Discovering Top-k Periodic-Frequent Patterns in Very Large Temporal Databases

Palla Likhitha(✉), Penugonda Ravikumar, Rage Uday Kiran, and Yutaka Watanobe

University of Aizu, Aizuwakamatsu, Japan
likhithapalla7@gmail.com, raviua138@gmail.com, {udayrage,yutaka}@u-aizu.ac.jp

Abstract. Discovering periodic-frequent patterns in temporal databases is a challenging data mining problem with abundant applications. It involves discovering all patterns in a database that satisfy the user-specified *minimum support* ($minSup$) and *maximum periodicity* ($maxPer$) constraints. $MinSup$ controls the minimum number of transactions in which a pattern must appear in a database. $MaxPer$ controls the maximum time interval within which a pattern must reappear in the database. Setting an appropriate $minSup$ and $maxPer$ values for any given database is an open research problem. 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 a total of k periodic-frequent patterns with the lowest *periodicity* value in a database. An efficient depth-first search algorithm, called 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 memory and runtime efficient and highly scalable.

Keywords: Data mining · Pattern mining · Temporal databases

1 Introduction

Frequent pattern mining [1] is a popular data mining model aiming to discover all frequently occurring patterns in a transactional database. However, a fundamental limitation of this model is that it fails to discover temporal regularities that may exist in a temporal database. When confronted with this problem in real-world applications, researchers proposed an extended model to discover periodic-frequent patterns [12] in a temporal database that satisfy the user-specified *minimum support* ($minSup$) and *maximum periodicity* ($maxPer$) constraints.

Several pattern mining techniques, such as fuzzy periodic-frequent pattern mining [6], local periodic pattern mining [4], partial periodic pattern mining

[9], stable periodic pattern mining [5], recurring pattern mining [8], and periodic sequential pattern mining, were inspired from the periodic-frequent pattern model. However, this model's widespread adoption and successful industrial application was hindered by this obstacle: "*minSup and maxPer are two key constraints that make periodic-frequent pattern mining practicable in real-world applications. They are used to prune the search space and limit the number of patterns generated. Unfortunately, setting these two constraints for an application is an open research problem and may require a profound knowledge of the application background.*" This paper addresses this challenging open problem by finding top-k periodically occurring frequent patterns in a temporal database. This paper's contribution is as follows: (i) we proposed a novel model of top-k periodic-frequent patterns that may exist in temporal databases. Only one constraint k is used to find the interesting top-k periodic-frequent patterns with the lowest *periodicity* in the entire database. (ii) We also introduced an upper-bound measure and a pruning technique called *dynamic maximum periodicity* to reduce the search space and in pruning the uninteresting patterns. (iii) We proposed an efficient search algorithm called top-k Periodic-frequent Pattern Miner (k-PFPMiner) to find all the desired patterns. (v) Experimental results on synthetic and real-world databases demonstrate that our algorithm is memory and runtime efficient and highly scalable.

The rest of the paper is organized as follows. Section 2 describes related work on finding top-k periodic-frequent patterns in databases. Section 3 describes a proposed model to find top-k periodic-frequent patterns in databases. Section 4 describes the proposed algorithm to discover the top-k periodic-frequent patterns. Section 5 presents the experimental results obtained. Finally, in Sect. 6, we conclude and discuss future research.

2 Related Work

Agrawal et al. [1] introduced the concept of frequent pattern mining to extract useful information from transactional databases. Luna et al. [10] conducted a detailed survey on frequent pattern mining and presented the improvements in the past 25 years. However, finding patterns that appear regularly in a database with the help of frequent pattern mining, which only considers frequency, is not appropriate.

Tanbeer et al. [12] generalized the frequent pattern model to discover periodic-frequent patterns in a temporal database. Amphawan et al. [2] introduced a model to find the most periodic-frequent patterns called Top-k periodic-frequent patterns. Uday et al. [7] have designed a novel concept named *local periodicity* to prune the non-periodic patterns locally. Authors have discarded the patterns whose *local periodicity* is less than the user-specified *maxPer* value. As a result, most of the non-periodic patterns tid-lists were not completely built, resulting in a decrease in the computational time of the proposed algorithm. Since its inception, the problem of finding periodic-frequent patterns has received a great deal of attention [3,11]. The basic model used in most of these algorithms remains the same. Discovering complete set of patterns in

Table 1. Temporal database

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

temporal databases that satisfy the user-specified $minSup$ and $maxPer$ values. Setting a user-specified $minSup$ and $maxPer$ for a given database is an open research problem. When confronted with this problem in real-world applications, researchers have tried to find top-k periodic-frequent patterns. Amphawan et al. [2] described the top-k periodic-frequent patterns from transactional databases without $minSup$ constraint. However, the authors have used the user-specified $maxPer$ constraint and a k value to generate top-k periodic-frequent patterns.

In this paper, we are updating the $maxPer$ value dynamically without user intervention while generating exciting patterns. By setting the user-defined k value only, we are extracting the top-k periodic-frequent patterns from the large temporal databases.

3 Proposed Model: top-*k* Periodic-Frequent Patterns

Let O be the set of objects (or items). Let $P \subseteq O$ be a **itemset** (or a pattern). A itemset containing α, $\alpha \geq 1$, number of items is called a **α-pattern**. In a **transaction**, $t_k = (ts,\ X)$ is a tuple, where ts represents the timestamp at which the pattern X has occurred. A **temporal database** TDB over O is a set of transactions, i.e., $TDB = \{tr_1, \cdots, tr_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. For a transaction $tr_k = (ts,\ X)$, $k \geq 1$, such that $Z \subseteq X$, it is said that Z occurs in tr_k (or tr_k contains Z) and such a timestamp is denoted as ts^Z. Let $TS^Z = \{ts_j^Z, \cdots, ts_k^Z\}$, $j,\ k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where Z has occurred in TDB.

Example 1. Let $O = \{p, q, r, s, t\}$ be the set of items. A temporal database generated from items O is shown in Table 1. The set of items 'r' and 'p', i.e., $\{r,\ p\}$ is considered as a pattern. For short, we represent this pattern as 'rp .' This pattern is a 2-pattern because it contains two items. The pattern 'rp' appears at the timestamps of 1, 3, 4, 6, 7, 8, and 9. Therefore, the list of timestamps containing 'rp', i.e., $TS^{rp} = \{1,\ 3,\ 4,\ 6,\ 7,\ 8,\ 9\}$.

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 $P^Z = \{p_1^Z, p_2^Z, \cdots, p_k^Z\}, k = |TS^Z| + 1$, be the set of all periods of Z in $UTDB$. The periodicity of Z, denoted as $per(Z) = max(p_1^Z, p_2^Z, \cdots, p_k^Z)$.

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

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 said to be a top-k periodic pattern if its periodicity is no more than the periodicity of pattern X_k in the database. That is, X_a is said to be a top-k periodic pattern if $per(X_a) \leq per(X_k)$.*

Example 3. If the $per(rp) \leq per(t)$, the pattern *rp* is considered as top-*k* periodic frequent patterns.

Definition 3. *(**Problem definition.**) Given a temporal database (TDB) and a user-specified k value, the goal of top-k periodic pattern mining is to discover only top-k periodic-frequent patterns that have the lowest periodicities in a database.*

4 Our Algorithm

4.1 Basic Idea: Dynamic Maximum Periodicity

Reducing the enormous search space is challenging as our model does not employ any constraint to reduce the search space. In this context, our idea to reduce this huge search space is as follows: "*Create an empty list known as* candidate periodic pattern-list *(or cpp-List). Also, create a Max-heap data structure and set its root to null. Then, scan the database and keep adding the patterns to the cpp-List. Simultaneously, update the Max-heap with the periodicity values of those items. Once the size of cpp-List reaches the size of k, set* dynamic maximum periodicity *(dMaxPer) equal to the value of root in Max-heap. Prune the search space (or itemsets) using the $dMaxPer$ constraints. If we find any pattern in the constraint, add the corresponding pattern into the cpp-List by removing the existing k-pattern. We will update $dMaxPer$ accordingly. We keep repeating this process until we complete the search space.*" The time complexity to determine *periodicity* of a pattern are $O(1)$ and $O(n)$, respectively. Where n represents the number of timestamps (or $frequency$) of a pattern in the database.

Definition 4. *(**Dynamic maximum periodicity constraint.**) Let $AP = \{X_1, X_2, \cdots, X_{2^n - 1}\}$, $n \geq 1$, be the set of all patterns in a database. Let $EP \subseteq SP$ be the set of patterns explored by our algorithm until now. Let $EP_k \subseteq EP$ such that $|EP_k| = k$ be a set of top-k candidate periodic patterns* *found until now. The* dynamic maximum periodicity, *denoted as* $dMaxPer$, *represents the highest periodicity among all patterns in* EP_k. *That is,* $dMaxPer = \{max(per(X_p)|\forall X_p \in EP_k)\}$.

Example 4. Let $AP = \{p, q, r, s, t, pq, pr, \cdots, pqrst\}$ be the set of all patterns in a database. Let $EP = \{p, q, r, s, t\} \subset AP$ be the set of patterns explored until now. If $k = 5$, then $EP_k = \{p, q, r, s, t\}$. Thus, $dMaxPer = max(per(p), per(q), per(r), per(s), per(t)) = max(2, 2, 1, 3, 4) = 4$. For the pattern pr, $TS^{pr} = \{1, 3, 4, 6, 7, 8, 9\}$ and $per(pr) = 2$. If we explore a different pattern pr, then $EP = \{p, q, r, s, t, pr\}$. As $per(pr) < dMaxPer$, we prune pattern t and add pr in the EP_k. Thus, $EP_k = \{p, q, r, s, pr\}$ and $dMaxPer = max(2, 2, 1, 3, 2) = 3$. Thus, $dMaxPer$ automatically gets updated whenever a candidate top-k periodic-frequent pattern is found in the database.

The above constraint says that a pattern must have *periodicity* less than the $dMaxPer$ to be a candidate top-k periodic pattern. Thus, this constraint can be used to determine the minimal occurrences a pattern must have to be a candidate top-k periodic-frequent pattern.

Property 1. (**Pruning technique:**) Prune the pattern X if $per(X) > dMaxPer$. It is because neither X nor its supersets can be 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 straight forward to prove from Property 2.

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

4.2 *k*-PFPMiner

k-PFPMiner starts with finding top-k single items in the database and stores them in cPP-List described in Algorithm 1. Next, Algorithm 2 describes the procedure for finding top-k periodic patterns in a depth-first search manner. We now describe the working of this algorithm using the newly generated cPP-list.

We start with item r, the pattern in the cPP-list with the lowest periodicity (line 2 in Algorithm 2). We preserve the *periodicity* of r, as shown in Fig. 1(a). Since r is a periodic-frequent pattern, we proceed to its child node rp by combining with other periodic items in the database and generate its TS-list by intersecting TS-lists of both items r and p, i.e., $TS^{rp} = TS^r \cap TS^p$ (lines 3 and

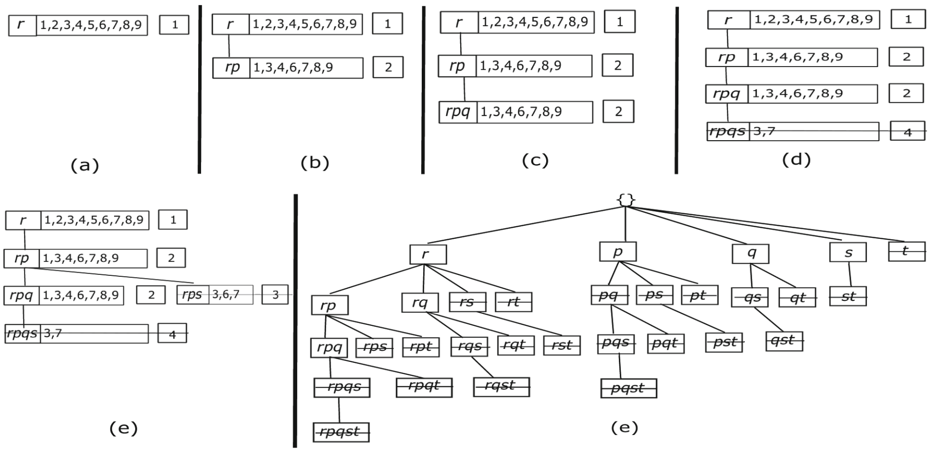


Fig. 1. Mining Top-K periodic patterns using DFS

4 in Algorithm 2). We record *periodicity* of *rp*, as shown in Fig. 1(b). We verify whether *rp* is a candidate periodic-frequent pattern or uninteresting pattern (line 6 in Algorithm 2). Since *rp* is a candidate periodic-frequent pattern, we check if *periodicity* of *rp* if $per(pr) < dMaxPer$, (in Algorithm 3) and calculate $dMaxPer$ as a maximum of all the *periodicities* of periodic-frequent patterns in current existing *topkPatterns*. Since *rp* is a top-k periodic-frequent pattern, we proceed to its child node *rpq* and generate its TS-list by performing the intersection of TS-lists of *rp* and *q*, i.e., $TS^{rpq} = TS^{rp} \cap TS^{q}$. We record *periodicity* of *rpq*, as shown in Fig. 1(c), identify it as a periodic-frequent pattern and check if it can be a top-k periodic frequent pattern. Since *rpq* is a top-k periodic frequent pattern, we again move to its child node *rpqs* and generate its TS-list by performing the intersection of TS-lists of *rpq* and *s*, i.e., $TS^{rpqs} = TS^{rpq} \cap TS^{s}$. As *periodicity* of *rpqs* is greater than the $dMaxPer$, we will prune the pattern *rpqs* from the candidate periodic patterns list as shown in Fig. 1(d). We move to the other child of *rp* and generate its TS-list by performing the intersection of TS-lists *rp* and *s*, i.e., $TS^{rps} = TS^{rp} \cap TS^{s}$. As the *periodicity* of *rps* is greater than the $dMaxPer$, we will prune the pattern *rps* from the candidate periodic-frequent patterns list as shown in Fig. 1(e). The exact process is done for all the remaining nodes in the tree to find all periodic-frequent patterns. The complete set of periodic-frequent patterns generated from Table 1 is shown in Fig. 1(f) without striking. The above approach of finding periodic-frequent patterns using the downward closure property is efficient because it effectively reduces the search space and the computational cost.

5 Experimental Results

Since there exists no algorithm to find Top-k periodic-frequent patterns in temporal databases with only k constraint, we evaluated our algorithm k-PFPMiner on different databases varying k.

Algorithm 1. PeriodicItems(Temporal Database (TDB), K (k):

```
1: Let's say that the T-PFPList=($Y, TS\text{-}list(Y)$) is a dictionary that keeps track of
   temporal information about a pattern that occurs in a $TDB$. First, let's create
   a temporary list called $TS_l$ and use it to keep track of the timestamp of the last
   time an item appeared in the database. Let $Per$ 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 every transaction $t_{cur} \in TDB$ do
3:    Set $ts_{cur} = t_{cur}.ts$;
4:    for every 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: for each item $i$ in cPP-list do
10:   Calculate $P[i] = max(Per[i], (ts_{final} - TS_l[i]))$;
11: Sort the remaining items in the cPP-list in ascending order of their periodicity.
12: for each item $i$ in PFP-list do
13:    if $length(topkPatterns) < K$: then
14:       Store the item into $topkPatterns$
15: $dMaxPer = max$(periodicity of all items in $topkPatterns$)
16: Call $k$-PFPMiner(cPP-List).
```

Algorithm 2. k-PFPMiner(cPP-List)

```
1: for each item $i$ in cPP-List do
2:    Set $tp = \emptyset$ and $X = i$;
3:    for each item $j$ that comes after $i$ in the cPP-list do
4:       Set $Y = X \cup j$ and $TS^Y = TS^X \cap TS^j$;
5:       Calculate periodicity of $Y$;
6:       if $per(TS^Y) \leq dMaxPer$ then
7:          Add $Y$ to $tp$ and $Y$ is considered as candidate top-k periodic-frequent item-
            set;
8:       Check($Y, TS^Y$)
         (to check if pattern can make in to top-$k$ periodic-frequent pattern)
9:    $k$-$PFPMiner(tp)$
```

Algorithm 3. Check(X, TS-List)

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

Table 2. The complete statistical information of the databases used in our experiments

S.No	Database	Type	Transaction Length (in count)			Items (in count)	Database Size (in count)
			min.	avg.	max.		
1	T10I4D100K	Synthetic	1	10	29	870	1,00,000
2	Retail	Real	2	12	77	16,471	88,162
3	Congestion	Real	1	58	337	1,414	8, 928
4	Pollution	Real	11	460	971	1,600	720
5	Kosarak	Real	2	9	2,499	41,270	99,00,00

5.1 Experimental Setup

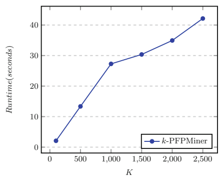
Our k-PFPMiner algorithm was developed in Python 3.7 and executed on a Gigabyte R282-z94 rack server machine containing two AMD EPIC 7542 CPUs and 600 GB RAM. The operating system of this machine is Ubuntu Server OS 20.04. The experiments have been conducted on both synthetic (**T25I10D10K**) and (**Retail**) and real-world (**Congestion**, **Pollution**, and **Kosarak**) databases. The T10I10D100K database is a synthetic database generated using the procedure described in [3]. Table 2 presents the statistical information that may be found in the databases mentioned above. Pollution and Congestion may be high-dimensional databases that include extensive transactions.

In the field of intelligent transportation systems, one challenging but important task is monitoring traffic congestion in smart cities. In this regard, the JARTIC situated in Kobe, Japan has established many sensor networks to monitor congestion in several smart cities. *Each transaction in this database takes place every 5 min and includes a timestamp and the identifiers of road segments that have reported traffic jams of more than 300 m.* The time period covered by the data collection is from July 1st to July 31st, 2015.

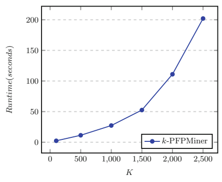
The Japanese Ministry of the Environment developed the AEROS to tackle air pollution problems. Several air pollution measurement sensors are scattered around Japan as part of this system. Each station collects the data of various air pollutants, say $PM_{2.5}$, NO_2, and O_3, on an hourly basis. For our experiment, we confine to $PM_{2.5}$, since that particular particle size is the primary contributor to the wide variety of cardio-respiratory issues experienced by Japanese citizens. According to Air Quality Index Standards, $PM_{2.5}$ values greater than 16 $\mu g/m^3$ per hour are unsuitable for people.

5.2 Evaluation of Algorithm by Varying only *k*

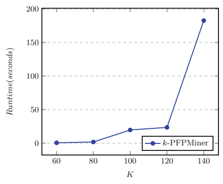
Figures 2a, 2b, 2c and 2d show the time consumed at a different number of k values in T10I10D100K, Retail, Congestion and Pollution databases, respectively. It can be observed that an increase in K increases the runtime to find all top-k periodic-frequent patterns being generated at different k values. As k increases, the number of patterns to be mined increases, resulting in time consumption.



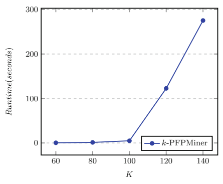
(a) T10I4D100K



(b) Retail

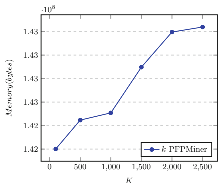


(c) Congestion

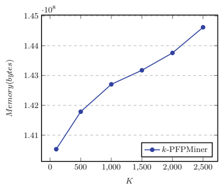


(d) Pollution

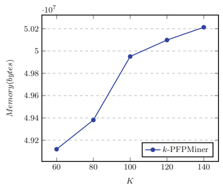
Fig. 2. Runtime evaluation on various databases by varying k



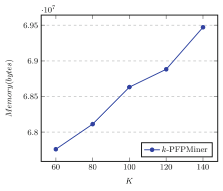
(a) T10I4D100K



(b) Retail



(c) Congestion



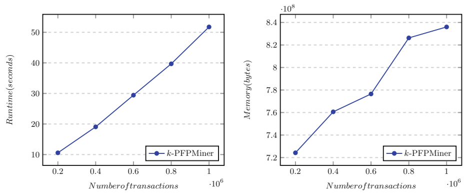
(d) Pollution

Fig. 3. Memory evaluation on various databases by varying k

Figures 3a, 3b, 3c and 3d show the memory consumed at a different number of k values in T10I10D200K, Retail, Congestion and Pollution databases, respectively. It can be observed that an increase in K increases the memory to find all top-k periodic-frequent patterns being generated at different k values.

5.3 Scalability Test

In this experiment, we have used the Kosarak database, which is a huge database having 9,90,000 transactions (in count). We have divided this database into five segments, each consisting of 2,00,000 transactions. We have evaluated the performance of k-PFPMiner by adding each successive segment to the ones that came before it. The runtime requirements and memory consumption k-PFPMiner for each segment of the Kosarak database are shown in Fig. 4a and 4b, when $k = 200$. The following are some noteworthy findings that can be derived from these figures: (i) runtime requirements of k-PFPMiner increases almost proportionally as database size grows. (ii) memory requirements of k-PFPMiner where we can observe same as 4a.



(a) Runtime (b) Memory

Fig. 4. Scalability of k-PFPMiner

6 Conclusions and Future Work

In this paper, we have proposed an efficient depth-first search algorithm, called top-k Periodic-frequent Pattern Miner (k-PFPMiner), to find all desired patterns in big temporal databases. We have solved the open research problem of setting $maxPer$ and $minSup$ constraints by introducing a novel upper-bound measure named *dynamic maximum periodicity*. With the help of a novel pruning technique, we have reduced the best and worst-case time complexity of identifying whether a pattern is a periodic or aperiodic pattern to O(1) and O(n), respectively. An in-depth examination of the proposed k-PFPMiner approach on four synthetic and real-world databases revealed that its memory consumption and runtime are efficient and highly scalable. As for future work, we will work on discovering top top-k periodic-frequent patterns in uncertain databases.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216 (1993)
2. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-K periodic-frequent pattern from transactional databases without support threshold. In: Papasratorn, B., Chutimaskul, W., Porkaew, K., Vanijja, V. (eds.) IAIT 2009. CCIS, vol. 55, pp. 18–29. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10392-6_3
3. Anirudh, A., Kiran, R.U., Reddy, P.K., Kitsuregawa, M.: Memory efficient mining of periodic-frequent patterns in transactional databases. In: 2016 IEEE Symposium Series on Computational Intelligence, pp. 1–8 (2016)
4. Fournier-Viger, P., Yang, P., Kiran, R.U., Ventura, S., Luna, J.M.: Mining local periodic patterns in a discrete sequence. Inf. Sci. **544**, 519–548 (2021)
5. Fournier-Viger, P., Yang, P., Lin, J.C.-W., Kiran, R.U.: Discovering stable periodic-frequent patterns in transactional data. In: Wotawa, F., Friedrich, G., Pill, I., Koitz-Hristov, R., Ali, M. (eds.) IEA/AIE 2019. LNCS (LNAI), vol. 11606, pp. 230–244. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22999-3_21
6. Kiran, R.U., et al.: Discovering fuzzy periodic-frequent patterns in quantitative temporal databases. In: FUZZ-IEEE 2020, pp. 1–8 (2020)
7. Kiran, R.U., Kitsuregawa, M.: Novel techniques to reduce search space in periodic-frequent pattern mining. In: DASFAA, pp. 377–391 (2014)

8. Kiran, R.U., Shang, H., Toyoda, M., Kitsuregawa, M.: Discovering recurring patterns in time series. In: Proceedings of the 18th International Conference on Extending Database Technology, pp. 97–108 (2015)
9. Kiran, R.U., Venkatesh, J., Toyoda, M., Kitsuregawa, M., Reddy, P.K.: Discovering partial periodic-frequent patterns in a transactional database. J. Syst. Softw. **125**, 170–182 (2017)
10. Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: a 25 years review. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **9**(6) (2019)
11. Ravikumar, P., Likhitha, P., Venus Vikranth Raj, B., Uday Kiran, R., Watanobe, Y., Zettsu, K.: Efficient discovery of periodic-frequent patterns in columnar temporal databases. Electronics **10**(12) (2021)
12. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering periodic-frequent patterns in transactional databases. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS (LNAI), vol. 5476, pp. 242–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01307-2_24