IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 September 2023, accepted 12 October 2023, date of publication 19 October 2023, date of current version 1 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3325839

RESEARCH ARTICLE

k-PFPMiner: Top-k Periodic Frequent Patterns in Big Temporal Databases

PALLA LIKHITHA, (Member, IEEE), PENUGONDA RAVIKUMAR, DEEPIKA SAXENA, (Member, IEEE), RAGE UDAY KIRAN, (Senior Member, IEEE), AND YUTAKA WATANOBE, (Member, IEEE)
Division of Information Systems, The University of Aizu, Aizu-Wakamatsu, Fukushima 965-0006, Japan

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 Finding periodic-frequent patterns in temporal databases is a prominent data mining problem with bountiful applications. It involves discovering all patterns in a database that satisfy the user-specified *minimum support* (*min_sup*) and *maximum periodicity* (*max_per*) constraints. *Min_sup* controls the least number of transactions in which a pattern must appear in a database. *Max_per* controls the maximum time interval within which a pattern must reappear in the database. The popular adoption of this task has been hindered by an open problem, which involves setting appropriate *min_sup* and *max_per* values for any given database. This paper addresses this open problem by proposing a solution to discover top-*k* periodic-frequent patterns in a temporal database. Top-*k* periodic-frequent patterns represent the *k* number of periodic-frequent patterns having the lowest *periodicity* value in a database. An efficient depth-first search algorithm, Top-*k* Periodic-Frequent Pattern Miner (*k*-PFPMiner), which takes only *k* threshold as an input, was presented to find all desired patterns in a database. Experimental results on synthetic and real-world databases demonstrate that our algorithm is efficient and scalable.

INDEX TERMS Data mining, pattern mining, temporal databases, top-k, periodic-frequent patterns.

I. INTRODUCTION

Pattern mining is an important sub-field of data mining. It involves discovering all user interest-based patterns hidden in a database. Frequent pattern mining (FPM) [1] is a fundamental knowledge discovery technique introduced as a key intermediary step to discover association rules between the items in a database. It involves discovering all patterns having *support* no less than the user-specified *minimum support* (*min_sup*) value. The *min_sup* is a hyper-parameter that controls all the minimum number of transactions in which a pattern must appear in a database. It is also an important parameter that controls the number of patterns discovered in a database. Unfortunately, setting this parameter for any database is an open research problem. When confronted with this problem in real-world applications, researchers try to solve it by finding top-*k* frequent patterns [2] as the constraint *k* is relatively easier to specify than the *min_sup* value. Since then, the problem of finding top-*k* frequent patterns has received much attention [2]. A classic application of top-*k* frequent pattern mining is market-basket analysis. It involves identifying the top-*k* frequently purchased itemsets by customers. An example of a top-*k* frequent pattern is as follows:

$$\{Milk, Beer\}\ [support = 10\%].$$

The above pattern indicates that 10% of the customers have purchased the items 'Milk' and 'Beer' together. This knowledge may help managers to make profitable business decisions, such as product placement and campaigning.

Over the years, the FPM has inspired many knowledge discovery techniques, such as closed frequent pattern mining [3], maximal frequent pattern mining [4], fuzzy frequent pattern mining [5], high utility pattern mining [6], frequent sequence pattern mining [7], and generators [8]. However, the wide societal adoption of this technique has been hindered by

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

its inability to consider the temporal occurrence information of an item in a database. Tanbeer et al. [9] tried to solve this problem by extending the FPM to discover periodic-frequent patterns (PFPs) in a temporal database. An example of a PFP is as follows:

$$\{Milk, Beer\}\ [support = 80\%, periodicity = 2\%].$$

The pattern indicates that items are sold every 2 hours with 80% support. Based on the identified PFP, the retailer can send text reminders to customers on weekends, encouraging them to purchase the associated items and providing discounts as an incentive. This proactive approach aims to enhance customer engagement, foster loyalty, and drive sales. Furthermore, in the literature, the model of PFP was extended to find local periodic patterns [10], periodic sequential patterns [11], fuzzy PFPs [12], maximal PFPs [13], recurring patterns [14], geo-referenced PFPs (GPFPs) [15], fuzzy GPFPs [16] and stable periodic patterns [17], [18], [19]. However, there are certain limitations of PFPM. The two significant limitations of PFP are: First, generating too many patterns that are uninteresting to the user. Secondly, relying on *min_sup* and *max_per* constraints to extract the desired patterns.

This paper presents a novel approach to address the above-mentioned limitations in PFPM. Rather than relying on the conventional *min_sup* and *max_per* constraints, we propose utilizing a single constraint k to discover desired patterns. This constraint represents the selection of the top-k most PFPs. By adopting this approach, we mitigate the challenge of selecting appropriate *minSup* and *maxPer* values, which often require prior knowledge of the database characteristics. Instead, the user can leverage the top-k PFPs, which are helpful to the user in analyzing the pattern behavior.

The contributions of this paper are as follows:

1) This paper proposes a novel model of top-k PFPs that may exist in a temporal database. Informally, a PFP is a pattern that frequently occurs at regular intervals in a temporal database. The top-k PFPs represent the k number of PFPs with the lowest *periodicity* in the entire database.
2) The space of an itemset lattice represents the search space of finding PFPs. The size of this search space is $2^n - 1$, where n represents the total number of items in a temporal database. Reducing this enormous search space is challenging without *min_sup* and *max_per* constraints. Our model employs a new upper-bound measure, *dynamic maximum periodicity*, to reduce the search space effectively. Please note that our model's upper-bound value will automatically be calculated and updated without human intervention.
3) This paper presents two novel pruning techniques to reduce the computational cost of finding the PFPs. Both of these techniques rely on *dynamic maximum periodicity*. The objective of the first pruning technique is to prune the search space in the itemset lattice effectively. The objective of the second pruning technique is to determine whether a pattern is periodic or aperiodic in the database effectively. Traditionally, the time complexity to determine whether a pattern is periodic or aperiodic in the database is $\mathcal{O}(m)$, where m represents the number of occurrences of a pattern in the database.
4) An efficient single-pass algorithm using a best-first search strategy, called top-k PFP Miner (k-PFPMiner), is proposed to find all desired patterns in a database.
5) Experimental results on synthetic and real-world databases demonstrate that our algorithm is memory and runtime efficient and also highly scalable.

This paper is an expanded version of a previous conference paper [20], providing a brief overview of the literature on this topic. In this paper, we have extended the related work by comprehensively reviewing the current literature. Furthermore, we report new experimental findings that demonstrate the superior performance of k-PFPMiner over the naïve on various databases, regardless of the k value.

The rest of this paper is organized as follows. Section II describes the related work on finding top-k PFPs in databases. Section III describes the proposed model to find top-k PFPs in databases. Section IV describes the proposed algorithm to discover the top-k PFPs. Section VI presents the experimental results obtained. Finally, in section VII, we conclude and discuss future research.

II. RELATED WORK

In this section, we briefly look at past studies about frequent pattern mining, periodic-frequent pattern mining, and top-k periodic-frequent pattern mining.

A. FREQUENT PATTERN MINING

Frequent pattern mining is a fundamental technique in the field of big data analytics, finding extensive applications across various domains such as bio-informatics [21], market basket analysis [22], energy reduction in smart homes [23], malware analysis [24], webpage click-stream analysis [25], proof sequence analysis [26], and text analysis [27]. The goal of frequent pattern mining is to discover patterns that occur frequently in a given transactional database, enabling valuable insights and knowledge extraction. Several well-known algorithms, including Apriori [1], FP-growth [28], and Eclat [29], have been developed to tackle the challenge of finding frequent patterns efficiently. These algorithms employ different strategies and have varying degrees of scalability and efficiency, offering flexibility for different data scenarios [27]. However, while frequent pattern mining techniques have proven effective in uncovering commonly occurring patterns, they may not adequately capture patterns that exhibit consistent temporal behavior. Traditional frequent pattern mining approaches focus solely on identifying patterns that occur frequently, without considering the temporal dimension of the data. This limitation hinders their ability to detect patterns that occur consistently over time or exhibit periodicity. To address this gap, researchers

have proposed alternative techniques, such as temporal data mining and periodic-frequent pattern mining, to specifically target temporal patterns. These approaches take into account the temporal variations and dependencies within the data, allowing for the discovery of patterns that exhibit temporal consistency or periodicity. By incorporating time-related information into the mining process, these methods identify valuable patterns that may have significant implications in various domains, such as understanding customer behavior over time or detecting recurring events in temporal databases [9].

B. PERIODIC FREQUENT PATTERN MINING

Periodic-frequent pattern mining considers temporal patterns that occur periodically or cyclically in databases. Tanbeer et al. [9] proposed a novel algorithm, periodic frequent pattern-growth, to identify periodic-frequent patterns (PFPs) in transactional databases. Their algorithm utilizes the PF-tree, a tree data structure with tail nodes storing transaction identifiers for each pattern. Pruning involves moving the tail node to the parent node, enabling efficient pattern storage. PFPs are generated using a measure based on *min_sup* and *max_per*. The authors claimed efficiency in their mining process and demonstrated effective identification of PFPs in large databases.

Kiran and Kitsuregawa [30] proposed the ExPF-growth algorithm for identifying PFPs using the *minimum periodic ratio* measure. The algorithm utilizes potential patterns and employs an ExPF-list and a prefix tree to store transactional identifiers. It recursively expands patterns, checking their *periodicity* and *support* thresholds and pruning uninteresting patterns. The ExPF-growth algorithm terminates when no more patterns can be expanded or when all patterns have been mined.

Kiran et al. [31], [32] proposed the PFP-growth++ algorithm for mining PFPs from large transactional databases. The algorithm utilizes the PF-tree++ data structure, an extension of the PF-tree, to efficiently store patterns. A significant contribution of the algorithm is the introduction of *local periodicity*, which allows for early termination when no new periodic patterns can be found. By leveraging the PF-tree++ structure, *local periodicity* concept, and pruning techniques, the PFP-growth++ algorithm can handle large databases and reduce unnecessary computations.

Anirudh et al. [33] proposed PS-growth, a memory-efficient algorithm for mining PFPs in sparse databases. It uses periodic summaries to reduce search space and memory usage. Transactions are divided into intervals based on *periodicity*, and candidate patterns are generated using these summaries. PS-growth achieves faster mining and lower memory consumption by reducing the number of candidate patterns at each level of the search tree.

Surana et al. [34] introduced the MaxCPF model for discovering PFPs with constraints on maximum length or size. They developed the MaxCPF-tree, a modified version of the FP-tree, to efficiently mine these patterns. The MaxCPF-tree stores pattern information in a compressed form, reducing memory usage and improving efficiency. They also proposed the MaxCPF-List to filter out false positives during mining. This approach effectively identifies frequent and periodic patterns while considering constraints.

Ravikumar et al. [35], [36] proposed PF-ECLAT, an extension of the ECLAT algorithm, for mining PFPs in columnar temporal databases. It employs a list-based approach and utilizes pruning techniques to efficiently discover interesting patterns.

C. TOP-K PERIODIC PATTERN MINING

Top-k periodic-frequent pattern mining involves finding the k most periodic-frequent patterns in a database. Amphawan et al. [37] proposed MTKPP, a non-support metric-based algorithm for discovering PFPs. It utilizes a sliding window technique and a top-k list structure to identify the k most frequent PFPs. MTKPP employs a best-first strategy, pruning unlikely candidates until either k patterns are found or no further pruning is possible. Its uniqueness lies in not relying on a *support* metric, making it suitable for diverse applications.

Viger et al. [17] proposed *TSPIN* to find the top-k stable periodic patterns. It considers user-specified constraints *max_per*, and *maxLA* in addition to k. It stores all the transactions in SPP-tree and internal *min_sup* as 1. SPP-tree is mined recursively, the lowest frequent pattern is removed from the queue, and *min_sup* is raised to the least frequent pattern in the remaining queue. However, the above studies still require users to specify the other constraints in addition to k.

III. MODEL OF TOP-*K* PERIODIC-FREQUENT ITEMSET

Let O denote the set of objects (or items). An itemset $P \subseteq O$ is referred to as an itemset or pattern. If an itemset contains α, $\alpha \geq 1$, number of items, it is known as an α-pattern. In a transaction, represented by $t_k = (ts,\ X)$, the tuple contains the timestamp ts indicating when the pattern X occurred. A temporal database TDB over O is a collection of transactions, specifically $TDB = \{tr_1, \cdots,\ tr_m\}$, where $m = |TDB|$ represents the number of transactions in TDB. For a transaction $tr_k = (ts,\ X)$ with $k \geq 1$ and a pattern $Z \subseteq X$, it is said that Z occurs in tr_k (or tr_k contains Z), and the timestamp associated with this occurrence is denoted as ts^Z. Let $TS^Z = \{ts^Z_j, \cdots, ts^Z_k\}$, with $j,\ k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** indicating the occurrences of pattern Z in the temporal database TDB.

Example 1: Consider a set of items $O = \{p, q, r, s, t\}$. Table 1 shows a temporal database constructed from these items. We are interested in the pattern $\{r,\ p\}$, which we represent as '*rp*' for brevity. This pattern is referred to as a 2-pattern due to the presence of two items. In the given database, the pattern '*rp*' appears at the timestamps 1, 3, 4, 6, 7, 8, and 9. Therefore, the list of timestamps

TABLE 1. Temporal database.

ts	items
1	pqr
2	qrs
3	pqrs
4	pqrt
5	qr
6	prs
7	pqrst
8	pr
9	pqr
10	st

in which the pattern '*rp*' occurs is denoted as $TS^{rp} = \{1, 3, 4, 6, 7, 8, 9\}$.

*Definition 1 (**Periodicity of** Z):* A *period* of Z in *TDB* is calculated using the following three ways: (*i*) $p_1^Z = ts_a^Z - ts_{min}$, (*ii*) $p_i^Z = ts_q^Z - ts_p^Z$, where $2 \leq i \leq |TS^Z|$ and $a \leq p \leq q \leq c$ represent the periods (or inter-arrivals) of Z in the database, and (*iii*) $p_{|TS^Z|+1}^Z = ts_{max} - ts_c^Z$. The maximal and minimal timestamps of all transactions in the database are represented as ts_{min} and ts_{max}. Let $P^Z = p_1^Z, p_2^Z, \cdots, p_k^Z$, where $k = |TS^Z| + 1$, be the set of all periods of Z in the temporal database. The periodicity of Z, denoted as $per(Z)$, is defined as the maximum value among the periods in P^Z, i.e., $per(Z) = \max(p_1^Z, p_2^Z, \cdots, p_k^Z)$.

Example 2: The periods for the pattern '*rp*' are calculated as follows: $p_1^{rp} = 1 (= 1 - ts_{initial})$, $p_2^{rp} = 2 (= 3 - 1)$, $p_3^{rp} = 1 (= 4 - 3)$, $p_4^{rp} = 2 (= 6 - 4)$, $p_5^{rp} = 1 (= 7 - 6)$, $p_6^{rp} = 1 (= 8 - 7)$, $p_7^{rp} = 1 (= 9 - 8)$, and $p_8^{rp} = 1 (= ts_{final} - 9)$, where $ts_{initial} = 0$ represents the timestamp of the initial transaction and $ts_{final} = |TDB| = 10$ represents the timestamp of the final transaction in the database. The periodicity of the pattern '*rp*,' denoted as $per(rp)$, is calculated as the maximum value among the periods, resulting in $per(rp) = \max(1,\ 2,\ 1,\ 2,\ 1,\ 1,\ 1,\ 1) = 2$.

*Definition 2 (**Top-K Periodic-Frequent Pattern** X):* Let $X_1, X_2, \cdots, X_k, \cdots, X_p,\ 1 \leq k \leq p \leq 2^m - 1$, be an ordered set of all patterns such that $per(X_1) \leq per(X_2) \leq \cdots \leq per(X_k) \leq \cdots \leq per(X_p)$. A pattern $X_a,\ 1 \leq a \leq p$, is considered a top-k periodic pattern if its periodicity is no greater than the periodicity of pattern X_k in the database. In other words, X_a is classified as a top-k periodic pattern if $per(X_a) \leq per(X_k)$.

Example 3: Let us assume that the current candidate top-K periodic pattern list as $\{p,\ q,\ r,\ s,\ t\}$. If the $per(rp) \leq per(t)$, the pattern *rp* is considered as top-k periodic frequent patterns.

*Definition 3 (**Problem Definition**):* The objective of top-k periodic pattern mining in a temporal database (*TDB*) is to identify the k periodic-frequent patterns with the lowest *periodicities*. The goal is to focus specifically on these top-k patterns rather than discovering all periodic-frequent patterns in the database.

IV. OUR ALGORITHM

A. BASIC IDEA: DYNAMIC MAXIMUM PERIODICITY

To address the challenge of reducing the large search space in top-k periodic-frequent pattern mining, we propose an approach that involves the following steps:

- Create an empty list called the *candidate periodic pattern-list (cPP-List)* and initialize a Max-heap data structure with its *root* set to *null*.
- Scan the database and add the patterns to the cPP-List. At the same time, update the Max-heap with the *periodicity* values of these patterns.
- As the cPP-List grows, keep track of its size. Once the size of the cPP-List reaches the desired value of k, set the dynamic maximum periodicity (*dMaxPer*) equal to the value stored at the root of the Max-heap.
- Prune the search space by applying the *dMaxPer* constraint to the itemsets. Remove any patterns that do not satisfy the constraint.
- If any pattern satisfies the constraint, add it to the cPP-List by removing the existing k-pattern and updating *dMaxPer* accordingly.
- Repeat this process until the entire search space is explored.

By employing this approach, we can effectively reduce the search space by dynamically updating the maximum periodicity value and pruning patterns that do not meet the constraint. The time complexity to determine the periodicity of a pattern is $\mathcal{O}(1)$ for each pattern and $\mathcal{O}(n)$ for the entire database, where n represents the number of timestamps or frequency of a pattern in the database. This approach allows us to focus on the most relevant top-k periodic-frequent patterns, improving the efficiency and effectiveness of the mining process. $\mathcal{O}(n)$

*Definition 4 (**Dynamic Maximum Periodicity Constraint**):* Let AP represent the set of all patterns in the database, where $AP = \{X_1, X_2, \cdots, X_{2^n-1}\}$ and n is the number of items. The patterns explored by our algorithm so far are stored in the set EP, a subset of the set of all patterns ($EP \subseteq SP$). Among the patterns explored, we have a subset EP_k that contains the top-k candidate periodic patterns found up to this point, where $|EP_k| = k$. The *dynamic maximum periodicity*, denoted as *dMaxPer*, measures the highest *periodicity* among all the patterns in EP_k. In other words, *dMaxPer* is calculated as the maximum value of the *periodicity* (*per*) among all the patterns in EP_k: $dMaxPer = \max(per(X_p) | forall X_p in EP_k)$.

Example 4: Given a set of all patterns in the database $AP = \{p, q, r, s, t, pq, pr, \cdots, pqrst\}$, and the set of explored patterns until now $EP = \{p, q, r, s, t\}$ which is a subset of AP. If $k = 5$, then EP_k represents the top-5 candidate periodic patterns discovered so far, which is $EP_k = p, q, r, s, t$. The *dMaxPer* is calculated as the maximum *periodicity* among the patterns in EP_k. Let's consider the pattern *rp* as an example. The *ts* at which *rp* occurs are $TS^{rp} = \{1, 3, 4, 6, 7, 8, 9\}$, and the $per(rp) = 2$. If we explore an additional pattern *rp* and update EP to $EP = \{p, q, r, s, t, rp\}$, we need to determine if *rp* should be included in the top-5 candidate periodic patterns. Since $per(rp) = 2$ and $dMaxPer = 4$, we compare the *periodicity* of *rp* with *dMaxPer*. As $per(rp)$ is less than *dMaxPer*, we prune the pattern *t* from *EP* since its *periodicity* is higher than that of *rp*. The updated EP_k becomes $\{p, q, r, s, pr\}$, and

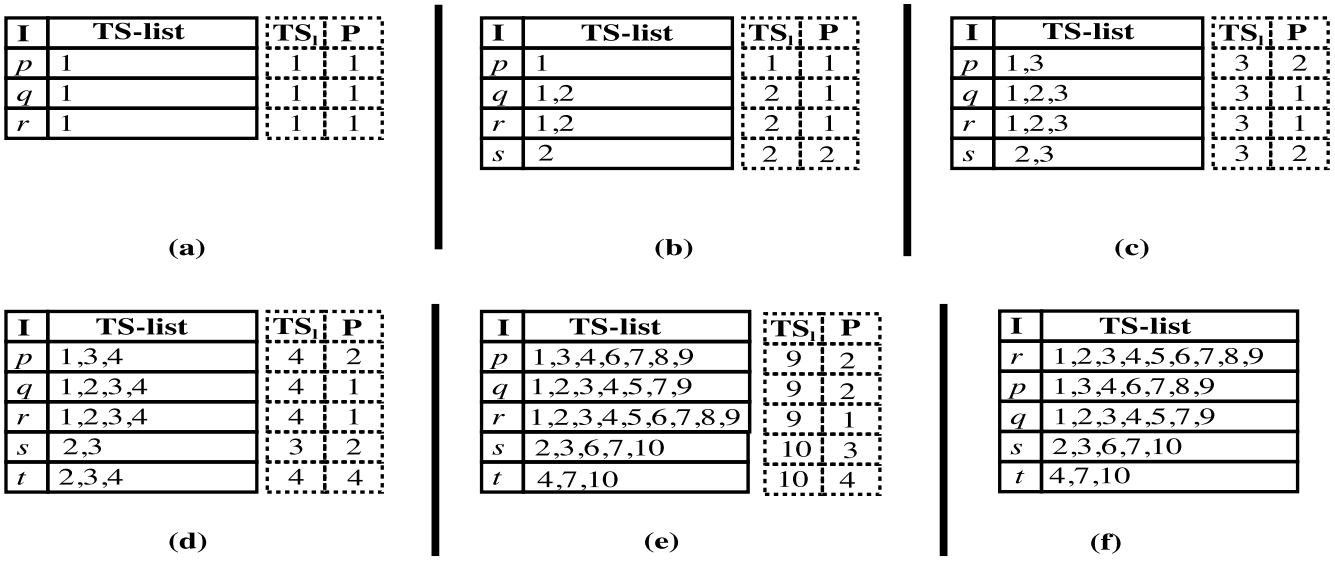

FIGURE 1. Finding periodic-frequent patterns. (a) After scanning the first transaction. (b) After scanning the second transaction. (c) After scanning the third transaction. (d) After scanning the fourth transaction. (e) After scanning all the transactions. (f) the final list of periodic-frequent items sorted in ascending order of their *periodicity*.

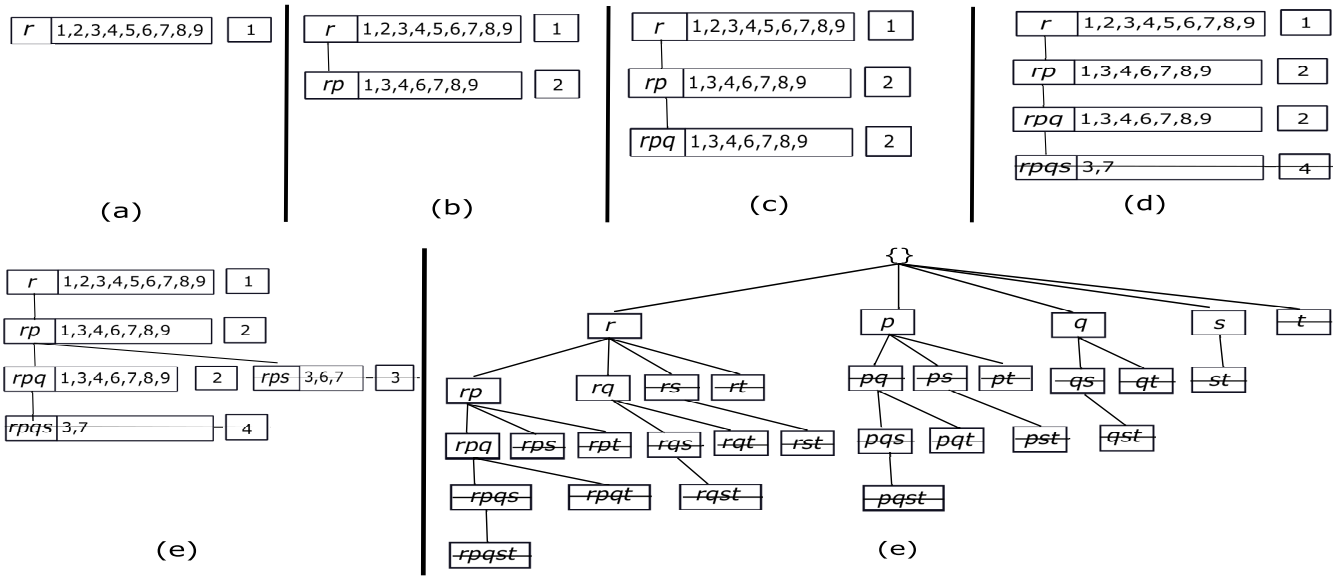

FIGURE 2. Mining Top-K periodic patterns using DFS.

the new *dMaxPer* is calculated as the *maximum periodicity* among the patterns in EP_k, which is $\max(2, 2, 1, 3, 2) = 3$. By updating *EP* and *dMaxPer*, we ensure that the set of top-*K* periodic-frequent patterns remains up-to-date with the highest *periodicities* observed so far.

The constraint mentioned above states that for a pattern to be considered a candidate top-*k* periodic pattern, its *periodicity* must be less than the current value of *dMaxPer*. This constraint serves as a criterion to determine the minimum occurrences required for a pattern to be considered a candidate top-*k* periodic-frequent pattern.

*Property 1 (**Pruning Technique**):* If the $per(X)$ is greater than the current value of *dMaxPer*, it implies that *X* and its supersets cannot be considered top-*k* periodic-frequent patterns. In other words, if the *periodicity* of *X* exceeds *dMaxPer*, the pattern occurs less frequently or exhibits a longer time interval between occurrences than the existing top-*k* periodic-frequent patterns.

The correctness of this property is based on Properties 2, 3, and Lemma 1. Our algorithm uses the above pruning technique to discover top-*k* periodic patterns effectively.

Property 2: For a pattern X, if $per(X) > dMaxPer$, then X cannot be a top-k periodic pattern.

Lemma 1: For a pattern X, if $per(X) > dMaxPer$, then X cannot be a top-k periodic pattern.

Proof: The correctness is straightforward to prove from Property 2. ■

Property 3: If $X \subset Y$, then $per(X) \leq per(Y)$ as $TS^X \supseteq TS^Y$.

V. K-PFPMINER

A. FINDING 1-LENGTH PERIODIC PATTERNS

The downward closure property of periodic-frequent patterns, as demonstrated in Property 1, is a fundamental characteristic that contributes to identifying top-k periodic patterns in a temporal database. This property asserts that if a pattern is deemed periodic-frequent, all of its subsets are also periodic-frequent. Consequently, 1-length periodic-frequent patterns hold significant importance in discovering top-k periodic patterns. Algorithm 1 presents a systematic approach that leverages the cPP-List to identify periodic-frequent patterns. To provide an understanding of the algorithm's functionality, we will now delineate its operation using the temporal database illustrated in Table 1, with the parameter K set to 7.

The database is scanned sequentially to generate 1-length periodic-frequent patterns. Consider the first transaction, ''1:pqr'', with the current timestamp $ts_{cur} = 1$. The items p, q, and r are inserted into the cPP-List. Their corresponding timestamps are set to 1, and the values of P and TS_l are also set to 1. This process is described in lines 5 and 6 of Algorithm 1. The cPP-List after scanning the first transaction is illustrated in Figure 1(a). Moving on to the second transaction, ''2:qrs'', with $ts_{cur} = 2$, the new item s is added to the cPP-List with its timestamp set to 2. Similarly, the timestamps of the existing items q and r are updated to 2. The values of P and TS_l are also adjusted accordingly. This process is carried out in lines 7 and 8 of Algorithm 1. The resulting cPP-List after scanning the second transaction is depicted in Figure 1(b). No new items are added in the third transaction, ''3:pqrs'', with $ts_{cur} = 3$. Therefore, the timestamps of the existing items are updated. This step is executed in lines 7 and 8 of Algorithm 1. The resulting cPP-List after scanning the third transaction is shown in Figure 1(c). Proceeding to the fourth transaction, ''4:pqrt'', with $ts_{cur} = 4$, the new item t is inserted into the cPP-List with its timestamp set to 4. Additionally, the timestamps of the existing items are updated accordingly. This process is performed in lines 7 and 8 of Algorithm 1. The resulting cPP-List after scanning the fourth transaction is displayed in Figure 1(d). A similar procedure is repeated for the remaining transactions in the database. The final cPP-List obtained after scanning the entire database is demonstrated in Figure 1(e). The cPP-List is then sorted in ascending order based on the periodicity of the patterns, as shown in Figure 1(f). The top k patterns are stored in the variable *topkPatterns*, and *dMaxPer* is calculated as the maximum periodicity within the cPP-List, as outlined in lines 13-15 of Algorithm 1.

Algorithm 1 PeriodicItems(Temporal Database (*TDB*), K (k))

1: Let *cPP-list* $= (X, TS\text{-}list(X))$ be a dictionary that records the temporal occurrence information of a pattern 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 P be a temporary list to record the *periodicity* of an item in the database. Let *topkPatterns* be a list to record the top items with lowest periodicity. Let *dMaxPer* be a variable to store the dynamic maximum period *dMaxPer* among *topkPatterns*.
2: **for** each transaction $t_{cur} \in TDB$ **do**
3: Set $ts_{cur} = t_{cur}.ts$;
4: **for** each item $i \in t_{cur}.X$ **do**
5: **if** i does not exit in cPP-List **then**
6: Insert i and its timestamp into the PFP-list. Set $TS_l[i] = ts_{cur}$ and $P[i] = (ts_{cur} - ts_{initial})$;
7: **else**
8: Add i's timestamp in the cPP-List. Update $TS_l[i] = ts_{cur}$ and $P[i] = max(P[i], (ts_{cur} - TS_l[i]))$;
9: **end if**
10: **end for**
11: **end for**
12: **for** each item i in cPP-List **do**
13: Calculate $P[i] = max(Per[i], (ts_{final} - TS_l[i]))$;
14: **end for**
15: Sort the remaining items in the cPP-List in ascending order of their *periodicity*.
16: **for** each item i in PFP-list **do**
17: **if** $length(topkPatterns) < K$: **then**
18: Store the item into *topkPatterns*
19: **end if**
20: **end for**
21: $dMaxPer = max$(periodicity of all items in *topkPatterns*)
22: Call k-PFPMiner(cPP-List).

1) FINDING TOP-K PERIODIC PATTERNS USING CPP-LIST

Algorithm 2 outlines the procedure for discovering top-k periodic patterns in a database. Now, we will describe the algorithm's functioning using the recently generated cPP-List.

We initiate the process by considering the first pattern in the cPP-List, which is pattern r (line 2 in Algorithm 2). The *periodicity* of r is recorded and displayed in Fig. 2(a). As r is a periodic-frequent pattern, we proceed to its child node rp and generate its TS-list by intersecting the TS-lists of r and p, denoted as $TS^{rp} = TS^r \cap TS^p$ (lines 3 and 4 in Algorithm 2). The periodicity of rp is recorded and displayed in Fig. 2(b). We then check if rp is a candidate periodic-frequent or uninteresting pattern (line 6 in Algorithm 2). Since rp is a

candidate periodic-frequent pattern, we proceed to check if its periodicity ($per(rp)$) is less than $dMaxPer$, as defined in Algorithm 3. We calculate $dMaxPer$ as the maximum periodicity among all the periodic-frequent patterns in the current set of *topkPatterns*. Since rp satisfies the condition of being a top-k periodic-frequent pattern, we proceed to its child node rpq and generate its TS-list by intersecting the TS-lists of rp and q, denoted as $TS^{rpq} = TS^{rp} \cap TS^{q}$. The periodicity of rpq is recorded and displayed in Fig. 2(c). We identify rpq as a periodic-frequent pattern and check if it qualifies as a top-k periodic-frequent pattern. Once again, we proceed to its child node $rpqs$ and generate its TS-list by intersecting the TS-lists of rpq and s, denoted as $TS^{rpqs} = TS^{rpq} \cap TS^{s}$. As the periodicity of $rpqs$ is greater than $dMaxPer$, we prune it from the candidate periodic patterns list, as shown in Fig. 2(d). We then move to the other child of rp and generate its TS-list by intersecting the TS-lists of rp and s, denoted as $TS^{rps} = TS^{rp} \cap TS^{s}$. As the periodicity of rps is greater than $dMaxPer$, we prune it from the candidate periodic-frequent patterns list, as shown in Fig. 2(e). This process is repeated for the remaining nodes in the set-enumeration tree to find all periodic-frequent patterns. The final list of periodic-frequent patterns generated from the temporal database in Table 1 is shown in Fig. 2(f). This approach of finding periodic-frequent patterns using the downward closure property is efficient as it effectively reduces the search space and computational cost.

Algorithm 2 k-PFPMiner(cPP-List)

```
for each item i in cPP-List do
    Set pi = ∅ and X = i;
    for each item j that comes after i in the cPP-List do
        Set Y = X ∪ j and TS^Y = TS^X ∩ TS^j;
        Calculate periodicity of Y;
        if per(TS^Y) ≤ dMaxPer then
            Add Y to pi and Y is considered as candidate top-
            k periodic-frequent itemset;
        end if
        Check(Y, TS^Y)
        (to check if pattern can make in to top-k periodic-
        frequent pattern)
    end for
    k-PFPMiner(pi)
end for
```

Algorithm 3 Check(X, TS-List)

```
if per(TS − List) < dMaxPer then
    Pop the Last pattern and insert X in topk − patterns.
end if
dMaxPer = max(periodicity of all items in topkPatterns)
```

B. TIME COMPLEXITY ANALYSIS

Suppose we are examining a database that stores temporal information. This database contains a total of a transactions, each corresponding to a specific time point. Across all of these transactions, c unique items have been recorded. 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 item count, and length of transactions, is crucial for performing the complexity analysis.

Due to its effective computation and identification of $k-$PFPs, the $k-$PFPMiner method makes significant contributions to the field of pattern mining. The Algorithm 1 begins by calculating the per and storing ts of each item by building the cPP-List. The current $dMaxPer$ is updated as the maximum periodicity of all the current top-k periodic-frequent patterns. The cPP-List is sorted in ascending order based on their per. The complexity of this initial Algorithm 1 is $\mathcal{O}(ab)$, where a is the number of transactions and b is the average transaction length.

Once the one-length $k-$PFPs have been identified, we generate combinations of items to form larger $k-$PFPs. This is accomplished using procedures outlined in Algorithm 2. Algorithm 3 follows a two-step procedure. The first step involves accessing two items and comparing their $(d-1)$-itemset timestamp lists to generate a d-itemset timestamp list with a complexity of $\mathcal{O}(c^2)$. The second step involves calculating the $dMaxPer$ of each itemset and discarding the uninteresting patterns based on the user-specified k parameter, which have a complexity of $\mathcal{O}(a)$. Overall, the complexity of this Algorithm 2 is $\mathcal{O}(c^2)$.

Finally, the entire complexity of finding all the $k-$PFPs using $k-$PFPMiner is $\mathcal{O}(c^2)$, which makes $k-$PFPMiner a highly efficient method for pattern mining.

VI. EXPERIMENTAL RESULTS

We conducted an evaluation of our algorithm, k-PFPMiner, for the task of discovering Top-k periodic-frequent patterns in temporal databases. As no existing algorithm specifically addresses this task with only the k constraint, our evaluation aimed to assess the effectiveness and performance of k-PFPMiner. We evaluated various databases, systematically varying the value of k to gain insights into the algorithm's behavior and capabilities.

A. EXPERIMENTAL SETUP

Our k-PFPMiner algorithm was implemented in Python 3.7 and executed on a high-performance Gigabyte R282-z94 rack server machine. This server machine has two AMD EPIC 7542 CPUs and 600 GB RAM, running the Ubuntu Server OS 20.04. The experiments have used synthetic (T10I4D100K, T20I6D10K) and real-world (Retail, BMS-WebView-2, Chess, Pollution, and Kosarak) databases.

The synthetic databases, **T10I4D100K** and **T20I6D100K**, were generated following the procedure described in [1]. The **Retail** database is a real-world database. The real-world database named **BMS-WebView-2** is sparse and contains click-stream data of an e-commerce website for several

TABLE 2. Statistics of the database.

Database	Type	Nature	Transaction Length			Database Size
			min.	avg.	max.	
T10I4D100K	Synthetic	Sparse	1	10	29	1,00,000
T20I6D100K	Synthetic	Sparse	2	19	46	1,99,844
Retail	Real	Sparse	2	12	77	88,162
BMS-WebView-2	Real	Sparse	2	5	161	77,512
Chess	Real	Dense	38	38	38	3192
Pollution	Real	Dense	11	460	971	720
Kosarak	sparse	Dense	2	9	2499	990,000

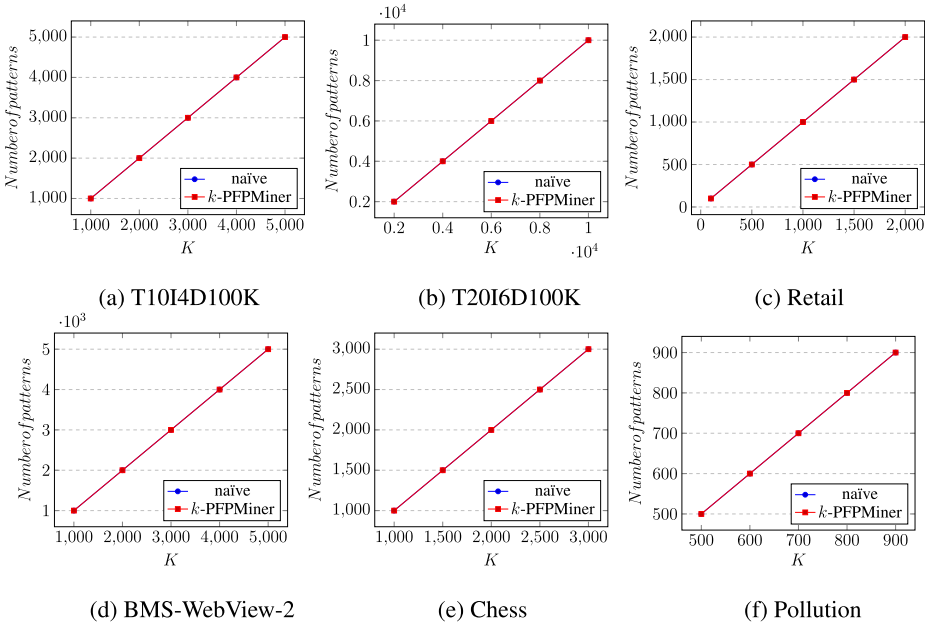

FIGURE 3. Top-*k* most patterns by varying *k* in different databases.

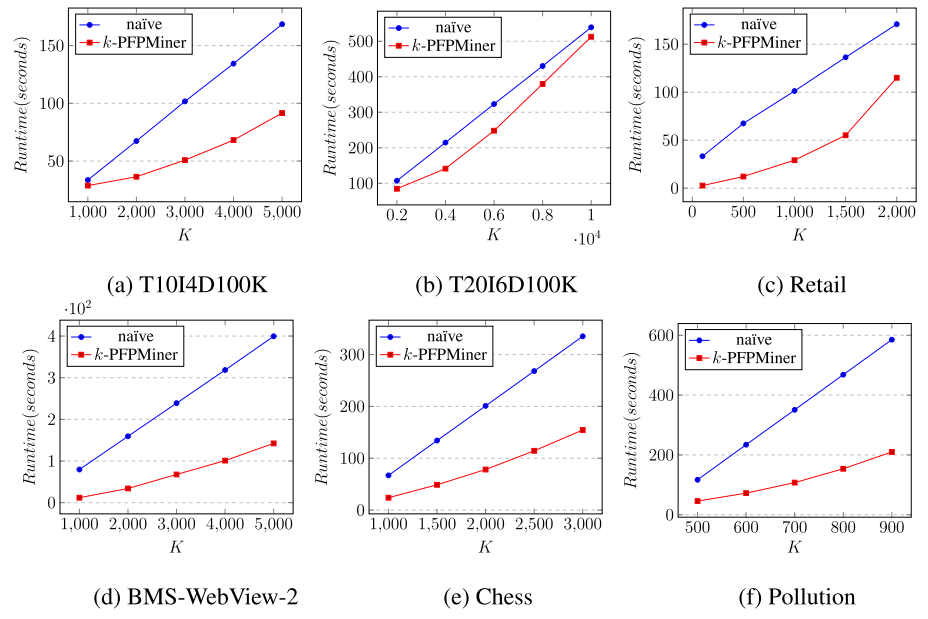

FIGURE 4. Runtime evaluation on various databases by varying *k*.

months. It was used for the KDD CUP 2000 competition. The **Chess** database is a high-dimensional real-world database containing 75 items and 3196 transactions prepared from the UCI chess database. The **Kosarak** database is an extensive

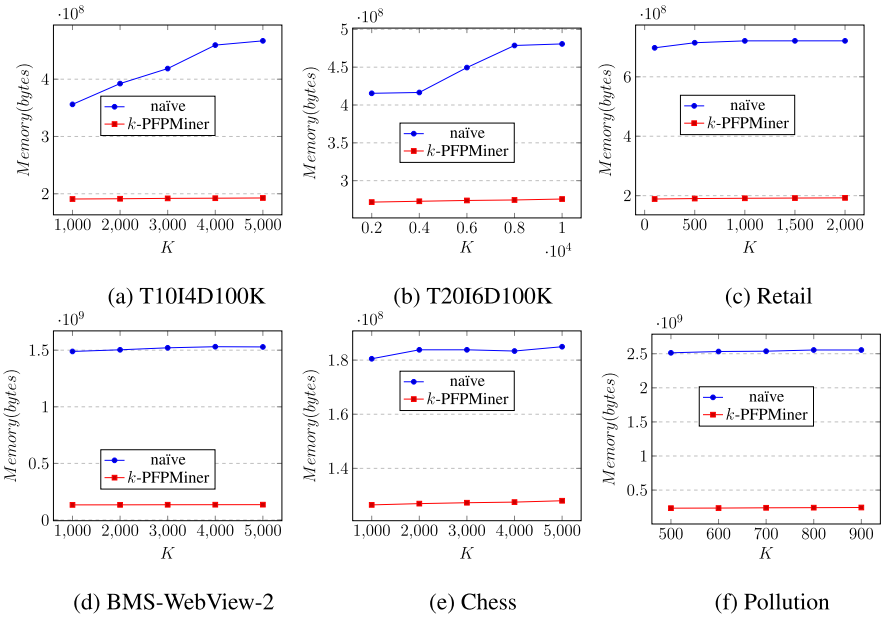

FIGURE 5. Memory evaluation on various databases by varying *k*.

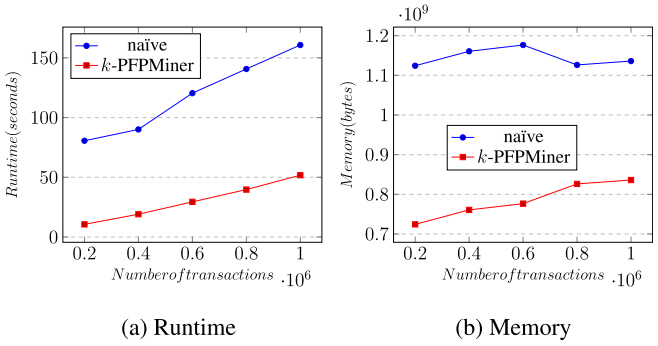

FIGURE 6. Scalability of *k*-PFPMiner.

clickstream data collection of 990,000 sequences from a Hungarian news portal.

The **Pollution** database, on the other hand, is a high-dimensional real-world database provided by the Japanese Ministry of the Environment through the Atmospheric Environmental Regional Observation System (AEROS) [38]. It aims to address air pollution issues. Each transaction in this database contains information such as the timestamp in hours and the station identifiers that have recorded $PM_{2.5}$ values equal to or greater than 16 $\mu g/m^3$. The Pollution database comprises 1,600 unique items and 720 transactions. The transaction lengths also vary in this database, with the minimum, average, and maximum lengths being 11, 460, and 971 items, respectively.

These databases were chosen to evaluate our *k*-PFPMiner algorithm's performance and effectiveness in different scenarios and assess its scalability and applicability in both synthetic and real-world contexts. We provide statistical details for each database used in our experiments in Table 2. The algorithms are available in Github [39] to verify the repeatability of our experiments.

B. INFLUENCE OF THE K-PARAMETER ON THE PERFORMANCE OF THE K-PFPMINER

In the first experiment, we varied the parameter *k* to assess its impact on the performance of *k*-PFPMiner in terms of runtime and memory usage. The experiments were conducted on the six databases: T10I4D100K, T20I6D100K, Retail, BMS-WebView-2, Chess, and Pollution 3.

Fig. 3 illustrates the number of PFPs generated by the naïve and *k*-PFPMiner algorithms over different databases at varying *k* values. The Y-axis represents the number of PFPs generated by each algorithm, while the X-axis denotes the specific *k* value at which the patterns were generated. Based on this figure, the following observations can be made:

- Firstly, the naïve and *k*-PFPMiner algorithms generate an equal number of PFPs for each database. As a result, both curves overlap one another.
- Secondly, an increase in the *k* threshold increases the number of 3Ps generated. This can be attributed to the fact that a higher *k* value results in more patterns that can be extracted from the complete set of PFPs.

The runtime results of the experiments are presented in Figure 4, which provides an overview of the execution time for different values of k. This graph allows us to analyze the relationship between k and the runtime required to mine the top-k periodic-frequent patterns.

In Fig. 4, the x-axis represents the values of the parameter k, while the y-axis represents the execution time of the k-PFPMiner algorithm. Based on this figure, the following observations can be made:

- Firstly, increasing the value of k tends to increase the runtime of the k-PFPMiner algorithm. This is a reasonable observation since as k increases, more patterns are discovered, and a larger number of itemsets from the search space need to be considered to identify the top-k periodic-frequent patterns. Consequently, k-PFPMiner may need to evaluate more patterns to populate the candidate set cPP-List, resulting in increased computational time.
- Secondly, we observe that the k-PFPMiner algorithm requires considerably less runtime when compared to the naïve algorithm against all the databases (either sparse or dense nature), regardless of the k value.

The memory results of the experiments are presented in Figure 5, which provides an overview of the memory consumption for different values of k. This graph allows us to analyze the relationship between k and the memory required to mine the top-k periodic-frequent patterns.

In Fig. 3, the x-axis represents the values of the parameter k, while the y-axis represents the memory consumption of the k-PFPMiner algorithm. Based on this figure, the following observations can be made:

- As the value of k increases in the k-PFPMiner algorithm, the memory consumption also tends to increase. This is primarily because when k is set to larger values, the algorithm needs to consider more itemsets to populate the candidate set Q_k. The process of evaluating and storing these additional itemsets requires more memory resources.
- k-PFPMiner algorithm requires significantly less memory than the naïve algorithm across all evaluated databases, regardless of their density or sparsity and the k value. Furthermore, this difference in memory usage was exceptionally high at high k values.

C. INFLUENCE OF THE NUMBER OF TRANSACTIONS ON K-PFPMINER PERFORMANCE

To assess the scalability of the proposed algorithm, we experimented to measure its execution time and peak memory usage as the number of transactions in the database was varied. The real-world database *kosarak* was selected for this experiment due to its many distinct items and transactions. The database was partitioned into five parts, and the algorithm's performance was evaluated after adding each part to the previously processed data. Figure 6a and 6b respectively show the runtime and memory requirements of k-PFPMiner algorithm at different database sizes when $k = 200$. It is evident that both the execution time and memory usage increase as the size of the database grows. This observation is reasonable as larger databases tend to have more itemsets, requiring additional processing time for their evaluation.

VII. CONCLUSION AND FUTURE RESEARCH

Traditional algorithms for mining periodic-frequent patterns often rely on manually setting thresholds for *min_sup* and *max_per*, which can be challenging and may not yield optimal results. This paper addresses these limitations by proposing a novel model for discovering top-k periodic-frequent patterns in temporal databases. These patterns represent the k most frequent patterns with the lowest periodicity values across the entire database. Our model introduces a new upper-bound measure called *dynamic maximum periodicity* to reduce the search space effectively. Importantly, this upper-bound value is automatically calculated and updated without human intervention, providing a more efficient and adaptive approach. Furthermore, we have developed a novel pruning technique that significantly reduces the time complexity of identifying whether a pattern is periodic or aperiodic. In the best case, this complexity has been reduced to O(1); in the worst case, it is O(n). To efficiently discover all desired patterns in the database, we propose an efficient single-pass algorithm called top-k Periodic-Frequent Pattern Miner (k-PFPMiner). This algorithm utilizes a best-first search strategy, enabling it to effectively navigate the search space and find the top-k periodic-frequent patterns in an optimized manner. Extensive experimental evaluations have been conducted using synthetic and real-world databases to assess the performance of k-PFPMiner. The results demonstrate that our algorithm is highly efficient and can discover valuable patterns in temporal data. As for future work, we will work on discovering top-k periodic-frequent patterns in uncertain databases.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 1993, pp. 207–216.

[2] L. Shen, H. Shen, P. Pritchard, and R. Topor, *Finding The N Largest Itemsets*. Southampton, U.K.: WIT Press, 1998.

[3] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory—ICDT*, vol. 1540. Berlin, Germany: Springer, Aug. 2000.

[4] G. Yang, "The complexity of mining maximal frequent itemsets and maximal frequent patterns," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Assoc. Comput. Machinery, Aug. 2004, pp. 344–353, doi: 10.1145/1014052.1014091.

[5] J. C. Lin, T. Li, P. Fournier-Viger, T.-P. Hong, and J.-H. Su, "Fast algorithms for mining multiple fuzzy frequent itemsets," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2016, pp. 2113–2119.

[6] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2004, pp. 482–486.

[7] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Advances in Database Technology—EDBT*, P. Apers, M. Bouzeghoub, and G. Gardarin, Eds. Berlin, Germany: Springer, 1996, pp. 1–17.

[8] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal, "Mining frequent patterns with counting inference," *ACM SIGKDD Explor. Newslett.*, vol. 2, no. 2, pp. 66–75, Dec. 2000, doi: 10.1145/380995.381017.

[9] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *Advances in Knowledge Discovery and Data Mining*. Berlin, Germany: Springer, 2009, pp. 242–253.

[10] 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.

[11] P. Fournier-Viger, Z. Li, J. C.-W. Lin, R. U. Kiran, and H. Fujita, "Efficient algorithms to identify periodic patterns in multiple sequences," *Inf. Sci.*, vol. 489, pp. 205–226, Jul. 2019.

[12] 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. (FUZZ-IEEE)*, Jul. 2020, pp. 1–8.

[13] R. U. Kiran, Y. Watanobe, B. Chaudhury, K. Zettsu, M. Toyoda, and M. Kitsuregawa, "Discovering maximal periodic-frequent patterns in very large temporal databases," in *Proc. IEEE 7th Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2020, pp. 11–20.

[14] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering recurring patterns in time series," in *Proc. 18th Int. Conf. Extending Database Technol.*, 2015, pp. 97–108.

[15] 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. 9th Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2022, pp. 1–10.

[16] P. Veena, P. Ravikumar, K. Kwangwari, R. U. Kiran, K. Goda, Y. Watanobe, and K. Zettsu, "Discovering fuzzy geo-referenced periodic-frequent patterns in geo-referenced time series databases," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2022, pp. 1–8.

[17] P. Fournier-Viger, P. Yang, J. C.-W. Lin, and R. U. Kiran, "Discovering stable periodic-frequent patterns in transactional data," in *Advances and Trends in Artificial Intelligence. From Theory to Practice*, F. Wotawa, G. Friedrich, I. Pill, R. Koitz-Hristov, and M. Ali, Eds. Cham, Switzerland: Springer, 2019, pp. 230–244.

[18] H. N. Dao, P. Ravikumar, P. Likitha, B. V. V. Raj, R. U. Kiran, Y. Watanobe, and I. Paik, "Towards efficient discovery of stable periodic patterns in big columnar temporal databases," in *Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence*, H. Fujita, P. Fournier-Viger, M. Ali, and Y. Wang, Eds. Cham, Switzerland: Springer, 2022, pp. 831–843.

[19] 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.

[20] P. Likhitha, P. Ravikumar, R. U. Kiran, and Y. Watanobe, "Discovering top-*k* periodic-frequent patterns in very large temporal databases," in *Big Data Analytics*, P. P. Roy, A. Agarwal, T. Li, P. K. Reddy, and R. U. Kiran, Eds. Cham, Switzerland: Springer, 2022, pp. 200–210.

[21] M. S. Nawaz, P. Fournier-Viger, A. Shojaee, and H. Fujita, "Using artificial intelligence techniques for COVID-19 genome analysis," *Appl. Intell.*, vol. 51, no. 5, pp. 3086–3103, May 2021, doi: 10.1007/s10489-021-02193-w.

[22] L. Ni, W. Luo, N. Lu, and W. Zhu, "Mining the local dependency itemset in a products network," *ACM Trans. Manage. Inf. Syst.*, vol. 11, no. 1, pp. 1–31, Apr. 2020, doi: 10.1145/3384473.

[23] D. Schweizer, M. Zehnder, H. Wache, H.-F. Witschel, D. Zanatta, and M. Rodriguez, "Using consumer behavior data to reduce energy consumption in smart homes: Applying machine learning to save energy without lowering comfort of inhabitants," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2015, pp. 1123–1129.

[24] M. S. Nawaz, P. Fournier-Viger, M. Z. Nawaz, G. Chen, and Y. Wu, "MalSPM: Metamorphic malware behavior analysis and classification using sequential pattern mining," *Comput. Secur.*, vol. 118, Jul. 2022, Art. no. 102741. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822001365

[25] P. Fournier-Viger, T. Gueniche, and V. S. Tseng, "Using partially-ordered sequential rules to generate more accurate sequence prediction," in *Advanced Data Mining and Applications*, S. Zhou, S. Zhang, and G. Karypis, Eds. Berlin, Germany: Springer, 2012, pp. 431–442.

[26] M. S. Nawaz, M. Sun, and P. Fournier-Viger, "Proof guidance in PVS with sequential pattern mining," in *Fundamentals of Software Engineering*, H. Hojjat and M. Massink, Eds. Cham, Switzerland: Springer, 2019, pp. 45–60.

[27] 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, Nov. 2019, Art. no. e1329.

[28] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, May 2000, doi: 10.1145/335191.335372.

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

[30] R. U. Kiran and P. K. Reddy, "An alternative interestingness measure for mining periodic-frequent patterns," in *Database Systems for Advanced Applications*. Berlin, Germany: Springer, 2011, pp. 183–192.

[31] 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. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121215002307

[32] R. U. Kiran and M. Kitsuregawa, "Novel techniques to reduce search space in periodic-frequent pattern mining," in *Database Systems for Advanced Applications*. Berlin, Germany: Springer, 2014, pp. 377–391.

[33] 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, doi: 10.1109/SSCI.2016.7849926.

[34] A. Surana, R. U. Kiran, and P. K. Reddy, "An efficient approach to mine periodic-frequent patterns in transactional databases," in *Proc. PAKDD Workshops*, 2011, pp. 254–266.

[35] 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

[36] R. Penugonda, L. Palla, U. K. Rage, Y. Watanobe, and K. Zettsu, "Towards efficient discovery of periodic-frequent patterns in columnar temporal databases," in *Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices*, H. Fujita, A. Selamat, J. C.-W. Lin, and M. Ali, Eds. Cham, Switzerland: Springer, 2021, pp. 28–40.

[37] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-*k* periodic-frequent pattern from transactional databases without support threshold," in *Advances in Information Technology*. Berlin, Germany: Springer, 2009, pp. 18–29.

[38] Journal Ministry of Environment. (2021). *Atmospheric Environmental Regional Observation System*. Accessed: Jun. 1, 2021. [Online]. Available: http://soramame.taiki.go.jp/

[39] R. U. Kiran. (2023). *PAMI-PyKit: PAttern MIning-Python Kit*. Accessed: Jun. 4, 2023. [Online]. Available: https://github.com/UdayLab/codeData/tree/main

PALLA LIKHITHA (Member, IEEE) is currently pursuing the master's degree in computer science and engineering with The University of Aizu, Japan. She has published papers in IEEE Big Data, ICONIP, IEA/AIE, SIGSPATIAL IEEE DSAA, PAKDD, and IEEE-FUZZ.

PENUGONDA RAVIKUMAR received the M.E. degree in computer science from the Indian Institute of Science, Bengaluru, Karnataka, India. He is currently pursuing the Ph.D. degree in computer and information systems with The University of Aizu, Aizuwakamatsu, Fukushima, Japan, on a deputation basis. He holds the position of an Assistant Professor with the Department of Computer Science and Engineering, IIIT–RK Valley, Rajiv Gandhi University of Knowledge Technologies, Andhra Pradesh, India. He has authored numerous journal articles published in esteemed publications, such as *Applied Intelligence* and *Electronics*, and presented his work at renowned international conferences, including IEEE FUZZ, IEEE Big Data, IEEE DSAA, IEEE SSCI-CIDM, IEA/AIE, ISCMI, DEXA, and ACIIDS. His current research interests include data mining, air pollution data analytics, traffic congestion data analytics, recommender systems, and time series classification.

DEEPIKA SAXENA (Member, IEEE) received the Ph.D. degree in computer science from the National Institute of Technology, Kurukshetra, India, and the Ph.D. degree from the Department of Computer Science, Goethe University, Frankfurt, Germany. She is currently an Associate Professor with the Division of Information Systems, The University of Aizu, Japan. Her research interests include neural networks, evolutionary algorithms, scheduling, security in cloud computing, internet traffic management, resource management, quantum machine learning, DataLakes, and dynamic caching management. Some of her research findings are published in top-cited journals, such as IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE SYSTEMS JOURNAL, IEEE COMMUNICATIONS LETTERS, IEEE NETWORKING LETTERS, *Neurocomputing*, and *IET Electronics Letters*.

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 is also a Researcher with The University of Tokyo, Tokyo, Japan. He has published over 80 papers in refereed journals and international conferences, such as EDBT, CIKM, IEEE FUZZ, PAKDD, SIGSPATIAL, IEEE Big Data, DEXA, and DASFAA. His research interests include data mining, parallel computation, air pollution data analytics, traffic congestion data analytics, recommender systems, and ICTs for agriculture.

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.