The Journal of Systems and Software 125 (2017) 170–182

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Discovering partial periodic-frequent patterns in a transactional database

R. Uday Kiran [a,b,*], J.N. Venkatesh [d], Masashi Toyoda [a], Masaru Kitsuregawa [a,c], P. Krishna Reddy [d]

[a] The University of Tokyo, Japan
[b] National Institute of Communication Technology, Japan
[c] National Institute of Informatics, Japan
[d] International Institute of Information Technology-Hyderabad, India

ARTICLE INFO

Article history:
Received 24 March 2016
Revised 27 September 2016
Accepted 23 November 2016
Available online 28 November 2016

Keywords:
Data mining
Knowledge discovery in databases
Pattern mining
Partial periodicity
Algorithms

ABSTRACT

Time and frequency are two important dimensions to determine the interestingness of a pattern in a database. Periodic-frequent patterns are an important class of regularities that exist in a database with respect to these two dimensions. Current studies on periodic-frequent pattern mining have focused on discovering full periodic-frequent patterns, i.e., finding all frequent patterns that have exhibited complete cyclic repetitions in a database. However, partial periodic-frequent patterns are more common due to the imperfect nature of real-world. This paper proposes a flexible and generic model to find partial periodic-frequent patterns. A new interesting measure, *periodic-ratio*, has been introduced to determine the periodic interestingness of a frequent pattern by taking into account its proportion of cyclic repetitions in a database. The proposed patterns do not satisfy the anti-monotonic property. A novel pruning technique has been introduced to reduce the search space effectively. A pattern-growth algorithm to find all partial periodic-frequent patterns has also been presented in this paper. Experimental results demonstrate that the proposed model can discover useful information, and the algorithm is efficient.

1. Introduction

Frequent patterns (or itemsets) are an important class of regularities that exist in a transactional database. These patterns play a key role in many knowledge discovery tasks such as association rule mining (Agrawal et al., 1993; Han et al., 2007), clustering (Zhang et al., 2010), classification (Dong and Li, 1999) and recommender systems (Adomavicius and Tuzhilin, 2001, 2005). The popular adoption and successful industrial application of finding these patterns has been hindered by a major obstacle: *frequent pattern mining often generates a huge number of patterns and majority of them may be found insignificant depending on application or user requirements.* When confronted with this problem in real-world applications, researchers have tried to reduce the desired set by finding user interest-based patterns such as maximal frequent patterns (Gouda and Zaki, 2001), demand driven patterns, utility patterns (Yao et al., 2004), constraint-based patterns (Pei et al., 2004), diverse-frequent patterns (Swamy et al., 2014), top-*k* patterns (Han et al., 2002) and periodic-frequent patterns (Tanbeer et al., 2009). This paper focuses on finding periodic-frequent patterns.

An important criterion to assess the interestingness of a frequent pattern is its temporal occurrences in a database. That is, whether a frequent pattern is occurring periodically, irregularly, or mostly at specific time intervals in a database. The class of frequent patterns that are occurring periodically within a database are known as periodic-frequent patterns. Finding these patterns is a significant task with many real-world applications. Examples include improving the performance of recommender systems (Stormer, 2007), intrusion detection in computer networks (Ma and Hellerstein, 2001) and discovering events in Twitter (Kiran et al., 2015). A classic application to illustrate the usefulness of these patterns is market-basket analysis. It analyzes how regularly the set of items are being purchased by the customers. An example of a periodic-frequent pattern is as follows:

$\{Bat, Ball\}$ $[support = 5\%, \quad periodicity = 1\ hour]$.

* Corresponding author.
E-mail addresses: uday_rage@tkl.iis.u-tokyo.ac.jp, uday.rage@gmail.com (R.U. Kiran), jn.venkatesh@research.iiit.ac.in (J.N. Venkatesh), toyoda@tkl.iis.u-tokyo.ac.jp (M. Toyoda), kitsure@tkl.iis.u-tokyo.ac.jp (M. Kitsuregawa), pkreddy@iiit.ac.in (P.K. Reddy).
URL: http://researchweb.iiit.ac.in/~uday_rage/index.html (R.U. Kiran), http://www.tkl.iis.u-tokyo.ac.jp/~toyoda/index_e.html (M. Toyoda), http://www.tkl.iis.u-tokyo.ac.jp/Kilab/Members/memo/kitsure_e.html (M. Kitsuregawa), http://faculty.iiit.ac.in/~pkreddy/index.html (P.K. Reddy)

http://dx.doi.org/10.1016/j.jss.2016.11.035
0164-1212/

The above pattern says that 5% of the customers have periodically purchased the items '*Bat*' and '*Ball*' at least once in every hour. This predictive behavior of the customers' purchases can facilitate the users in product recommendation and inventory management.

The problem of finding periodic-frequent patterns has been widely studied in the past (Tanbeer et al., 2009; Amphawan et al., 2009; Kiran and Reddy, 2010; Surana et al., 2011; Kiran and Kitsuregawa, 2014). However, most of these studies have focused on finding full periodic-frequent patterns, i.e., frequent patterns that have exhibited complete cyclic repetitions in a database. A useful related type of periodic-frequent patterns is partial periodic-frequent patterns, i.e., frequent patterns that have exhibited partial cyclic repetitions in a database. These patterns are a looser kind of full periodic-frequent patterns, and they exist ubiquitously in the real-world databases. The proposed patterns can find useful information in many real-life applications. Few examples are as follows:

- In the market-basket analysis, partial periodic-frequent patterns provide useful information pertaining to regularly purchased itemsets. This information can be useful for inventory management.
- Partial periodic-frequent pattern mining on a web-log data can find the sets of web pages that were not only visited heavily, but also regularly by the users. This information can be useful for the user for improved web site design or web administration.
- In an accident data set, partial periodic-frequent patterns can discover useful information pertaining to the periodicity of regularly occurring accidents. This information can be high useful for improving passenger safety.
- Partial periodic behavior of sets of hashtags has been exploited to discover minor events from Twitter data (Kiran et al., 2015).

The purpose of this paper is to discover partial periodic-frequent patterns efficiently.

Finding partial periodic-frequent patterns is a non-trivial and challenging task. The reasons are as follows:

- The problem of finding partial periodic patterns has been widely studied in time series data (Han et al., 1998; 1999; Yang et al., 2003; Aref et al., 2004). Unfortunately, these studies cannot be enhanced to find partial periodic-frequent patterns in a transactional database. The main reason is that these studies consider time series as a symbolic sequence, and therefore, do not take into account the actual temporal information of the items within a series.
- Existing full periodic-frequent pattern mining approaches assess the periodic interestingness of a frequent pattern by simply determining whether all of its *periods* (or inter-arrival times) are within the user-specified maximum *period* threshold value. As a result, there exists no interestingness measure to assess the partial periodic behavior of a frequent pattern in a transactional database.

With this motivation, this paper introduces a generic and flexible model to find partial periodic-frequent patterns. The proposed model is generic because it allows the user to find both full and partial periodically occurring frequent patterns. The model is flexible as it enables every pattern to satisfy a different minimum number of cyclic repetitions depending on its frequency. This flexible nature of our model facilitates us to capture the non-uniform frequencies of the items (or patterns) within a database effectively.

The contributions of this paper are as follows:

- A generic model of partial periodic-frequent patterns has been proposed in this paper. This model employs a new measure, *periodic-ratio*, to determine the partial periodic interestingness of a frequent pattern in a database. The proposed measure determines the interestingness of a pattern by taking into account its proportion of cyclic repetitions in a database.
- The patterns discovered by the proposed model do not satisfy the anti-monotonic property. That is, all non-empty subsets of a partial periodic-frequent pattern may not be partial periodic-frequent patterns. This increases the search space, which in turn increases the computational cost of finding these patterns. A novel pruning technique has been introduced to reduce the search space and the computational cost of these patterns effectively.
- A pattern-growth algorithm, called Generalized Periodic-Frequent pattern-growth (GPF-growth), has been proposed to discover the complete set of partial periodic-frequent patterns in a database.
- Experimental results demonstrate that the proposed model can discover useful information, and GPF-growth is efficient.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the existing model of periodic-frequent patterns. Section 4 introduces the proposed model of partial periodic-frequent patterns. Section 5 describes the GPF-growth algorithm. Section 6 reports on the experimental results. Section 7 concludes the paper with future research directions.

The problem of finding partial periodic-frequent patterns was first discussed in Kiran and Reddy (2011). In this paper, we introduce a novel pruning technique to tackle the predicament caused by the violation of anti-monotonic property by the *periodic-ratio* measure. We also provide theoretical correctness of GPF-growth, and discuss the usefulness of proposed model by conducting extensive experiments on both synthetic and real-world databases.

2. Related work

Time series is a collection of events obtained from sequential measurements over time. The problem of finding partial periodic patterns has received a great deal of attention in time series data (Han et al., 1998; 1999; Yang et al., 2003; Berberidis et al., 2002; Cao et al., 2004; Aref et al., 2004; Chen et al., 2011). The basic model used in all of these approaches, however, remains the same. It involves the following two steps (Han et al., 1998):

1. Partition the time series into distinct subsets (or periodic-segments) of a fixed length (or *period*).
2. Discover all partial periodic patterns that satisfy the user-specified minimum support (*minSup*). The *minSup* controls the minimum number of periodic-segments that a pattern must cover.

Example 1. Given the time series $TS = a\{bc\}baebace$ and the user-specified $period = 3$, TS is divided into three periodic-segments: $TS_1 = a\{bc\}b$, $TS_2 = aeb$ and $TS_3 = ace$. Let $a \star b$ be a pattern, where '$\star$' denotes a wild character that can represent any single set of events. This pattern appears in the periodic-segments of TS_1 and TS_2. Therefore, its *support* count is 2. If the user-defined $minSup = 2$, then $a \star b$ represents a partial periodic pattern in TS.

The major limitation of this model is that it considers time series as a symbolic sequence, and therefore, do not take into account the actual temporal information of the events within a sequence.

Ma and Hellerstein (2001) have proposed an alternative partial periodic pattern model by considering the temporal information of items within a series. This model considers time series as a time-based sequence, specifies *period* for each pattern using Fast Fourier Transformation (FFT), and discovers all patterns that satisfy the user-defined minimum support (*minSup*). The discovered patterns are known as **p-patterns**. Unfortunately, this model is

computationally expensive to use in real-world applications. The reason is as follows: *The computational cost of FFT to find period of a pattern is O(nlog n), where n represents the number of time units. As n can be a very large number in voluminous databases, specifying the period for each pattern using FFT can make this model computationally expensive to use in real-life.*

Kiran et al. (2015) have studied the problem of finding recurring patterns by representing time series (Ma and Hellerstein, 2001) as a transactional database. These patterns are distinct from the proposed partial periodic-frequent patterns as the former patterns exhibit full periodic behavior in distinct subsets of the data.

Özden et al. (1998) have enhanced the transactional database by a time attribute that describes the time when a transaction has appeared, investigated the periodic behavior of the patterns to discover cyclic association rules. In this study, a database is fragmented into non-overlapping subsets with respect to time. The association rules that are appearing in at least a certain number of subsets are discovered as cyclic association rules. By fragmenting the data and counting the number of subsets in which a pattern occurs greatly simplifies the design of the mining algorithm. However, the drawback is that patterns (or association rules) that span multiple windows cannot be discovered.

Tanbeer et al. (2009) have described a simplified model to find periodic-frequent patterns in a transactional database. Without any need of data fragmentation, this model discovers all frequent patterns that have exhibited complete (or full) cyclic repetitions in a database. Amphawan et al. (2009) have investigated the problem of finding top-k periodic-frequent patterns in a transactional database. Kiran and Reddy (2010) and Surana et al. (2011) have enhanced the Tanbeer's model to discover periodic-frequent patterns involving both frequent and rare items effectively. Kiran and Kitsuregawa (2014); Kiran et al. (2016) have discussed greedy search techniques to discover periodic-frequent patterns effectively. All of these studies have focused on finding full periodic-frequent patterns, and therefore, fail to discover those interesting frequent patterns that have exhibited partial cyclic repetitions in a database. This paper focuses on finding partial periodic-frequent patterns, and thus, generalizing the existing model of full periodic-frequent patterns.

The problem of finding sequential patterns (Mooney and Roddick, 2013) and frequent episodes (Mannila, 1997; Laxman et al., 2007) has received a great deal of attention. However, it should be noted that these studies do not take into account the *periodicity* of a pattern.

Finding partial periodic patterns (Esling and Agon, 2012), motifs (Oates, 2002), and recurring patterns (Mohammad and Nishida, 2013) has also been studied in time series; however, the focus was on finding numerical curve patterns rather than the symbolic patterns.

Overall, the proposed model of finding partial periodic-frequent patterns in a transactional database is novel and distinct from current models.

3. The basic model of periodic-frequent patterns

Let $I = \{i_1, i_2, \ldots, i_n\}$, $1 \leq n$, be the set of n items. Let $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β number of items is called a **β-pattern**. A **transaction**, $tr_k = (ts, Y)$, $1 \leq k$, is a tuple, where $ts \in \mathbb{R}$ represents the timestamp of Y pattern. For a transaction $tr_k = (ts, Y)$, such that $X \subseteq Y$, it is said that X occurs in tr_k and such timestamp is denoted as ts^X. A **transactional database** *TDB* over I is a set of transactions, $TDB = \{tr_1, \ldots, tr_m\}$, $m = |TDB|$, where $|TDB|$ represents the total number of transactions in a database. Let $TS^X = \{ts^X_j, \ldots, ts^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where X has occurred in *TDB*.

Table 1
Running example: a transactional database.

ts	Items	ts	Items
1	*ac*	8	*cdfg*
2	*abg*	9	*ab*
3	*de*	10	*cdef*
4	*abcd*	11	*abef*
6	*abcd*	12	*abcdg*
7	*abe*	13	*cef*

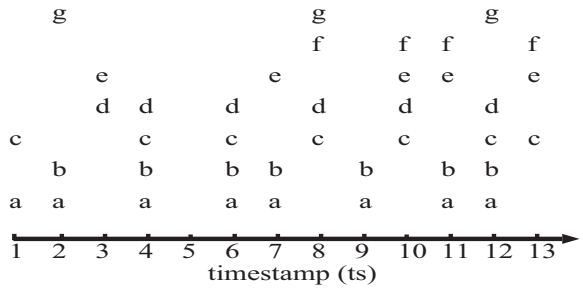


Fig. 1. Temporal occurrences of the items in Table 1.

Example 2. Consider the transactional database shown in Table 1. The set of all items in this database, i.e., $I = \{a, b, c, d, e, f, g\}$. Each transaction in this database is uniquely identifiable with a timestamp (*ts*). This database contains 12 transactions. Therefore, $|TDB| = 12$. This database do not contain any transaction with timestamp 5. However, it has to be noted that this timestamp still contributes in determining the periodic interestingness of a pattern in time dimension. Fig. 1 shows the temporal appearances of the items in Table 1. The set of items '*a*' and '*b*', i.e., *ab* is a pattern. This pattern contains only two items. Therefore, this pattern is referred as a 2-pattern. It can be observed that '*ab*' appears at the timestamps of 2, 4, 6, 7, 9, 11 and 12. Therefore, the ordered list of timestamps containing '*ab*, ' i.e., $TS^{ab} = \{2, 4, 6, 7, 9, 11, 12\}$.

Definition 1 (The support of X)**.** The number of transactions containing X in *TDB* is defined as the **support** of X and denoted as $sup(X)$. That is, $sup(X) = |TS^X|$.

Example 3. The *support* of '*ab*' in Table 1 is the size of TS^{ab}. Therefore, $sup(ab) = |TS^{ab}| = |\{2, 4, 6, 7, 9, 11, 12\}| = 7$.

Definition 2 (Frequent pattern X)**.** The pattern X is said to be frequent if $sup(X) \geq minSup$, where *minSup* represents the user-specified minimum support threshold value.

Example 4. Continuing with the previous example, if the user-specified $minSup = 3$, then '*ab*' is a frequent pattern because $sup(ab) \geq minSup$.

Definition 3 (A period of X)**.** Given the TS^X, a *period* of X, denoted as p^X_k, is calculated as follows:

- $p^X_k = ts^X_a - ts_{ini}$, $k = 1$, where $ts_{ini} = 0$ denotes the initial timestamp of all transactions in *TDB*.
- $p^X_k = ts^X_q - ts^X_p$, $1 < k < sup(X) + 1$, where ts^X_p and ts^X_q, $a \leq p \leq q \leq c$, denote any two consecutive timestamps in TS^X.
- $p^X_k = ts_{fin} - ts^X_c$, $k = sup(X) + 1$, where ts_{fin} denotes the final timestamp of all transactions in *TDB*.

The terms 'ts_{ini}' and 'ts_{fin}' play a key role in determining the periodic appearance of X in the entire database. Let $P^X = \{p^X_1, p^X_2, \cdots, p^X_k\}$, $k = sup(X) + 1$, denote the set of all periods of X in *TDB*. The first *period* in P^X (i.e., p^X_1) provides useful information pertaining to the time taken for initial appearance of X in *TDB*. The last *period* in P^X (i.e., $p^X_{sup(X)+1}$) provides useful information pertaining to the time elapsed after the final appearance of X in

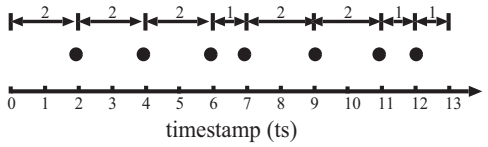


Fig. 2. Periods of '*ab*', P^{ab}.

Table 2
Periodic-frequent patterns discovered from Table 1. The terms **P**, ***S*** and ***PR*** represent 'Pattern,' '*support*' and '*periodic-ratio*, ' respectively. The columns titled '*I*' and '*II*' represent the periodic-frequent patterns generated by basic model and proposed model, respectively.

P	*S*	*PR*	*I*	*II*	P	*S*	*PR*	*I*	*II*
a	8	1.0	✓	✓	*f*	4	0.8	×	✓
b	7	1.0	✓	✓	*ab*	7	1.0	✓	✓
c	7	0.88	×	✓	*cd*	5	0.83	×	✓
d	6	0.86	×	✓	*ef*	3	0.75	×	✓

TDB. Remaining periods in P^X provide information pertaining to the inter-arrival times of *X* in *TDB*.

Example 5. The initial and final timestamps of the transactions in Table 1 are 0 and 13, respectively. Therefore, $ts_{ini} = 0$ and $ts_{fin} = 13$. Given the $TS^{ab} = \{2, 4, 6, 7, 9, 11, 12\}$, all *periods* of '*ab*' are calculated as follows: $p_1^{ab} = ts_1^{ab} - ts_{ini} = 2 - 0 = 2$, $p_2^{ab} = 4 - 2 = 2$, $p_3^{ab} = 6 - 4 = 2$, $p_4^{ab} = 7 - 6 = 1$, $p_5^{ab} = 9 - 7 = 2$, $p_6^{ab} = 11 - 9 = 2$, $p_7^{ab} = 12 - 11 = 1$ and $p_8^{ab} = ts_{fin} - ts_{10}^{ab} = 13 - 12 = 1$. Therefore, the complete set of *periods* of '*ab*' in Table 1, i.e., $P^{ab} = \{2, 2, 2, 1, 2, 2, 1, 1\}$. Fig. 2 shows the periods of *ab* in Table 1. The X-axis represents the timestamp. The black circles represent the timestamps at which '*ab*' has appeared in a transactional database.

Definition 4 (Periodicity of pattern *X*)**.** The maximum *period* in P^X represents the *periodicity* of *X*, and denoted as *periodicity*(*X*). That is, $periodicity(X) = max(p_i^X | \forall p_i^X \in P^X)$.

Example 6. The *periodicity* of '*ab*' in Table 1, i.e., $periodicity(ab) = max(2, 2, 2, 1, 2, 2, 1, 1) = 2$.

Definition 5 (Periodic-frequent pattern *X*)**.** The frequent pattern *X* is said to be a *periodic-frequent pattern* if $periodicity(X) \leq maxPeriodicity$, where *maxPeriodicity* represents the user-specified maximum periodicity.

Example 7. If the user-specified $maxPeriodicity = 2$, then the frequent pattern '*ab*' represents a periodic-frequent pattern because $periodicity(ab) \leq maxPeriodicity$.

The *support* and *periodicity* of a pattern can also be expressed in percentage of $(ts_{fin} - ts_{ini})$. However, we use the former definitions for ease of explanation.

The **problem definition** of the basic model is as follows: given a *TDB* and the user-specified *minSup* and *maxPeriodicity* constraints, discover all patterns in *TDB* that have support no less than *minSup* and *periodicity* no more than *maxPeriodicity*. The complete set of periodic-frequent patterns generated from Table 1 are shown in the column titled '*I*' of Table 2.

4. Proposed model

In this section, we first describe the major obstacle encountered by the basic model of periodic-frequent patterns. We next introduce the proposed model of partial periodic-frequent patterns.

4.1. The limitation of basic model

The *maxPeriodicity* plays a key role in the practical applicability of periodic-frequent patterns. This constraint ensures that the *anti-monotonic property* (see Property 1) of frequent patterns is still preserved while finding periodic-frequent patterns. Since *maxPeriodicity* determines the interestingness of a pattern by taking into account only its maximum *period*, all periodic-frequent patterns generated by the basic model represent only full periodic-frequent patterns. In other words, this model fails to discover those interesting frequent patterns that have exhibited partial cyclic repetitions in a database.

Example 8. In Table 1, the pattern '*cd*' appears at the timestamps of 4, 6, 8, 10 and 12. Therefore, $TS^{cd} = \{4, 6, 8, 10, 12\}$, $sup(cd) = 5$, $P^{cd} = \{4, 2, 2, 2, 2, 1\}$ and $per(cd) = 4$. If the user-specified $minSup = 3$ and $maxPeriodicity = 2$, then the basic model fails to generate '*cd*' as a periodic-frequent pattern because $per(cd) \not\leq maxPeriodicity$. However, this pattern may still be interesting to the users as most of its *periods* (or reoccurrences) are within the *maxPeriodicity*.

Property 1. *A measure M is anti-monotonic if a pattern X satisfying M implies that every non-empty subset of X also satisfies M.*

4.2. Partial periodic-frequent pattern model

In the proposed model, the definition of frequent patterns remains the same. However, the definition of periodic-frequent patterns is changed to capture the partial periodic behavior of frequent patterns. The basic intuition of the proposed model involves determining the periodic interestingness of a pattern by taking into account its proportion of cyclic repetitions in a database. The proposed model is as follows.

Definition 6 (An interesting period of *X*)**.** A period of *X* is said to be interesting if it is no more than the user-specified maximum period (*maxPer*). That is, a $p_i^X \in P^X$ is said to be interesting if $p_i^X \leq maxPer$.

Example 9. Consider the first and seconds *periods* of '*cd*' (i.e., p_1^{cd} and p_2^{cd}) in Example 8. If the user-specified $maxPer = 2$, then p_1^{cd} will be considered as an uninteresting *period* because $p_1^{cd} \not\leq maxPer$. On the contrary, p_2^{cd} will be considered as an interesting *period* because $p_2^{cd} \leq maxPer$.

The *maxPer* constraint determines whether an occurrence of a pattern is cyclic or acyclic in a database. We now define our measure that determines the partial periodic interestingness of a pattern by taking into account its number of cyclic repetitions in a database.

Definition 7 (Periodic-ratio of *X*)**.** Let IP^X be the set of all *periods* in P^X that satisfy the user-specified *maxPer*. That is, $IP^X \subseteq P^X$ such that if $\exists p_k^X \in P^X : p_k^X \leq maxPer$, then $p_k^X \in IP^X$. The periodic-ratio of a pattern *X*, denoted as *PR*(*X*), represents the ratio of $|IP^X|$ to $|P^X|$. That is, $PR(X) = \frac{|IP^X|}{|P^X|}$.

Example 10. The set of all interesting *periods* of '*cd*' in Table 1, i.e., $IP^{cd} = \{2, 2, 2, 2, 1\}$. Therefore, the *periodic-ratio* of '*cd*, ' i.e., $PR(cd) = \frac{|IP^{cd}|}{|P^{cd}|} = \frac{5}{6} = 0.833$ $(= 83.3\%)$.

The *periodic-ratio*, as defined above, determines the proportion of cyclic occurrences of a pattern in a transactional database. For a pattern *X*, $PR(X) \in [0, 1]$. If $PR(X) = 0$, then *X* is an aperiodic-frequent pattern because all of its *periods* will be more than the user-specified *maxPer* value. If $PR(X) = 1$, then *X* is a full periodic-frequent pattern as all of its *periods* will be within the user-defined *maxPer* value. Thus, generalizing the existing model of periodic-frequent patterns (Tanbeer et al., 2009).

Definition 8 (Partial periodic-frequent pattern *X*)**.** The frequent pattern *X* is said to be *partial periodic-frequent* if $PR(X) \geq minPR$, where *minPR* represents the user-specified *minimum periodic-ratio*.

Example 11. If the user-defined $minPR = 0.75$, then the frequent pattern '*cd*' represents a partial periodic-frequent pattern because $PR(cd) \geq minPR$. The complete set of partial periodic-frequent patterns discovered from Table 1 are shown in the column titled *II* of Table 2. It can be observed from this table that the proposed model has not only discovered full periodically occurring frequent patterns, but also discovered those interesting frequent patterns that have exhibited partial cyclic repetitions in the database.

If *X* is a partial periodic-frequent pattern, then $|IP^X| \geq (minPR \times (sup(X)+1))$. The correctness of this statement is based on Property 2 and shown in Lemma 1. Thus, *periodic-ratio* facilitates every pattern to satisfy a different minimum number of cyclic repetitions (i.e., $|IP^X|$) depending on its *support*. This flexible nature of *periodic-ratio* facilitates the proposed model to capture the varied frequencies of the patterns in a database.

Example 12. Consider the patterns '*ab*' and '*ef*' in Table 1. For the pattern '*ab*, ' $TS^{ab} = \{2, 4, 6, 7, 9, 11, 12\}$ and $sup(ab) = 7$. For the pattern '*ef*, ' $TS^{ef} = \{10, 11, 13\}$ and $sup(ef) = 3$. Given the user-defined $minPR = 0.75$, '*ab*' can be a partial periodic-frequent pattern if and only if $|IP^{ab}| \geq 6$ ($\approx 0.75 \times (7+1)$). Similarly, '*ef*' can be a partial periodic-frequent pattern if and only if $|IP^{ef}| \geq 3$ ($= 0.75 \times (3+1)$). Therefore, the *periodic-ratio* facilitates every pattern to satisfy a different number of minimum cyclic repetitions depending upon its *support*.

An advantage of *periodic-ratio* measure is that it enables the user to find partial periodic-frequent patterns using quartiles. For example, given the user-specified $minPR = 0.5$ (or 50%), the proposed model considers a frequent pattern as partial periodic-frequent if the *median* of its *periods* is no more than the user-specified *maxPer* value.

Property 2. *For a pattern X,* $|P^X| = sup(X) + 1$.

Lemma 1. *If X is a partial periodic-frequent pattern, then* $|IP^X| \geq (minPR \times (sup(X)+1))$.

Proof. If *X* is a partial periodic-frequent pattern, then

$$\frac{|IP^X|}{|P^X|} \geq minPR$$

$$= \frac{|IP^X|}{sup(X)+1} \geq minPR \qquad (\because |P^X| = sup(X)+1)$$

$$= |IP^X| \geq minPR \times (sup(X)+1). \qquad (1)$$

Hence proved. □

Definition 9 (Problem definition)**.** Given the transactional database (*TDB*) and the user-defined minimum support (*minSup*), maximum *period* (*maxPer*) and minimum periodic-ratio (*minPR*), discover all partial periodic-frequent patterns that have *support* and *periodic-ratio* no less than the *minSup* and *minPR*, respectively.

5. Proposed algorithm

5.1. Basic idea: candidate patterns

The space of items in a database gives rise to an itemset lattice. This lattice is the conceptualization of search space while finding interesting patterns. Most of the pattern mining algorithms employ the *anti-monotonic property* to reduce this search space effectively. Unfortunately, the proposed patterns do not satisfy this property (see Example 13). Thus increasing the search space and the computational cost of finding the partial periodic-frequent patterns.

Example 13. Consider the pattern '*e*' in Table 1. This pattern appears at the timestamps of 3, 7, 10, 11 and 13. Therefore, $TS^e = \{3, 7, 10, 11, 13\}$, $sup(e) = 5$, $P^e = \{3, 4, 3, 1, 2, 0\}$, $IP^e = \{1, 2, 0\}$ and $PR(e) = \frac{3}{6} = 0.5$. As $sup(e) \geq minSup$ and $PR(e) \ngeq minPR$, '*e*' is not a partial periodic-frequent pattern. Let us consider another pattern '*ef*⊃*e*.' This pattern appears at the timestamps of 10, 11 and 13. Therefore, $TS^{ef} = \{10, 11, 13\}$, $sup(ef) = 3$, $P^{ef} = \{10, 1, 2, 0\}$, $IP^{ef} = \{1, 2, 0\}$ and $PR(ef) = \frac{3}{4} = 0.75$. The pattern '*ef*' is a partial periodic-frequent pattern because $sup(ef) \geq minSup$ and $PR(ef) \geq minPR$. The same can be observed in the column titled *II* in Table 2.

A naive pruning technique to reduce the search space involves **finding partial periodic-frequent** k-patterns using frequent $(k-1)$**-patterns**. The reason is that frequent patterns satisfy the anti-monotonic property. Initially, we have applied this pruning technique in the proposed algorithm (discussed in subsequent subsection) to find partial periodic-frequent patterns. While investigating the performance of the algorithm on many real-world databases, we have observed that the usage of above pruning technique alone is inefficient to reduce the search space. The reason is the above pruning technique was considering some of those frequent patterns whose supersets can never be partial periodic-frequent patterns.

Example 14. Consider the pattern '*g*' in Table 1. This pattern appears at the timestamps of 2, 8 and 12. Therefore, $TS^g = \{2, 8, 12\}$, $sup(g) = 3$ and $P^g = \{2, 6, 4, 1\}$. Given the user-specified $minSup = 3$, $maxPer = 2$ and $minPR = 0.75$, $IP^g = \{2, 1\}$ and $PR(g) = \frac{2}{4} = 0.5$. The pattern '*g*' is a frequent pattern because $sup(g) \geq minSup$. However, it is not a partial periodic-frequent pattern as $PR(g) \ngeq minPR$. The above pruning technique says that the mining algorithm has to consider the frequent pattern '*g*' to discover partial periodic-frequent patterns of higher-order. However, it can be observed that no superset of '*g*' can be a partial periodic-frequent pattern.

With this motivation, we introduce another pruning technique as follows: "*Let X and Y be the two patterns such that* $X \subset Y$ *and* $X \neq \emptyset$. *If* $|IP^X| < (minPR \times (minSup+1))$, *then neither X nor Y will be partial periodic-frequent patterns.*" The correctness of our pruning technique is based on the Property 3, and shown in Lemmas 2 and 3. The basic idea behind the above pruning technique is as follows: "every partial periodic-frequent pattern will have *support* no less than *minSup*. Therefore, every partial periodic-frequent pattern will have at least $(minSup+1)$ number of *periods*, and at least $minPR \times (minSup+1)$ number of interesting *periods*."

Example 15. Given the user-specified $minSup = 3$, $maxPer = 2$ and $minPR = 0.75$, our pruning technique says that if a pattern *X* has $|IP^X| < 3$ ($= minPR \times (minSup+1)$), then neither *X* nor its supersets can be partial periodic-frequent patterns. Continuing with the previous example, $|IP^g| < 3$. Therefore, our pruning technique do not consider '*g*' for finding partial periodic-frequent patterns of higher order.

Property 3. *If* $X \subset Y$, *then* $TS^X \supseteq TS^Y$. *Therefore,* $|P^X| \geq |P^Y|$ *and* $|IP^X| \geq |IP^Y|$.

Lemma 2. *If X is a partial periodic-frequent pattern, then* $|IP^X| \geq (minPR \times (minSup+1))$.

Proof. Based on Lemma 1 it turns out that if *X* is a partial periodic-frequent pattern, then

$$|IP^X| \geq minPR \times (sup(X)+1) \geq minPR \times (minSup+1). \qquad (2)$$

Hence proved. □

Lemma 3. *Let X and Y be the two patterns such that* $X \subset Y$. *If* $|IP^X| < minPR \times (minSup+1)$, *then both X and Y cannot be partial periodic-frequent patterns.*

Proof. According to Property 3 if $|IP^X| < minPR \times (minSup + 1)$, then $|IP^Y| < minPR \times (minSup + 1)$. Based on Lemma 2, it turns out that both X and Y cannot be partial periodic-frequent patterns. Hence proved. □

Based on the above pruning technique, we introduce the concept of candidate patterns.

Definition 10 (A candidate pattern X)**.** The pattern X is a candidate pattern if $sup(X) \geq minSup$ and $|IP^X| \geq (minPR \times (minSup + 1))$.

Example 16. Consider the aperiodic frequent items '*e*' and '*g*' in Table 1 (see Examples 13 and 14). For the item '*e*, ' $sup(e) \geq minSup$ and $|IP^e| \geq 3$. Therefore, this aperiodic frequent item will be considered as a candidate item (or 1-pattern) and used in finding partial periodic-frequent patterns of higher order. For the item '*g*, ' $sup(g) \geq minSup$ and $|IP^g| \not\geq 3$. Therefore, *g* will not be considered as a candidate item.

The candidate patterns satisfy the *anti-monotonic property* (see Lemma 4). Therefore, the proposed algorithm uses candidate *k*-patterns to find partial periodic-frequent $(k+1)$-patterns, $k \geq 1$.

Lemma 4. *Let X and Y be two patterns such that* $X \subset Y$ *and* $X \neq \emptyset$. *If Y is a candidate pattern, then X is also a candidate pattern.*

Proof. If $X \subset Y$, then $TS^X \supseteq TS^Y$, $sup(X) \geq sup(Y)$ and $|IP^X| \geq |IP^Y|$ (see Property 3). Therefore, if $sup(Y) \geq minSup$ and $|IP^Y| \geq minPR \times (minSup + 1)$, then $sup(X) \geq minSup$ and $|IP^X| \geq minPR \times (minSup + 1)$. Hence proved. □

In a transactional database, the relationship between the set of frequent patterns (*FPs*), candidate patterns (*CPs*), partial periodic-frequent patterns (*PPFPs*) and full periodic-frequent patterns (*FPFPs*) is $FPs \supseteq CPs \supseteq PPFPs \supseteq FPFPs$. Henceforth, although partial periodic-frequent patterns do not satisfy the anti-monotonic property, our pruning technique ensures that the search space for finding partial periodic-frequent patterns is no more than that of the frequent pattern mining. In other words, our pruning technique makes the partial periodic-frequent pattern model practicable in real-world applications.

5.2. Generalized periodic-frequent pattern-Growth

Traditional pattern-growth algorithms that rely on Frequent Pattern tree (FP-tree) cannot be used for finding periodic-frequent patterns. The reason is that FP-tree captures only the frequency and disregards the periodic behavior of the patterns in a database. Tanbeer et al. (2009) have introduced an alternative pattern-growth algorithm that relies on an enhanced FP-tree, called Periodic-Frequent Tree (PF-tree), to find full periodic-frequent patterns. We call this algorithm as Periodic-Frequent pattern-growth (PF-growth). Unfortunately, we cannot directly apply this algorithm to mine partial periodic-frequent patterns. The reasons are as follows: (*i*) The partial periodic-frequent patterns do not satisfy the *anti-monotonic property* and (*ii*) The *tree* structure must capture different interestingness measures, i.e., *maxPer* and *minPR*. In this paper, we enhance the PF-growth to discover the complete set of partial periodic-frequent patterns. We call this enhanced algorithm as Generalized Periodic-Frequent pattern-growth (GPF-growth).

The GPF-growth algorithm involves the following two steps: (*i*) compress the database into a tree structure, called Generalized Periodic-Frequent pattern-tree (GPF-tree) and (*ii*) recursively mining GPF-tree to discover the complete set of partial periodic-frequent patterns. Before we describe the above two steps, we introduce the structure of GPF-tree.

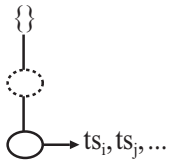


Fig. 3. Conceptual structure of prefix-tree in GPF-tree. Dotted ellipse represents ordinary node, while other ellipse represents tail-node of sorted transactions with timestamps ts_i, $ts_j \in R$.

5.2.1. Structure of GPF-tree

The structure of GPF-tree includes a prefix-tree and a candidate item list, called the GPF-list. The GPF-list consists of each distinct *item* (*I*) with *support* (*S*), number of interesting periods (*IP*), and a pointer pointing to the first node in the prefix-tree carrying the item.

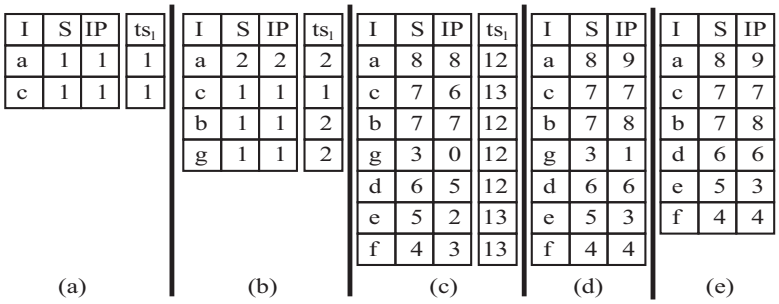
The prefix-tree in GPF-tree resembles the prefix-tree in FP-tree. However, in order to obtain both *support* and *periodic-ratio* of a pattern, the nodes in GPF-tree explicitly maintain the occurrence information of each transaction by keeping an occurrence timestamp list, called a ***ts*-list**. To achieve memory efficiency, only the last node of every transaction maintains the *ts*-list. Hence, two types of nodes are maintained in a GPF-tree: **ordinary node** and **tail-node**. The former is a type of node similar to that used in an FP-tree, whereas the latter represents the last item of any sorted transaction. Therefore, the structure of a *tail*-node is $i[ts_p, \ldots, ts_r]$, $1 \leq p \leq r \leq m$, where *i* is the node's item name and ts^i_k, $k \in [1, m]$, is the timestamp of a transaction containing the items from *root* up to the node *i*. The conceptual structure of GPF-tree is shown in Fig. 3. Like FP-tree, each node in GPF-tree maintains parent, children, and node traversal pointers. Please note that no node in GPF-tree maintains the support count as in a FP-tree. To facilitate a high degree of compactness, items in the prefix-tree are arranged in support-descending order.

One can assume that the structure of prefix-tree in GPF-tree may not be memory efficient since it explicitly maintains the timestamps of each transaction. However, it has been argued that such a tree can achieve memory efficiency by keeping transaction information only at the *tail*-nodes and avoiding the support count field at each node (Tanbeer et al., 2009). Furthermore, GPF-tree avoids the *complicated combinatorial explosion problem of candidate generation* as in Apriori-like algorithms (Agrawal et al., 1993). Keeping the information pertaining to transactional-identifiers in a tree can also be found in efficient frequent pattern mining (Zaki and Hsiao, 2005).

5.2.2. Construction of GPF-tree

Since partial periodic-frequent patterns do not satisfy the *anti-monotonic property*, candidate 1-patterns (or items) will play an important role in finding these patterns effectively. The set of candidate items *CI* in a database for the user-defined *minSup, maxPer* and *minPR* can be discovered by populating the GPF-list with a scan on the database. The procedure for constructing GPF-list is provided in Algorithm 1. Before we explain the steps of this algorithm, please note that *minSup, maxPer* and *minPR* values have been set to 3, 2 and 0.75, respectively.

The scan on the first transaction, "1: *ac*", with $ts_{cur} = 1$ initializes items '*a*' and '*c*' in the GPF-list by setting their *S, IP* and ts_l values to 1, 1 and 1, respectively (lines 3 to 10 in Algorithm 1). Fig. 4(*a*) shows the GPF-list generated after scanning the first transaction. The scan on the second transaction, "2: *abg*", with $ts_{cur} = 2$ initializes the items '*b*' and '*g*' in the GPF-list and sets their *S, IP* and ts_l values to 1, 1 and 2, respectively. Simultaneously, the *S, IP* and ts_l values of an already existing item '*a*' are updated to 2, 2 and 2, respectively (lines 11 to 16 in Algorithm 1). Similar pro-



(a)

I	S	IP	ts_l
a	1	1	1
c	1	1	1

(b)

I	S	IP	ts_l
a	2	2	2
c	1	1	1
b	1	1	2
g	1	1	2

(c)

I	S	IP	ts_l
a	8	8	12
c	7	6	13
b	7	7	12
g	3	0	12
d	6	5	12
e	5	2	13
f	4	3	13

(d)

I	S	IP
a	8	9
c	7	7
b	7	8
g	3	1
d	6	6
e	5	3
f	4	4

(e)

I	S	IP
a	8	9
c	7	7
b	7	8
d	6	6
e	5	3
f	4	4

Fig. 4. Construction of GPF-list. (*a*) After scanning first transaction (b) After scanning second transaction (*c*) After scanning entire database (*d*) Updating the number of interesting periods of all items in the database and (*e*) Sorted list of candidate items.

Algorithm 1 GPF-List(*TDB*: Transactional database, *I*: Set of items, *minSup*: minimum support, *maxPer*: maximum period, *minPR*: minimum periodic-ratio).

1: Let ts_l be a temporary array that records the *timestamp* of the last appearance of each item in the *TDB*. Let *ip* be another temporary array that records the number of interesting periods of an item in a database. Let $t = \{ts_{cur}, X\}$ denote the current transaction with ts_{cur} and *X* representing the timestamp of the current transaction and pattern, respectively.
2: **for** each transaction $t \in TDB$ **do**
3: **if** an item *i* occurs for the first time **then**
4: Add *i* to the GPF-list and set $s^i = 1$ and $ts_l^i = ts_{cur}$.
5: **if** $ts_{cur} \leq maxPer$ **then**
6: Set $ip^i = 1$;
7: **else**
8: Set $ip^i = 0$;
9: **end if**
10: **else**
11: **if** $(ts_{cur} - ts_l^i) \leq maxPer$ **then**
12: Update ip^i++.
13: **end if**
14: Update s^i++, ip^i++ and $ts_l^i = ts_{cur}$.
15: **end if**
16: **end for**
17: **for** each item *i* in GPF-list **do**
18: **if** $ts_{cur} - ts_l^i \leq maxPer$ **then**
19: Update ip^i++;
20: **end if**
21: **end for**
22: **for** each item *i* in GPF-list **do**
23: **if** $s^i \leq minSup$ or $ip^i < (minPR \times (minSup + 1))$ **then**
24: Remove *i* from the GPF-list.
25: **end if**
26: **end for**
27: Sort the remaining items in GPF-list in *support* descending order of items. Consider this sorted list of items as candidate items (*CI*).

Algorithm 2 GPF-Tree(*TDB*, GPF-list).

1: Create the root of GPF-tree, *T*, and label it "*null*".
2: **for** each transaction $t \in TDB$ **do**
3: Set the timestamp of the corresponding transaction as ts_{cur}.
4: Select and sort the candidate items in *t* according to the order of *CI*. Let the sorted candidate item list in *t* be $[p|P]$, where *p* is the first item and *P* is the remaining list.
5: Call *insert_tree*$([p|P], ts_{cur}, T)$.
6: **end for**
7: call GPF-growth (*Tree*, *null*);

Algorithm 3 insert_tree($[p|P]$, t_{cur}, *T*).

1: **while** *P* is non-empty **do**
2: **if** *T* has a child *N* such that *p.itemName* $\neq$ *N.itemName* **then**
3: Create a new node *N*. Let its parent link be linked to *T*. Let its node-link be linked to nodes with the same itemName via the node-link structure. Remove *p* from *P*.
4: **end if**
5: **end while**
6: Add t_{cur} to the leaf node.

cess is carried out for the remaining transactions in the database, and the GPF-list is updated accordingly. Fig. 4(*c*) shows the GPF-list generated after scanning every transaction in the database. The *IP* value of all items in the GPF-list is once again computed to reflect the correctness (lines 17 to 21 in Algorithm 1). Fig. 4(*d*) shows the updated *PS* value for all items in the GPF-list. Based the proposed pruning technique, the item '*g*' will be removed from the GPF-list as $IP(g) \not\geq minPR \times (minSup + 1)$ (lines 22 to 26 in Algorithm 1). The remaining items are sorted in descending order of their support values (line 27 in Algorithm 1). This sorted list of items are known as candidate items, and denoted as *CI*. Fig. 4(*e*) shows the complete list of all candidate items in the GPF-list.

After finding candidate items, prefix-tree of GPF-tree is constructed using the Algorithms 2 and 3. These two procedures are similar to those of the FP-tree construction (Han et al., 2004). However, the key difference is that no node in GPF-tree maintains the *support* count as in FP-tree. The prefix-tree is constructed as follows.

First, create the '*root*' node of the *tree* and labeled it with '*null*.' Scan the transactional database *TDB* a second time. The items in each transaction are processed in *CI* order (i.e., sorted list of candidate items), and a branch is created for each transaction. For example, the scan on the first transaction, '1: *ac*, ' which contains two items (*a* and *c* in *CI* order), leads to the construction of the first branch of the tree with two nodes $\langle a\rangle$ and $\langle c$: 1$\rangle$, where *a* is linked as a child of the *root* and *c* is linked as a child of '*a*.' As '*c*' is the leaf node of this branch, this node carries the timestamp of 1. Fig. 5(*a*) shows the prefix-tree generated after scanning the first transaction. The scan on the second transaction, '2: *abg*, ' containing the items '*a*' and '*b*' in *CI* order will result in another branch with two nodes $\langle a\rangle$ and $\langle b$: 2$\rangle$, where *a* is linked to the *root* and *b* is linked to '*a*.' The leaf node of this branch, i.e., '*b*' carries the timestamp of 2. However, this branch would share a common prefix, '*a*, ' with the existing path of first transaction. Therefore, we instead create a new node $\langle b$: 2$\rangle$ and connect it to the existing node $\langle a\rangle$. The resultant prefix-tree is shown in Fig. 5(*b*). Similar process is repeated for the remaining transactions and the prefix-tree is

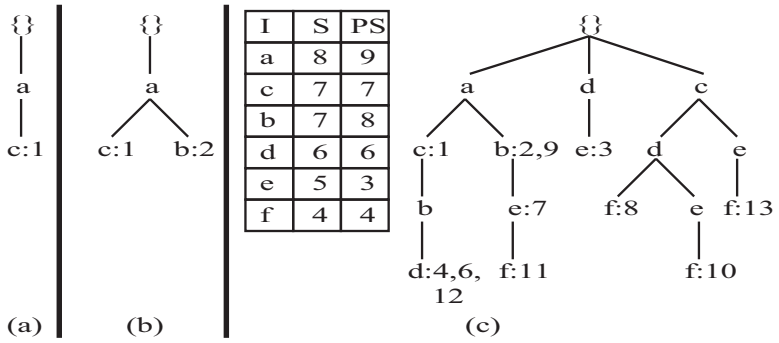


Fig. 5. Construction of GPF-Tree. (*a*) After scanning first transaction (b) After scanning second transaction and (*c*) After scanning entire database.

updated accordingly. The final GPF-tree generated after scanning every transaction in the database is shown in Fig. 5(*c*). Please note that every node in GPF-tree maintains parent, children, and node traversal pointers. For brevity, we are not showing the node traversal pointers, however, they are maintained as in FP-tree.

Based on the GPF-list population technique as discussed above, we have the following property and lemmas of a GPF-tree. For each transaction t in *TDB*, $CI(t)$ be the set of all candidate items in t, and is called the candidate item projection of t.

Property 4. *A GPF-tree maintains a complete set of candidate item projection for each transaction in a TDB only once.*

Lemma 5. *The complete set of all candidate item projections of all transactions in a TDB can derived from GPF-tree for the user-defined minSup, maxPeriodicity and minPR.*

Proof. Based on Property 4, $CI(t)$ of each transaction t is mapped to only one path in the tree and any path from the root up to a tail-node maintains the complete projection for exactly n transactions (where n is the total number of entries in ts-list of the tail-node). □

Lemma 6. *Size of GPF-tree excluding the root node is bound by* $\Sigma_{t \in TDB}|CI(t)|$.

Proof. As per the GPF-tree construction technique and Lemma 5, every transaction t contributes at most only one path of the size $|CI(t)|$ to a GPF-tree. Therefore, the overall size contribution of all transactions can be $\Sigma_{t \in TDB}|CI(t)|$ at best. However, since there are usually a lot of common prefix patterns among the transactions, the size of GPF-tree is typically smaller than $\Sigma_{t \in TDB}|CI(t)|$. □

5.3. Mining partial periodic-frequent patterns

Due to the structural differences between the FP-tree and GPF-tree, we cannot directly apply the pattern-growth technique of FP-growth to mine partial periodic-frequent patterns from GPF-tree. Henceforth, we propose an alternative pattern-growth-based bottom-up mining technique to discover partial periodic-frequent patterns from GPF-tree. The steps involved in mining GPF-tree are (*i*) finding partial periodic-frequent items (or 1-patterns), (*ii*) constructing the prefix-tree for each candidate pattern, and (*iii*) constructing the conditional tree from each prefix-tree. The periodic items are discovered from GPF-list. Before discussing the prefix-tree mining process we explore the following important property and lemma of GPF-tree.

Property 5. *A tail-node in a GPF-tree maintains the occurrence information for all the nodes in the path (from that tail-node to the root) at least in the transactions in its ts-list.*

Lemma 7. *Let* $B = \{b_1, b_2, \ldots, b_n\}$ *be a branch in GPF-tree where* b_n *is a tail-node carrying the ts-list of the branch. If the ts-list is pushed-up to node* b_{n-1}*, then* b_{n-1} *maintains the occurrence information of the path* $B' = \{b_1, b_2, \cdots, b_{n-1}\}$ *for the same set of transactions in the ts-list without any loss.*

Proof. According to Property 5, b_n maintains the occurrence information of the path B' at least in the transactions of its ts-list. Therefore, the same ts-list at node b_{n-1} maintains the same transaction information for B' without any lose. □

Algorithms 4 , 5 and 6 describe the procedure of finding partial periodic-frequent patterns in a GPF-tree. The working of these algorithms is as follows. Considering the bottom-most item i in GPF-list, we construct its prefix-tree with prefix sub-paths of nodes labeled i in GPF-tree. We call this prefix-tree of i as PT_i. Since i is the bottom-most item in the GPF-list, each node labeled i in the GPF-tree must be a tail-node. While constructing the PT_i, based on Property 5, we map the ts-list of every node of i to all

Algorithm 4 GPF-growth(*Tree*, α).

```
for each a_i in the header of Tree do
    Generate pattern β = a_i ∪ α. Collect all of the a_i's ts-lists into
    a temporary array, TS^β, and calculate PS(β).
    if PS(β) ≥ (minPR × (minSup + 1)) then
        Construct β's conditional pattern base then β's conditional
        GPF-tree Tree_β by calling getPeriodicRatio(β, TS^β).
        if Tree_β ≠ ∅ then
            call GPF-growth(Tree_β, β);
        end if
    end if
    Remove a_i from the Tree and push the a_i's ts-list to its parent
    nodes.
end for
```

Algorithm 5 IP(TS^β).

```
Let ts_i = 0 and ts_f = ts_m represent the initial and final timestamps of all transactions in TDB.
if TS^β[0] − ts_i ≤ maxPer then
    ps^β ++;
end if
for i = 0; i < TS^β.length − 1; ++i do
    if TS^β[i + 1] − TS^β[i] ≤ maxPer then
        ps^β ++
    end if
end for
if TS^β[TS^β.length − 1] − ts_f ≤ maxPer then
    ps^β ++.
end if
return ps^β.
```

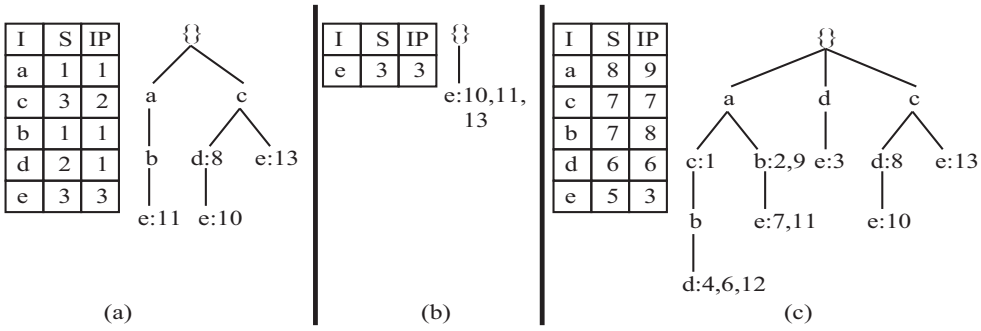


Fig. 6. Recursive mining of GPF-Tree. (*a*) Prefix-tree for '*f*' (*b*) Conditional-tree for '*f*' and (*c*) GPF-tree after removing the item '*f*'.

Algorithm 6 getPeriodicRatio(β, TS^{β}).

1: **if** $TS^{\beta}[0] - ts_i \leq maxPer$ **then**
2: $ps^{\beta}++$;
3: **end if**
4: **for** $i = 0; i < TS^{\beta}.length - 1; ++i$ **do**
5: **if** $TS^{\beta}[i+1] - TS^{\beta}[i] \leq maxPer$ **then**
6: $ps^{\beta}++$
7: **end if**
8: **end for**
9: **if** $TS^{\beta}[TS^{\beta}.length - 1] - ts_f \leq maxPer$ **then**
10: $ps^{\beta}++$.
11: **end if**
12: return $\frac{ps^{\beta}}{|TS^{\beta}|+1}$.

items in the respective path explicitly in a temporary array (one for each item). It facilitates the calculation of *support* and *periodic-ratio* for each item in PT_i. Using our pruning technique, the conditional tree CT_i for PT_i is constructed by removing all those items from PT_i that have number of interesting periods (i.e., *IP*) less than $minPR \times (minSup + 1)$. If the deleted node is a *tail* node, its *ts*-list is pushed-up to its parent node. The contents of the temporary array for the bottom item *j* in the GPF-list of CT_i represent TS^{ij} (i.e., the set of all timestamps where items *i* and *j* have appeared together in the database). Therefore, using Algorithm 6, the *periodic-ratio* of "*ij*" is computed to determine the periodic interestingness of "*ij*.". The same process of creating a prefix-tree and its corresponding conditional tree is repeated for the further extensions of "*ij*". The whole process of mining for each item is repeated until *GPF-list* $\neq \emptyset$. Moreover, to enable the construction of the prefix-tree for the next item in the GPF-list, based on Lemma 7 the ts-lists are pushed-up to respective parent nodes in the original GPF-tree as well. All nodes of *i* in the GPF-tree and *i*'s entry in the original GPF-tree are deleted thereafter.

Consider item '*f*', which is the last item in the GPF-list of Fig. 5(*c*). Collect all timestamps of '*f* in GPF-tree, construct TS^f and calculate *IP* of *f*, i.e., *IP*(*f*) (line 2 in Algorithm 4). The *IP*(*f*) is calculated using the Algorithm 5. This item is a periodic item because $\frac{IP(f)}{sup(f)+1} \geq minPR$ (line 3 in Algorithm 4). Therefore, we construct the prefix-tree for '*f*', i.e., PT_f as shown in Fig. 6(*a*). There are five items '*a, b, c, d*' and '*e*' in PT_f. Among them only the item '*e*' has *IP* value greater than or equal to $minPR \times (minSup + 1)$ threshold. Therefore, the conditional tree CT_f from PT_f is constructed with only one item '*e*', as shown in Fig. 6(*b*). The *ts*-list of '*e*' in CT_f generates TS^{ef}. The *periodic-ratio* of '*ef*' is measured using Algorithm 6. As $sup(ef) \geq minSup$ and $PR(ef) \geq minPR$, '*ef*' will be generated as a partial periodic-frequent pattern (lines 4 to 8 in Algorithm 4). A similar process is repeated for the other items in the GPF-list. Next, '*f*' is pruned from the original GPF-tree and its *ts*-lists are pushed to its parent nodes (line 9 in Algorithm 4). Fig. 6(*c*) shows the original GPF-tree after removing the item '*f*'. All the above processes are once again repeated until the GPF-list $\neq \emptyset$.

Table 3
Mining GPF-tree using conditional pattern bases. The terms 'SI' and 'PPFP' represent 'suffix item' and 'partial periodic-frequent pattern,' respectively.

SI	Conditional pattern base	Conditional GPF-tree	PPFP
f	{*abc*: 11}, {*cd*: 8}, {*cde*: 10} {*ce*: 13}	⟨*e*: 10, 11, 13⟩	{*ef*: 3, 0.75 }
e	{*ab*: 7, 11}, {*cd*: 10}, {*d*: 3} {*c*: 13}	-	-
d	{*acb*: 4, 6, 12}, {*c*: 8}	⟨*c*: 4, 6, 7, 10, 12⟩	{*cd*: 5, 0.83}
b	{*ac*: 4, 6, 12}, {*a*: 2, 7, 9, 11}	⟨*a*: 2, 4, 6, 7, 9, 11, 12⟩	{*ab*: 7, 1}
c	{*a*: 4, 6, 12}	-	-

Table 3 shows the complete steps involved in mining GPF-tree using conditional pattern bases. The formats used to represent *conditional pattern base*, *conditional GPF-tree* and *partial periodic-frequent patterns* are {*nodes*: *ts-list*}, ⟨*nodes*: *ts-list*⟩ and {*pattern*: *support*, *periodic-ratio*}, respectively.

The above bottom-up mining technique on a support descending GPF-tree is efficient, because it shrinks the search space dramatically as the mining process progresses. The correctness of GPF-growth is shown in Lemma 8.

Lemma 8. *Let α be a pattern in GPF-tree. Let minSup, maxPer and minPR be the user-defined minimum support, maximum period and minimum periodic-ratio, respectively. Let B be the α-conditional pattern base, and β be an item in B. Let TS^{β} be an array of timestamps containing β in B. If α is a candidate pattern, $TS^{\beta}.length \geq minSup$ and $PR(TS^{\beta}) \geq minPR$, then $< \alpha\beta >$ is a partial periodic-frequent pattern.*

Proof. According to the definition of conditional pattern base and compact GPF-tree, each subset in *B* occurs under the condition of the occurrence of α in the transactional database. If an item β in *B* has the timestamps of ts_i, ts_j and so on, then β appears with α at the timestamps of ts_i, ts_j and so on. Thus, TS^{β} in *B* is same as $TS^{\alpha\beta}$. From the definition of partial periodic-frequent pattern, if $TS^{\beta}.length \geq minSup$ and $PR(TS^{\beta}) \geq minPR$, then $< \alpha\beta >$ is a partial periodic-frequent pattern. Hence proved. □

6. Experimental results

This section evaluates the proposed model against the basic model of periodic-frequent patterns (Tanbeer et al., 2009). We show that the proposed model discovers useful information pertaining to both full and partial periodically occurring frequent patterns. We also show that the proposed GPF-growth is runtime efficient and highly scalable as well.

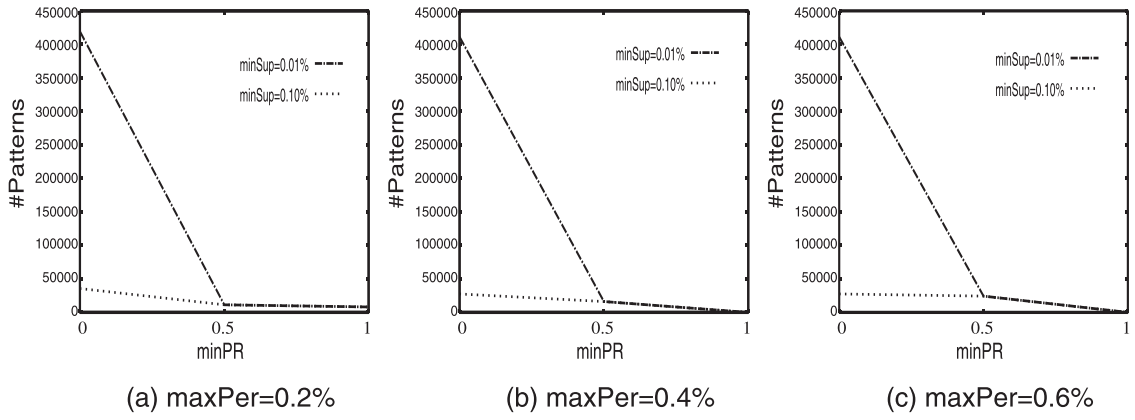


Fig. 7. Number of partial periodic-frequent patterns generated from T10I4D100K database at different *minSup, maxPrd* and *minPR* values.

Table 4
Database statistics. The terms, T_{min}, T_{avg} and T_{max}, represent the minimum, average and maximum number of items within a transaction, respectively.

Database	T_{min}	T_{avg}	T_{max}	Size	Items
T10I4D100K	1	10.1	29	100,000	870
T10I4D1000K	1	10.1	31	983,155	30,387
T25I6D1000K	1	24.9	55	999,960	1007
Retail	1	10.3	76	88,162	16,470
FAA-accidents	3	8.9	9	98,864	9290
Kosarak	1	8.1	2498	990,002	41,270

6.1. Experimental setup

The algorithms, PF-growth and GPF-growth, are written in GNU C++ and run with Ubuntu 14.4 on a 2.66 GHz machine with 8 GB of memory. The experiments have been conducted using both synthetic (**T10I4D100K, T10I4D1000K**, and **T25I6D1000K**) and real-world (**Retail** and **FAA-accidents**) databases. The synthetic databases that are used in our experiments are generated by using the IBM data generator (Agrawal et al., 1993). This data generator is widely used for evaluating the association rule mining algorithms. The **Retail** database contains 5 months of market basket data collected from an anonymous Belgian retail store (Brijs et al., 2000). The **Kosarak** database contains click-stream data of a Hungarian on-line news portal. The **Retail** and **Kosarak** databases are available at Frequent Itemset Mining repository (Goethals, 2005). The **FAA-accidents** database is constructed from the aircrafts accidents data recorded by Federal Aviation Authority (FAA) from 1-January-1978 to 31-December-2014 (FAA, 2015). The raw data collected by FAA contains both numerical and categorical attributes. For our experiments, we have considered only categorical attributes, namely 'local event date,' 'event city,' 'event state,' 'event airport,' 'event type,' 'aircraft damage,' 'flight phase,' 'aircraft make,' 'aircraft model,' 'operator,' 'primary flight type,' 'flight conduct code,' 'flight plan filed code' and 'PIC certificate type.' The missing values for these attributes are ignored while creating this database. The statistical details of these databases are shown in Table 4.

6.2. Generation of partial periodic-frequent patterns

Figs. 7, 8 and 9 show the number of partial periodic-frequent patterns generated in T10I4D100K, Retail and FAA-accidents databases, respectively. The partial periodic-frequent patterns generated at $minPR = 0$ represent the frequent patterns (independent of the *maxPer* values). The partial periodic-frequent patterns generated at $minPR = 1$ represent the full periodic-frequent patterns generated from the basic model (Tanbeer et al., 2009) when $maxPeriodicity = maxPer$. The following observations can be drawn from these figures:

- The increase in *minSup* has decreased the number of partial periodic-frequent patterns. The reason is increase in *minSup* has increased the number of transactions in which a pattern has to appear to be a frequent pattern.
- The increase in *minPR* has decreased the number of partial periodic-frequent patterns. It is because the increase in *minPR* has increased the number of cyclic repetitions necessary for a pattern to be a partial periodic-frequent pattern.
- The increase in *maxPer* has increased the number of partial periodic-frequent patterns. The reason is increase in *maxPer* has increased the inter-arrival time within which a pattern has to appear in a database.
- Very few full periodic-frequent patterns are being generated at $minPR = 1$. The reason is that it is difficult for the patterns to exhibit complete cyclic repetitions in the entire database. Most of the full periodic-frequent patterns discovered in all of these databases are singleton patterns (or items). Setting a very long *maxPer* value has enabled us to discover full periodic-frequent patterns containing more than one item. However, this long *maxPer* value has also generated many sporadically appearing frequent patterns as periodic-frequent patterns.

Fig. 10(a), (b) and (c) show the distribution of partial periodic-frequent patterns with respect to their itemset lengths. The *minSup* and *maxPer* values are set at 0.1% and 0.2%, respectively. The partial periodic-frequent patterns generated at $minPR = 1$ represent the full periodic-frequent patterns. The following two observations can be drawn from these figures:

- A strict constraint that a pattern must exhibit complete cyclic repetitions in the entire database often results in generating very few periodic-frequent patterns, and most of them are singleton patterns. Relaxing this constraint may facilitate the user in finding ample number of frequent patterns that have exhibited partial cyclic repetitions in the data.
- When we relax the strict constraint that a pattern must exhibit complete cyclic repetitions in the entire database too much (i.e., very low *minPR* values), we will discovering almost all frequent patterns as periodic-frequent patterns. Thus, it is necessary not to relax the constraint too much. We recommend *minPR* values greater than or equal to 50%. The reason is it is difficult to call those frequent patterns that have less than 50% of *periods* within *maxPer* as partial periodic-frequent patterns.

Table 5 shows some of the interesting patterns discovered in FAA-accidents at $minSup = 0.3\%$, $maxPer = 0.6\%$ ($= 81$ *days*) and $minPR = 0.8\%$. The first pattern reveals the useful information that

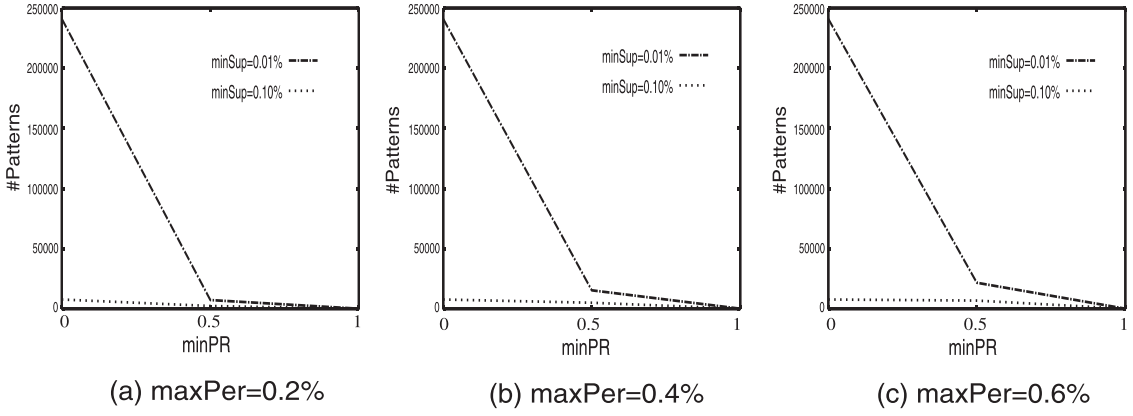


Fig. 8. Number of partial periodic-frequent patterns generated from Retail database at different *minSup, maxPrd* and *minPR* values.

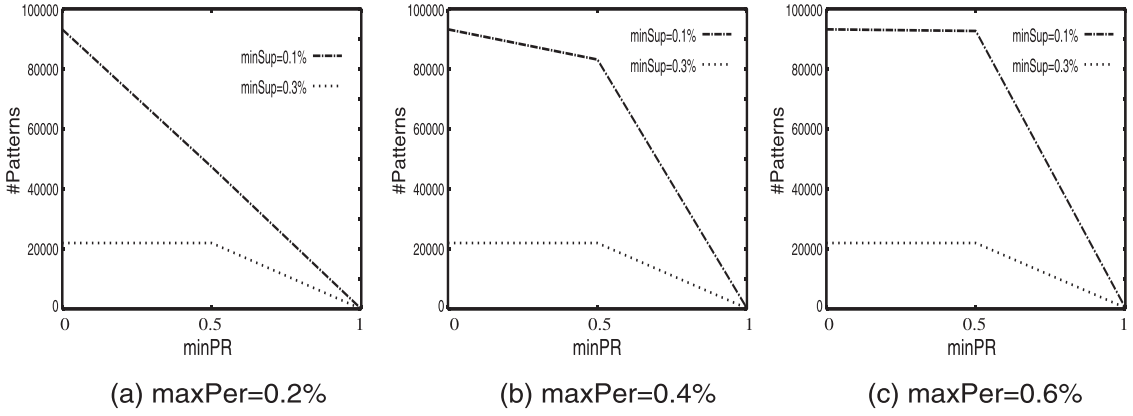


Fig. 9. Number of partial periodic-frequent patterns generated from FAA-accidents database at different *minSup, maxPrd* and *minPR* values.

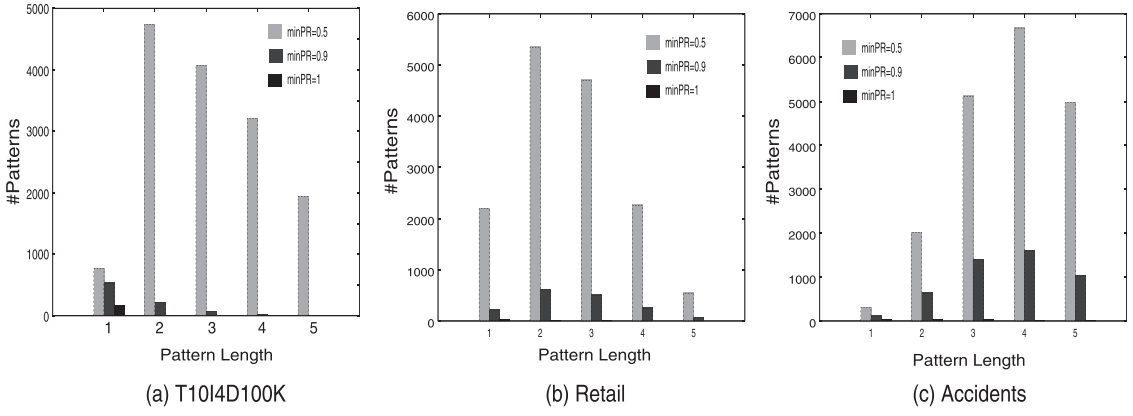


Fig. 10. Number of partial periodic-frequent patterns generated from FAA-accidents database at different *minSup, maxPrd* and *minPR* values.

with an inter-arrival time of 81 days, 515 (=537*0.964) personal aircrafts driven by private pilots have gone through substantial damages while performing general operating rules. The second pattern provides the information that 290 (=316*0.92) aircrafts have witnessed substantial damages when following general operating rules and flight phase is level-off touchdown. The third pattern provides the information that at least once in every 81 days, 290 (=323*.91) aircrafts have gone through minor damages while following instrument flight rules. The final patterns provides the information that at least once in every 81 days, 298 (=304*0.9) Cessna personal aircrafts driven by a private pilot have gone through minor damages when following general operating rules.

6.3. Runtime requirements of GPF-growth

Fig. 11(a), (b) and (c) show the graph of *minPR* versus runtime requirements of GPF-growth at different *minPer* values in T10I4D100K, Retail and FAA-accidents databases, respectively. The *minSup* values used in T10I4D100K, Retail and FAA-accidents databases are 0.01%, 0.01% and 0.03%, respectively. It can be observed from these figures that changes in *maxPer* and *minPR* values show a similar effect on runtime requirements as in the generation of partial periodic-frequent patterns.

6.4. Scalability results

We study the scalability of GPF-growth on memory and runtime requirements by varying the number of transactions in

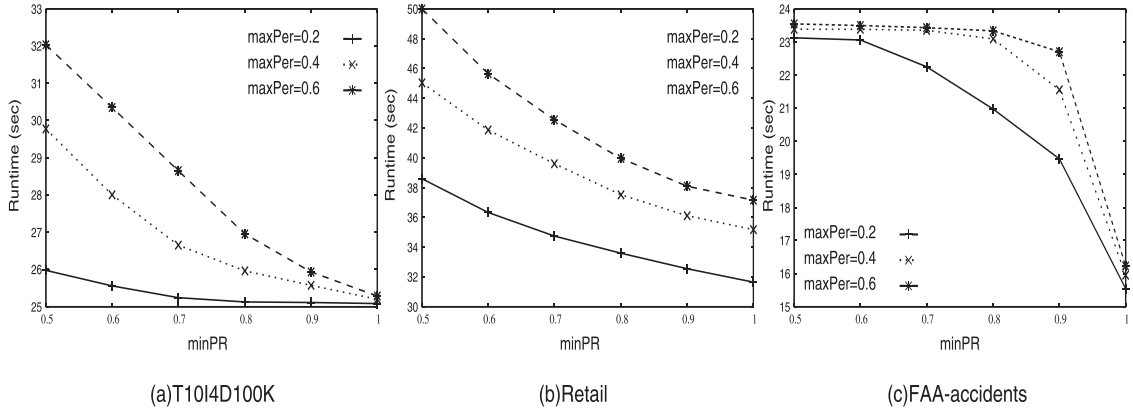


Fig. 11. Runtime requirements of GPF-growth at various *maxPer* and *minPR* values.

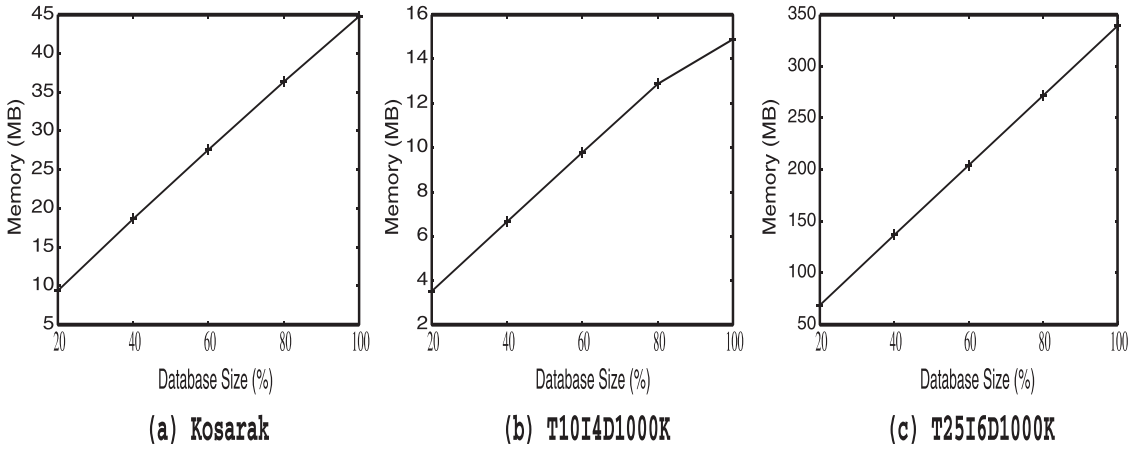


Fig. 12. Memory requirements of GPF-growth with increase in database size.

Table 5
Some of the interesting patterns discovered in FAA-accidents database.

S. No.	Pattern	*Support*	*Periodic-ratio*
1	{GENERAL-OPERATING-RULES, PERSONAL, PRIVATE-PILOT, SUBSTANTIAL}	537	0.964
2	{GENERAL-OPERATING-RULES, LEVEL-OFF-TOUCHDOWN, SUBSTANTIAL}	316	0.92
3	{MINOR, INSTRUMENT-FLIGHT-RULES, AIRLINE-TRANSPORT, ROLL-OUT-(FIXED-WING), AIR-CARRIER/COMMERCIAL}	323	0.91
4	{GENERAL-OPERATING-RULES, MINOR, PERSONAL, PRIVATE-PILOT, CESSNA, INSTRUMENT-FLIGHT-RULES}	304	0.90

a database. We use Kosarak, T10I4D1000K and T25I6D1000K databases for this experiment. We have divided each of these databases into five portions with 20% of transactions in each portion. Then we investigated the performance of GPF-growth after accumulating each portion with previous parts with performing partial periodic-frequent pattern mining each time. Table 6 shows the *minSup, maxPer* and *minPR* values used in various databases.

Fig. 12(a) (b) and (c) show the graphs of memory requirements versus database size of GPF-growth in Kosarak, T10I4D1000K and T25I6D1000K databases, respectively. Fig. 13(a) (b) and (c) show the graphs of runtime requirements versus database size of GPF-growth in T10I4D1000K and T25I6D1000K databases, respectively. It is clear from both the graphs that as the database size increases,

Table 6
The *minSup, maxPer* and *minPR* values used in scalability experiment.

Database	*minSup*	*maxPer*	*minPR*
Kosarak	0.2%	10%	0.3%
T10I4D1000K	0.1%	10%	0.3%
T25I6D1000K	0.2%	10%	0.3%

the memory and runtime requirements of GPF-growth also gets increased. However, GPF-growth shows stable performance of about linear increase of memory and runtime consumption with respect to the database size. Therefore, it can be observed from the scalability test that GPF-growth can mine the partial periodic-frequent patterns over large datasets and distinct items with considerable amount of memory and runtime.

7. Conclusions and future work

This paper proposes a model to discover partial periodic-frequent patterns in a temporally ordered transactional database. A new interestingness measure, *periodic-ratio*, has been introduced to determine the partial periodic interestingness of a pattern in time dimension. A pruning technique to reduce the search space has been discussed to discover the partial periodic-frequent patterns effectively. A pattern-growth algorithm has been proposed to discover all partial periodic-frequent patterns. Experimental results demonstrate that the proposed model can discover useful information and GPF-growth is efficient.

In this paper, we have not taken into account the change in periodic behavior of a pattern due to the affect of noise. As a

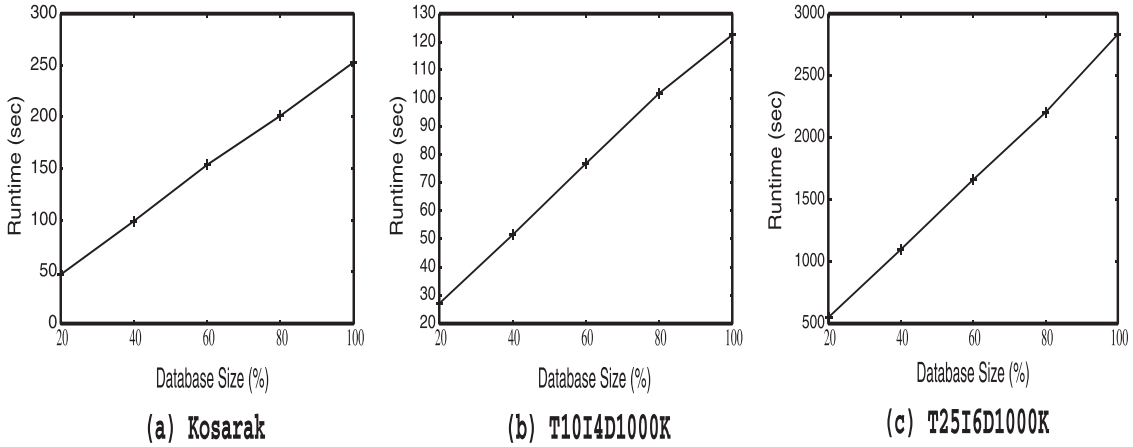


Fig. 13. Runtime requirements of GPF-growth with increase in database size.

part of future work, it is interesting to investigate methodologies to discover partial periodic-frequent patterns in noisy real-world databases. The existing state-of-the-art association rule mining classifiers take into account only the frequency dimension of a pattern. It is interesting to investigate how the information discovered by the proposed patterns can be used in developing efficient classifiers.

References

Adomavicius, G., Tuzhilin, A., 2001. Expert-driven validation of rule-based user models in personalization applications. Data Min. Knowl. Discov. 5 (1–2), 33–58.

Adomavicius, G., Tuzhilin, A., 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng. 17 (6), 734–749.

Agrawal, R., Imieliński, T., Swami, A., 1993. Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216.

Amphawan, K., Lenca, P., Surarerks, A., 2009. Mining top-k periodic-frequent pattern from transactional databases without support threshold. In: Advances in Information Technology, pp. 18–29.

Aref, W.G., Elfeky, M.G., Elmagarmid, A.K., 2004. Incremental, online, and merge mining of partial periodic patterns in time-series databases. IEEE TKDE 16 (3), 332–342.

Berberidis, C., Vlahavas, I., Aref, W., Atallah, M., Elmagarmid, A., 2002. On the discovery of weak periodicities in large time series. In: PKDD, pp. 51–61.

Brijs, T., Goethals, B., Swinnen, G., Vanhoof, K., Wets, G., 2000. A data mining framework for optimal product selection in retail supermarket data: the generalized profset model. In: KDD, pp. 300–304.

Cao, H., Cheung, D., Mamoulis, N., 2004. Discovering partial periodic patterns in discrete data sequences. In: Advances in Knowledge Discovery and Data Mining, 3056, pp. 653–658.

Chen, S.-S., Huang, T.C.-K., Lin, Z.-M., 2011. New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports. J. Syst. Softw. 84 (10), 1638–1651.

Dong, G., Li, J., 1999. Efficient mining of emerging patterns: Discovering trends and differences. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 43–52.

Esling, P., Agon, C., 2012. Time-series data mining. ACM Comput. Surv. 45 (1), 12:1–12:34.

FAA, 2015. Federal Aviation Authority. Available at http://www.asias.faa.gov/pls/apex/f?p=100:1:7565565412795.

Goethals, B., 2005. Frequent Itemset Mining repository. Available at http://fimi.ua.ac.be/data/.

Gouda, K., Zaki, M.J., 2001. Efficiently mining maximal frequent itemsets. In: Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 163–170.

Han, J., Cheng, H., Xin, D., Yan, X., 2007. Frequent pattern mining: current status and future directions. DMKD 14 (1).

Han, J., Dong, G., Yin, Y., 1999. Efficient mining of partial periodic patterns in time series database. In: ICDE, pp. 106–115.

Han, J., Gong, W., Yin, Y., 1998. Mining segment-wise periodic patterns in time-related databases. In: KDD, pp. 214–218.

Han, J., Pei, J., Yin, Y., Mao, R., 2004. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Discov. 8 (1), 53–87.

Han, J., Wang, J., Lu, Y., Tzvetkov, P., 2002. Mining top.k frequent closed patterns without minimum support. In: Proceedings of the 2002 IEEE International Conference on Data Mining, pp. 211–218.

Kiran, R.U., Kitsuregawa, M., 2014. Novel techniques to reduce search space in periodic-frequent pattern mining. In: DASFAA (2), pp. 377–391.

Kiran, R.U., Kitsuregawa, M., Reddy, P.K., 2016. Efficient discovery of periodic-frequent patterns in very large databases. J. Syst. Soft. 112, 110–121.

Kiran, R.U., Reddy, P.K., 2010. Towards efficient mining of periodic-frequent patterns in transactional databases. In: DEXA (2), pp. 194–208.

Kiran, R.U., Reddy, P.K., 2011. An alternative interestingness measure for mining periodic-frequent patterns. In: DASFAA (1), pp. 183–192.

Kiran, R.U., Shang, H., Toyoda, M., Kitsuregawa, M., 2015. Discovering recurring patterns in time series. EDBT, pp. 97–108.

Laxman, S., Sastry, P.S., Unnikrishnan, K.P., 2007. A fast algorithm for finding frequent episodes in event streams. In: KDD. Association for Computing Machinery, Inc., pp. 410–419.

Ma, S., Hellerstein, J., 2001. Mining partially periodic event patterns with unknown periods. In: ICDE, pp. 205–214.

Mannila, H., 1997. Methods and problems in data mining. In: The International Conference on Database Theory, pp. 41–55.

Mohammad, Y.F.O., Nishida, T., 2013. Approximately recurring motif discovery using shift density estimation. In: IEA/AIE, pp. 141–150.

Mooney, C.H., Roddick, J.F., 2013. Sequential pattern mining – approaches and algorithms. ACM Comput. Surv. 45 (2), 19:1–19:39.

Oates, T., 2002. Peruse: an unsupervised algorithm for finding recurrig patterns in time series. In: ICDM, pp. 330–337.

Özden, B., Ramaswamy, S., Silberschatz, A., 1998. Cyclic association rules. In: ICDE, pp. 412–421.

Pei, J., Han, J., Lakshmanan, L.V., 2004. Pushing convertible constraints in frequent itemset mining. Data Min. Knowl. Discov. 8, 227–252.

Stormer, H., 2007. Improving e-commerce recommender systems by the identification of seasonal products. In: Twenty second Conference on Artificial Intelligence, pp. 92–99.

Surana, A., Kiran, R.U., Reddy, P.K., 2011. An efficient approach to mine periodic-frequent patterns in transactional databases. In: PAKDD Workshops, pp. 254–266.

Swamy, M.K., Reddy, P.K., Srivastava, S., 2014. Extracting diverse patterns with unbalanced concept hierarchy. In: Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13–16, 2014. Proceedings, Part I, pp. 15–27.

Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y.K., 2009. Discovering periodic-frequent patterns in transactional databases. In: PAKDD, pp. 242–253.

Yang, J., Wang, W., Yu, P.S., 2003. Mining asynchronous periodic patterns in time series data. IEEE Trans. Knowl. Data Eng. 613–628.

Yao, H., Hamilton, H.J., Butz, C.J., 2004. A foundational approach to mining itemset utilities from databases. In: SDM. SIAM, pp. 482–486.

Zaki, M.J., Hsiao, C.-J., 2005. Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Trans. Knowl. Data Eng. 17 (4), 462–478.

Zhang, W., Yoshida, T., Tang, X., Wang, Q., 2010. Text clustering using frequent itemsets. Knowl. Based Syst. 23 (5), 379–388.