs and 600 GB RAM, running on Ubuntu Server OS 20.04. All algorithms were written in Python 3.7. On both synthetic (**C20D10K**) and real-world (**Chess**, **Connect**, **PUMSB**, **Mushroom**, and **Pollution**) databases were utilized to conduct the experiments. Mostly, dense databases are taken for experimentation as PFPM-C performance is poor in sparse databases.

The **C20D10K** is a synthetic database generated using the procedure described in [21]. The **Chess** database is a high-dimensional real-world database containing 75 items and 3196 transactions prepared from the UCI Chess database. The **Connect** is a dense real-world database that contains all positions in the game of Connect-4 prepared from the UCI Connect database. The **Mushroom** is a dense real-world database containing different species of grilled mushrooms prepared from the UCI mushrooms dataset. The **PUMSB** is a real-world database containing census data for population and housing with 49,046 transactions.

Many cardio-respiratory issues are caused by air pollution. The Japanese Ministry of the Environment developed the Atmospheric Environmental Regional Observation System (AEROS) to tackle air pollution problems. Several air pollution measurement sensors are scattered around Japan as part of this system. Each station collects the data of various air pollutants, say $PM_{2.5}$, NO_2, and O_3, hourly. For our experiment, we confine to $PM_{2.5}$ since particle size is the primary contributor to the wide variety of cardio-respiratory issues experienced by Japanese citizens. According to Air Quality Index Standards, PM2.5 values greater than 16 $\mu g/m^3$ per hour are unsuitable for the people. Consequently, the hourly raw data of $PM_{2.5}$ was then transformed into a temporal database.

TABLE 3. Statistics of the databases.

S.No	Database	Type	Nature	Sparsity value	Transaction Length (in count) min.	avg.	max.	Database Size (in count)
1	Chess	Real	Dense	0.494	37	37	37	3196
2	Connect	Real	Dense	0.334	43	43	43	67545
3	PUMSB	Real	Dense	0.976	74	74	74	49,046
4	Mushroom	Real	Dense	0.806	23	23	23	8326
5	C20D10K	Synthetic	Dense	0.895	20	20	20	10,000
6	Pollution	Real	Dense	0.5545	11	458	971	717

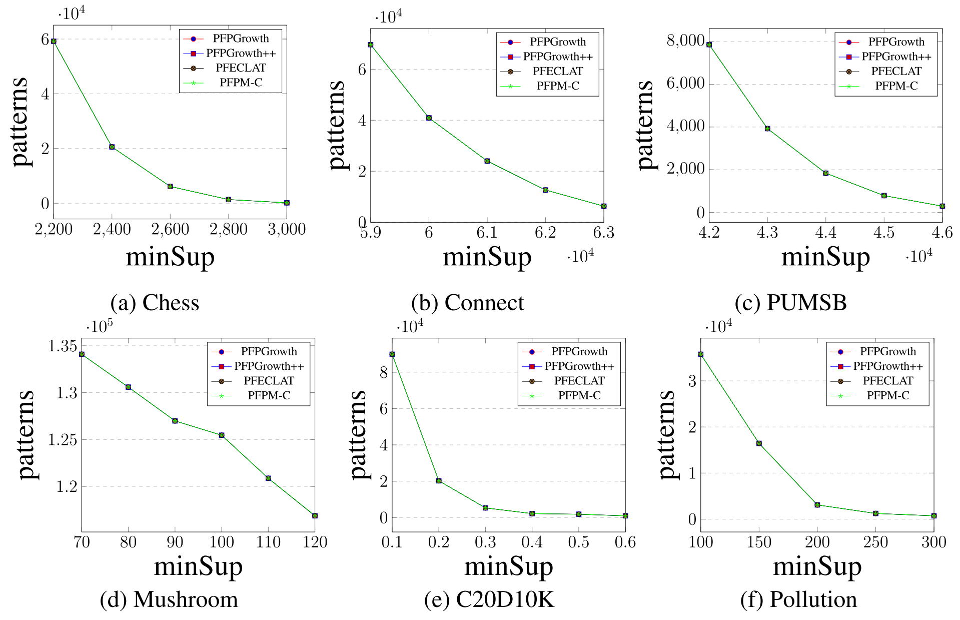


FIGURE 3. Number of patterns generated at different *minSup* thresholds.

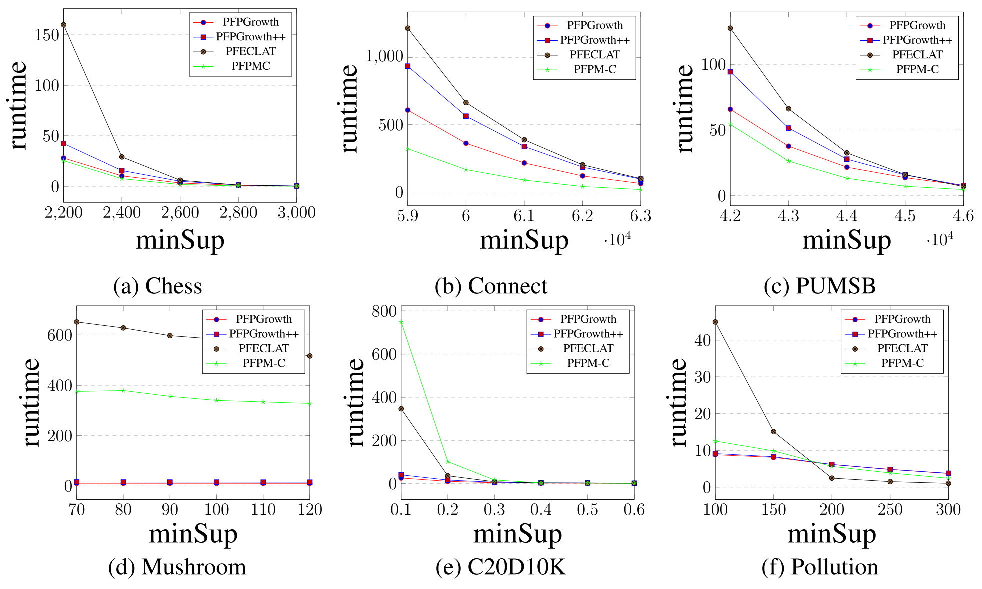


FIGURE 4. Runtime evaluation across various *minSup* thresholds.

As there is no optimal way to identify the appropriate values for the user-defined parameters, the parameters were chosen based on the statistics of the database. The statistics of all databases are shown in Table 3. All algorithms

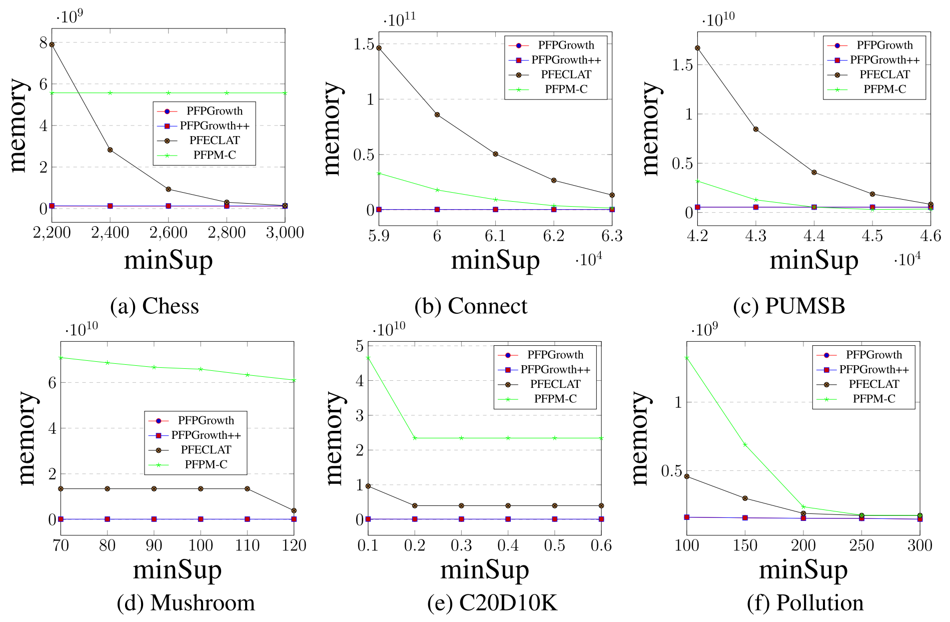


FIGURE 5. Memory consumption across various *minSup* thresholds.

employed for evaluation purposes were made available in the GitHub-hosted PAttern MIning (PAMI) repository [43] to verify the repeatability of our experiments.

B. EVALUATION OF ALGORITHMS BY VARYING MINSUP

In this first experiment, we evaluate PFPM-C against PFP-growth, PFP-growth++, PS-growth, and PFECLAT algorithms by varying only the *minSup* constraint in each database. The *maxPRD* value in each database will be set at a particular value. The *maxPRD* (specified in count) in Chess, Connect, PUMSB, Mushroom, C20D10K, C73D10K, and Pollution has been set at 400, 2000, 14, 2500, 5000, 4000, and 45, respectively.

Fig. 3 illustrates the total number of periodic-frequent patterns generated by PFP-growth, PFP-growth++, PFECLAT, and PFPM-C in different databases by varying *minSup* values. The X-axis denotes the *minSup*, and the Y-axis denotes the number of periodic-frequent patterns generated at a particular *minSup* value. Based on this figure, the following observations can be made:

1) Since all algorithms have generated the same number of periodic-frequent patterns, the line plots have overlapped with one another.
2) Increasing the *minSup* threshold value results in the generation of fewer periodic-frequent patterns. This is because many patterns fail to satisfy the increased *minSup* value. More importantly, long patterns fail to meet the increased *minSup* value due to the downward closure property.

The results of runtime requirements are shown in Fig. 4. These graphs help us to analyze the relationship between *minSup* and runtime required by all algorithms at each *minSup*. The X-axis denotes the *minSup*, and the Y-axis denotes the runtime requirement of the algorithms at a particular *minSup* value. Based on this figure, the following observations can be made:

1) The increase in value of *minSup* tends to decrease the runtime of all algorithms. This is because, as the *minSup* increases, the number of periodic-frequent patterns generated will decrease, automatically requiring less runtime.
2) Through these graphs, we note that PFPM-C performs strongly in dense real-world databases like Chess, Connect, PUMSB, Mushroom, and Pollution. However, in the case of the sparse C20D10K database, PFPM-C requires higher runtime when compared to state-of-the-art algorithms.
3) PFPM-C excels in dense databases primarily due to the pervasive occurrence of items in all transactions, resulting in shorter lengths of complement sets. Conversely, in sparse databases, the lengths of complement sets tend to be more extensive. Overall, complement sets play a pivotal role in reducing the runtime requirements to determine the periodicity of a pattern.

The memory consumption results are presented in Fig. 5. The graph allows us to analyze the relationship between *minSup* and the memory consumption to mine the periodic-frequent patterns. The Y-axis represents the memory requirements of PFP-growth, PFP-growth++, PFECLAT, and PFPM-C algorithms. The following observations can be made based on this figure:

1) Increase in *minSup* decreases the memory requirements of all algorithms. This is because fewer patterns will be generated with the increase in *minSup*.

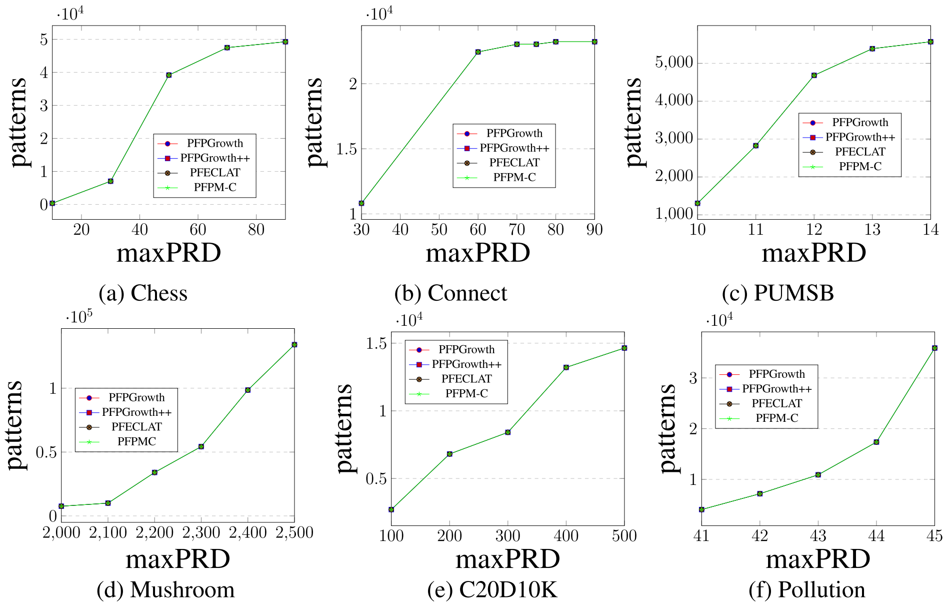


FIGURE 6. Number of patterns by varying *maxPRD* in different databases.

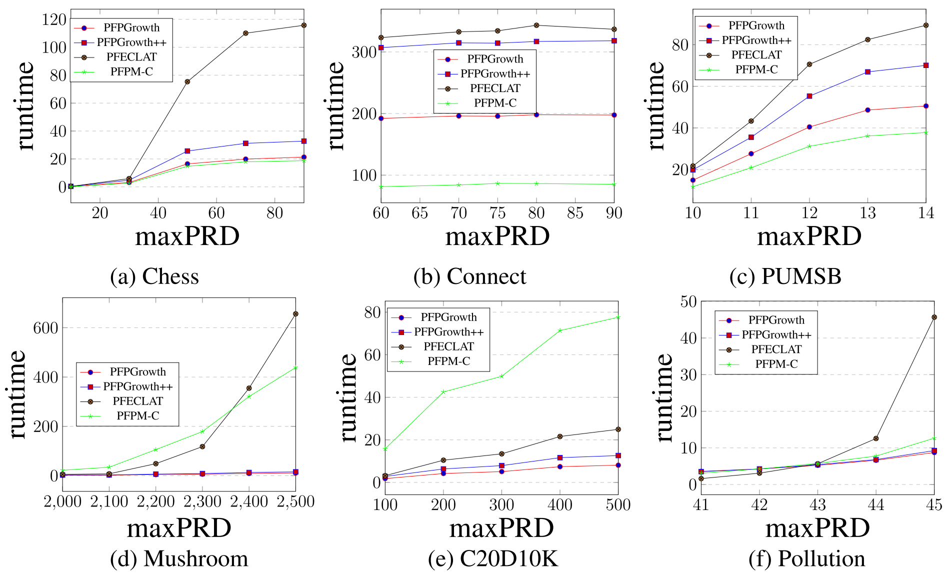


FIGURE 7. Runtime evaluation by varying *maxPRD* in different databases.

2) The algorithm PFP-growth and PFP-growth++ algorithms consumed less memory in all databases, whereas PFPM-C is more efficient than PFECLAT regarding memory.
3) PFPM-C has more advantages in large databases like PUMSB and Connect.

C. EVALUATION OF ALGORITHMS BY VARYING MAXPRD CONSTRAINT

We evaluated the PFP-growth, PFP-growth++, PFECLAT, and PFPM-C algorithms in the previous subsection by varying only the *minSup* value. We now evaluate the performance of these algorithms by varying only the *maxPRD* constraint in each of the databases. The *minSup* in Chess, Connect, PUMSB, Mushroom, C20D10K, and Pollution and Retail databases have been set at 2200, 60000, 42500, 70, 2000, and 50, respectively.

Fig. 6 illustrates the total number of periodic-frequent patterns generated by PFP-growth, PFP-growth++, PFECLAT, and PFPM-C in different databases by varying *maxPRD* values. The X-axis denotes the *maxPRD*, and the Y-axis denotes the number of periodic-frequent patterns generated at a particular *maxPRD* value. Based on this figure, we can say that raising the *maxPRD* threshold

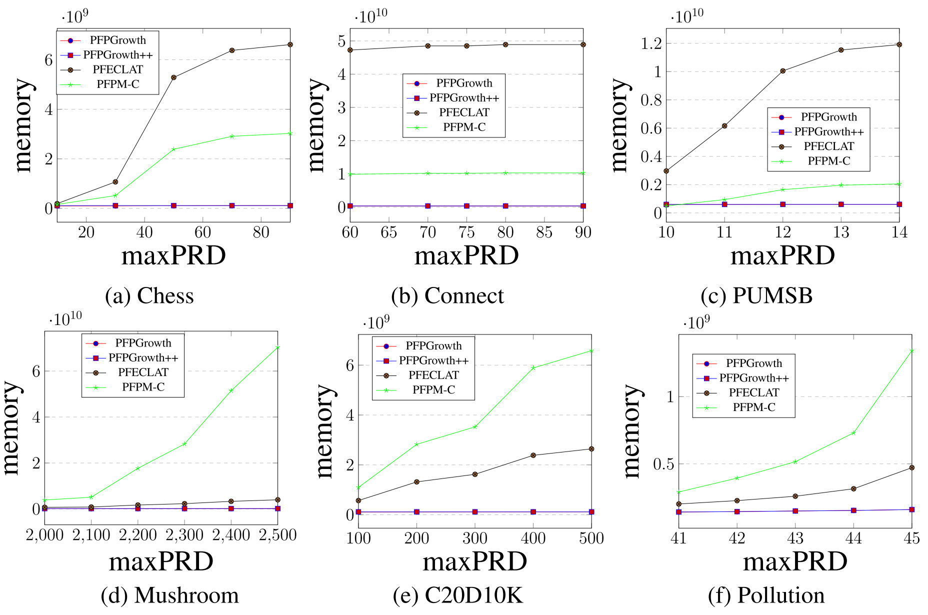


FIGURE 8. **Memory evaluation by varying *maxPRD* in different databases.**

results in more periodic-frequent patterns, as many patterns meet the *maxPRD* criteria. All algorithms generate the same set of periodic-frequent patterns at a given *maxPRD* value.

The results of runtime requirements of various algorithms are shown in Fig. 7. These graphs help us to analyze the relationship between *maxPRD* and runtime required by all algorithms at each *maxPRD*. The X-axis denotes the *maxPRD*, and the Y-axis denotes the runtime requirement of PFP-growth, PFP-growth++, PFECLAT, and PFPM-C at a particular *maxPRD* value. Based on this figure, the following observations can be made:

1) Increase in *maxPRD* increases the runtime requirements of all mining algorithms. This is because of the increase in the number of patterns being generated.
2) Except in the sparse C20D10K database, the PFPM-C algorithm performed very well in all other databases and took less time to mine the patterns than other state-of-the-art algorithms.

The memory consumption results are presented in Fig. 8. The graph allows us to analyze the relationship between *maxPRD* and the memory consumption to mine the periodic-frequent patterns. The Y-axis represents the memory requirements of PFP-growth, PFP-growth++, PFECLAT, and PFPM-C algorithms. The following observations can be made based on this figure:

1) Increase in *maxPRD* increases the memory requirements of the mining algorithms. This is because the mining algorithms must generate and store more patterns in the memory.
2) The tree-based algorithms, PFP-growth and PFP-growth++, consumed less memory than PFECLAT and PFPM-C algorithms.
3) PFPM-C consumes less memory than PFECLAT in some databases and vice-versa.

VI. CONCLUSION AND FUTURE WORK

An efficient algorithm, PFPM-C, was introduced in this paper to efficiently find periodic-frequent patterns in a database. This algorithm exploited the notion of "set complements" to reduce the size of patterns' timestamps lists. We introduce a new property to calculate the *periodicity* of a pattern from its complement temporal information. We studied the performance of the PFPM-C algorithm on various real-world and synthetic databases. Empirical results demonstrate that PFPM-C is memory efficient and can obtain all periodic-frequent patterns faster against state-of-the-art algorithms.

This paper focused on developing a sequential CPU-based algorithm to find periodic-frequent patterns in a temporal database. As a part of future work, we would like to investigate approaches to find the patterns in a distributed fashion. In future work, we would like to explore new measures or techniques to further reduce the computational cost of mining the periodic-frequent patterns.

REFERENCES

[1] S. K. Tanbeer, C. F. Ahmed, B. Jeong, and Y. Lee, "Discovering periodic-frequent patterns in transactional databases," in *Proc. 13th Pacific-Asia Conf. Knowl. Discovery Data Mining*, in Lecture Notes in Computer Science, vol. 5476. Cham, Switzerland: Springer, 2009, pp. 242–253.
[2] 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, Jan. 2018.

[3] R. U. Kiran and P. K. Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *Proc. 21st Int. Conf. Database Expert Syst. Appl.*, in Lecture Notes in Computer Science, vol. 6262. Cham, Switzerland: Springer, 2010, pp. 194–208.

[4] P. Fournier-Viger, P. Yang, R. U. Kiran, S. Ventura, and J. M. Luna, "Mining local periodic patterns in a discrete sequence," *Inf. Sci.*, vol. 544, pp. 519–548, Jan. 2021.

[5] T. Sreepada, R. Uday Kiran, Y. Watanobe, and K. Goda, "A novel GPU-accelerated algorithm to discover periodic-frequent patterns in temporal databases," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Osaka, Japan, Dec. 2022, pp. 121–126.

[6] X. Zhang, Y. Qi, G. Chen, W. Gan, and P. Fournier-Viger, "Fuzzy-driven periodic frequent pattern mining," *Inf. Sci.*, vol. 618, pp. 253–269, Dec. 2022.

[7] H. N. Dao, P. Ravikumar, P. Likhitha, U. K. Rage, Y. Watanobe, and I. Paik, "Finding stable periodic-frequent itemsets in big columnar databases," *IEEE Access*, vol. 11, pp. 12504–12524, 2023.

[8] R. U. Kiran, P. Fournier-Viger, J. M. Luna, J. C.-W. Lin, and A. Mondal, *Periodic Pattern Mining : Theory, Algorithms, and Applications*. Cham, Switzerland: Springer, 2021. [Online]. Available: https://link.springer.com/book/10.1007/978-981-16-3964-7(visited 2023-06-05).

[9] A. Anirudh, R. U. Kirany, P. K. Reddy, and M. Kitsuregaway, "Memory efficient mining of periodic-frequent patterns in transactional databases," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Athens, Greece, Dec. 2016, pp. 1–8.

[10] A. Anirudh, R. U. Kiran, P. K. Reddy, M. Toyoda, and M. Kitsuregawa, "An efficient map-reduce framework to mine periodic frequent patterns," in *Proc. 19th Int. Conf. Big Data Anal. Knowl. Discovery*, in Lecture Notes in Computer Science, vol. 10440. Cham, Switzerland: Springer, 2017, pp. 120–129.

[11] K. Amphawan, A. Surarerks, and P. Lenca, "Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree," in *Proc. 3rd Int. Conf. Knowl. Discovery Data Mining*, Jan. 2010, pp. 245–248.

[12] V. M. Nofong and J. Wondoh, "Towards fast and memory efficient discovery of periodic frequent patterns," *J. Inf. Telecommun.*, vol. 3, no. 4, pp. 480–493, Oct. 2019.

[13] M. K. Afriyie, V. M. Nofong, J. Wondoh, and H. Abdel-Fatao, "Efficient mining of non-redundant periodic frequent patterns," *Vietnam J. Comput. Sci.*, vol. 8, no. 4, pp. 455–469, Nov. 2021.

[14] P. Ravikumar, P. Likhitha, B. V. V. Raj, R. U. 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. [Online]. Available: https://www.mdpi.com/2079-9292/10/12/1478

[15] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Fast and space-preserving frequent pattern mining in large databases," *IIE Trans.*, vol. 39, no. 6, pp. 593–605, Mar. 2007, doi: 10.1080/07408170600897460.

[16] P. Braun, J. J. Cameron, A. Cuzzocrea, F. Jiang, and C. K. Leung, "Effectively and efficiently mining frequent patterns from dense graph streams on disk," *Proc. Comput. Sci.*, vol. 35, pp. 338–347, Jan. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914010795

[17] L. Vu and G. Alaghband, "Efficient algorithms for mining frequent patterns from sparse and dense databases," *J. Intell. Syst.*, vol. 24, no. 2, pp. 181–197, Jun. 2015, doi: 10.1515/jisys-2014-0040.

[18] J. M. Luna, P. Fournier-Viger, and S. Ventura, "Frequent itemset mining: A 25 years review," *WIREs Data Mining Knowl. Discovery*, vol. 9, no. 6, pp. 1–12, Nov. 2019.

[19] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2003, pp. 326–335.

[20] P. Veena, S. Tarun, R. U. Kiran, M.-S. Dao, K. Zettsu, Y. Watanobe, and J. Zhang, "Towards efficient discovery of periodic-frequent patterns in dense temporal databases using complements," in *Database and Expert Systems Applications*. Cham, Switzerland: Springer, 2022, pp. 204–215.

[21] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. data*, New York, NY, USA, Jun. 1993, pp. 207–216, doi: 10.1145/170035.170072.

[22] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, Feb. 2000.

[23] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD Int. Conf. Manage. data*, Dallas, TX, USA, May 2000, pp. 1–12.

[24] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *Proc. 14th Int. Conf. Data Eng.*, 1998, pp. 412–421.

[25] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases," in *Proc. 4th Int. Conf. Knowl. Discovery Data Mining*, 1998, pp. 214–218.

[26] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Proc. 15th Int. Conf. Data Eng.*, 1999, pp. 106–115.

[27] K.-J. Yang, G.-C. Lan, T.-P. Hong, and Y.-M. Chen, "Partial periodic patterns mining with multiple minimum supports," in *Proc. 9th Int. Conf. Inf., Commun. Signal Process.*, Dec. 2013, pp. 1–4.

[28] J. Yang, W. Wang, and P. S. Yu, "Mining asynchronous periodic patterns in time series data," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 3, pp. 613–628, May 2003.

[29] Y. Xun, L. Wang, H. Yang, and J. Cai, "Mining relevant partial periodic pattern of multi-source time series data," *Inf. Sci.*, vol. 615, pp. 638–656, Nov. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025522011690

[30] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 259–289, 1997, doi: 10.1023/A:1009748302351.

[31] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex sequences," *Inf. Syst.*, vol. 33, no. 1, pp. 96–114, Mar. 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437907000506

[32] M. Lahiri and T. Y. Berger-Wolf, "Periodic subgraph mining in dynamic networks," *Knowl. Inf. Syst.*, vol. 24, no. 3, pp. 467–497, Sep. 2010, doi: 10.1007/s10115-009-0253-8.

[33] Q. Zhang, D. Guo, X. Zhao, X. Li, and X. Wang, "Seasonal-periodic subgraph mining in temporal networks," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, New York, NY, USA, Oct. 2020, pp. 2309–2312, doi: 10.1145/3340531.3412091.

[34] R. U. Kiran, C. Saideep, P. Ravikumar, K. Zettsu, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering fuzzy periodic-frequent patterns in quantitative temporal databases," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jul. 2020, pp. 1–8.

[35] P. Fournier-Viger, P. Yang, J. C. Lin, and R. U. Kiran, "Discovering stable periodic-frequent patterns in transactional data," in *Proc. IEA/AIE*, in Lecture Notes in Computer Science, vol. 11606. Cham, Switzerland: Springer, 2019, pp. 230–244.

[36] P. Likhitha, R. Veena, R. U. Kiran, K. Zettsu, M. Toyoda, and P. Fournier-Viger, "UPFP-growth++: An efficient algorithm to find periodic-frequent patterns in uncertain temporal databases," in *Proc. ICONIP*, in Communications in Computer and Information Science, vol. 1792. Cham, Switzerland: Springer, 2022, pp. 182–194.

[37] P. Ravikumar, R. U. Kiran, P. Likhitha, T. Chandrasekhar, Y. Watanobe, and K. Zettsu, "Discovering geo-referenced periodic-frequent patterns in geo-referenced time series databases," in *Proc. IEEE 9th Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2022, pp. 1–10.

[38] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "RP-tree: A tree structure to discover regular patterns in transactional database," in *Intelligent Data Engineering and Automated Learning—IDEAL 2008*. Berlin, Germany: Springer, 2008, pp. 193–200.

[39] S. K. Tanbeer, C. F. Ahmed, and B.-S. Jeong, "Mining regular patterns in incremental transactional databases," in *Proc. 12th Int. Asia–Pacific Web Conf.*, Apr. 2010, pp. 375–377.

[40] 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.

[41] D.-T. Dinh, B. Le, P. Fournier-Viger, and V.-N. Huynh, "An efficient algorithm for mining periodic high-utility sequential patterns," *Appl. Intell.*, vol. 48, no. 12, pp. 4694–4714, Dec. 2018, doi: 10.1007/s10489-018-1227-x.

[42] 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.

[43] R. U. Kiran. (2023). *Pattern Mining (PAMI)*. Accessed: Oct. 16, 2023. [Online]. Available: https://github.com/UdayLab/PAMI/tree/main

PAMALLA VEENA received the M.C.A. degree (Hons.) from the Sri Balaji PG College, India. She has published papers in the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), the International Conference on Data Mining Workshops (ICDMW), the IEEE International Conference on Big Data (IEEE Big Data), and published journal articles in *Applied Intelligence* and *Electronics*.

TARUN SREEPADA (Member, IEEE) is currently pursuing the bachelor's degree in computer science and engineering with The University of Aizu, Aizuwakamatsu, Japan. He has published papers in DEXA and IEEE International Conference on Big Data (IEEE Big Data).

RAGE UDAY KIRAN (Senior Member, IEEE) received the Ph.D. degree in computer science from the International Institute of Information Technology, Hyderabad, Telangana, India. He is currently an Associate Professor with The University of Aizu, Aizuwakamatsu, Fukushima, Japan. He has published more than 100 papers in refereed journals and international conferences, such as EDBT and PAKDD. His research interests include data mining and machine learning.

MINH-SON DAO received the Ph.D. degree from the Dipartimento Informatica e Telecomunicazioni (DIT), Università degli studi di Trento, Trento, Italy, in February 2005. From 2005 to 2007, he was a Scientist with GraphiTech, Italy. From 2007 to 2010, he was a JSPS Post-doctoral Researcher with the Media Integrated Communication Laboratory (MICL), Graduate School of Engineering, Osaka University. From 2010 to 2013, he was a Researcher with the Multimedia Signal Processing and Understanding Laboratory (mmLAB), University of Trento, Italy. He is currently a Senior Researcher with the Big Data Analytics Laboratory, National Institute of Information and Communications Technology (NICT), Japan. He also leads several national and international projects under the Society 5.0 framework. His main interests include multimedia retrieval, event detection, video surveillance, data mining, computer vision, and pattern recognition.

KOJI ZETTSU received the Ph.D. degree in informatics from Kyoto University, in 2005. He has been researching and developing data analytics technology with the National Institute of Information and Communications Technology (NICT), where he has also been leading the Real Space Information Analytics Project, since 2016, to implement a smart data platform based on data mining and AI. For promoting industry-academia-government collaboration on the platform, he is a leader of the Cross-Data Collaboration Project of the Smart IoT Acceleration Forum in Japan. He is currently the Director General of the Big Data Integration Research Center, NICT. He has served on numerous academic societies, conference committees, and working groups. His research interests include database systems, data mining, information retrieval, and software engineering.

YUTAKA WATANOBE (Member, IEEE) is currently a Senior Associate Professor with the School of Computer Science and Engineering, The University of Aizu, Japan. His research interests include visual programming language, data mining, and cloud robotics.

JI ZHANG (Senior Member, IEEE) is currently a Full Professor in computer science with the University of Southern Queensland (UniSQ), Australia. He has published more than 300 papers in major peer-reviewed international journals and conferences, including IEEE Transactions on Knowledge and Data Engineering, *ACM Transactions on Knowledge Discovery from Data*, *ACM Transactions on Intelligent Systems and Technology*, AAAI, IJCAI, *The VLDB Journal*, CIKM, SIGKDD, ICDE, ICDM, and WWW. His research interests include big data analytics, knowledge discovery and data mining (KDD), and computational intelligence. He is an IET Fellow, BCS Fellow, RSA Fellow, Australian Endeavour Fellow, Queensland International Fellow, Australia, and Izaak Walton Killam Scholar, Canada.