Discovering Quasi-Periodic-Frequent Patterns in Transactional Databases

R. Uday Kiran and Masaru Kitsuregawa

Institute of Industrial Science,
The University of Tokyo, Tokyo, Japan.
{uday_rage, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. Periodic-frequent patterns are an important class of user-interest-based frequent patterns that exist in a transactional database. A frequent pattern can be said *periodic-frequent* if it appears periodically throughout the database. We have observed that it is difficult to mine periodic-frequent patterns in very large databases. The reason is that the occurrence behavior of the patterns can vary over a period of time causing periodically occurring patterns to be non-periodic and/or vice-versa. We call this problem as the "intermittence problem." Furthermore, in some of the real-world applications, the users may be interested in only those frequent patterns that might have appeared almost periodically throughout the database. With this motivation, we relax the constraint that a pattern must appear periodically throughout the database, and introduce a new class of user-interest-based frequent patterns, called quasi-periodic-frequent patterns. Informally, a frequent pattern is said to be *quasi-periodic-frequent* if most of its occurrences are periodic in a database. We propose a model and a pattern-growth algorithm to discover these patterns. We also introduce three pruning techniques to reduce the computational cost of mining the patterns. Experimental results show that the proposed patterns can provide useful information and the proposed algorithm is efficient.

Key words: Data mining, knowledge discovery in databases, frequent patterns and periodic behavior.

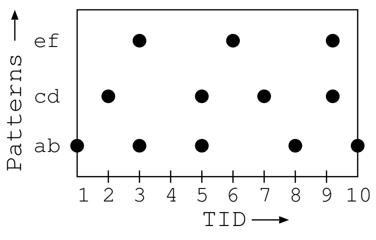
1 Introduction

Periodic-frequent patterns [3] are an important class of regularities that exist within a transactional database. In many real-world applications, periodic-frequent patterns provide useful information pertaining to the patterns that are not only occurring frequently, but also appearing at regular intervals specified by the user in a database. For example, in a retail market, among all frequently sold products, the user may be interested only in the regularly sold products compared to the rest. Periodic-frequent pattern mining on a market basket data can provide such information to the users. The basic model of periodic-frequent patterns is as follows [3].

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of items. A set $X = \{i_1, \cdots, i_k\} \subseteq I$, where $1 \leq k \leq n$ is called a **pattern** (or an itemset). A pattern containing k number of items is called **k-pattern**. A transaction $t = (tid, Y)$ is a tuple, where tid represents a transaction-id (or a timestamp) and Y is a pattern. A transactional database T over I is a set of transactions, $T = \{t_1, \cdots, t_m\}$, $m = |T|$, where $|T|$ is the size of T in total number of transactions. If $X \subseteq Y$, it is said that t contains X or X occurs in t and such transaction-id is denoted as t_j^X, $j \in [1,m]$. Let $T^X = \{t_k^X, \cdots, t_l^X\} \subseteq T$, where $k \leq l$ and $k,l \in [1,m]$ be the ordered set of transactions in which pattern X has occurred. Let t_j^X and t_{j+1}^X, where $j \in [k,(l-1)]$ be two consecutive transactions in T^X. The number of transactions (or time difference) between t_{j+1}^X and t_j^X can be defined as a ***period*** of X, say p_a^X. That is, $p_a^X = t_{j+1}^X - t_j^X$. Let $P^X = \{p_1^X, p_2^X, \cdots, p_r^X\}$, $r = |T^X| + 1$, be the complete set of periods for pattern X. The ***periodicity*** of X, denoted as $Per(X) = max(p_1^X, p_2^X, \cdots, p_r^X)$. The ***support*** of X, denoted as $S(X) = |T^X|$. The pattern X is said to be periodic-frequent if $S(X) \geq minsup$ and $Per(X) \leq maxprd$. The *minsup* and *maxprd* are user-specified minimum support and maximum periodicity constraints, respectively. Both periodicity and support of a pattern can be described in percentage of $|T|$.

TID	Items	TID	Items
1	a,b	6	e,f
2	c,d	7	c,d
3	a,b,e,f	8	a,b
4	b,e	9	c,d,e,f
5	a,b,c,d	10	a,b

a) Transactional database



b) Pattern occurrences

Fig. 1: Running example.

Example 1. Consider the transactional database shown in Fig. 1(a). Each transaction in this database is uniquely identifiable with a transactional-id (*tid*). The *tid* of a transaction also represents the ordered sequence of transactions based on a particular time stamp. **In this database, let us consider a sub-transactional database consisting of the first five transactions.** The set of items, $I = \{a,b,c,d,e,f\}$. The set of '*a*' and '*b*' i.e., $\{a,b\}$ is a pattern. For the purpose of simplicity, we represent this pattern as '*ab*'. This pattern occurs in *tids* 1,3 and 5. Therefore, $T^{ab} = \{1,3,5\}$. The support of '*ab*', i.e., $S(ab) = |T^{ab}| = 3$. The periods for this pattern are $p_1^{ab} = 1(= 1 - t_i)$, $p_2^{ab} = 2(= 3-1)$, $p_3^{ab} = 2(= 5-3)$ and $p_4^{ab} = 0(= t_l - 5)$, where $t_i = 0$ represents the initial transaction and $t_l = 5$ represents the last transaction in the sub-transactional database. The periodicity of '*ab*', $Per(ab) = maximum(1,2,2,0) = 2$. If the user-specified $minsup = 2$ and $maxprd = 2$, the pattern '*ab*' is a periodic-frequent pattern because $S(ab) \geq minsup$ and $Per(ab) \leq maxprd$.

The time duration of a database refers to the difference between the starting and ending timestamps of the transactions within it. Time duration of a database is independent to its size (i.e., number of transactions within it) because arrival rates of the transactions into a database can vary with respect to time. Mining periodic-frequent patterns is difficult if a database is composed over a very long time duration. It is because items' occurrence behavior can vary drastically causing many periodically occurring frequent patterns to be non-periodic and/or vice-versa.

Example 2. In a shopping mall, if we consider the transactions happened only during winter season, we can find interesting periodic-frequent patterns pertaining to woolen wears. However, if we consider the transactions of the whole year, finding periodic-frequent patterns pertaining to woolen wears is very difficult, because, they are not often purchased during summer season.

We call this problem as the "*intermittence problem*." A method to address this problem involves mining the patterns with a high *maxprd* value. However, such an approach may generate sporadically occurring frequent patterns as the periodic-frequent patterns.

Example 3. Consider the complete transactional database shown in Fig. 1(a).The pattern '*ab*' occurs in *tids* of $1,3,5,8$ and 10. Therefore, $T^{ab} = \{1,3,5,8,10\}$. The support of '*ab*,' i.e., $S(ab) = |T^{ab}| = 5$. The periods for this pattern are $p_1^{ab} = 1(= 1-t_i)$, $p_2^{ab} = 2(= 3-1)$, $p_3^{ab} = 2(= 5-3)$, $p_4^{ab} = 3(= 8-5)$, $p_5^{ab} = 2(= 10-8)$ and $p_6^{a} = 0(= t_l - 10)$. The set of periods for the pattern '*ab*', $P^{ab} = \{1,2,2,3,2,0\}$. The *periodicity* of '*ab*', i.e., $per(ab) = max(1,2,2,3,2,0) = 3$. If the user-specified $maxprd = 2$, then '*ab*' is a non-periodic frequent pattern because $Per(ab) > maxprd$. We can disocver the pattern '*ab*' as a periodic-frequent pattern by setting a high *maxprd* value, say $maxprd = 3$. However, this high *maxprd* value can result in generating the sporadically occurring frequent pattern, say 'ef', as a periodic-frequent pattern because $Per(ef) = 3$ (see Fig. 1(b)).

With this motivation, we have investigated the interestingness of the frequent patterns with respect to their proportion of periodic occurrences in the database. During this investigation, we have observed that the frequent patterns that were almost occurring periodically in a database can also provide useful knowledge to the users. Based on this observation, we relax the constraint that a frequent pattern must appear periodically throughout the database and introduce a new class of user-interest-based frequent patterns, called quasi-periodic-frequent patterns. Informally, a frequent pattern is said to be *quasi-periodic-frequent* if most of its occurrences are periodic in a transactional database. Please note that the proposed patterns are tolerant to the intermittence problem. The contributions of this paper are as follows:

- An alternative interestingness measure, called periodic-ratio, has been proposed to assess the periodic interestingness of a pattern.
- The quasi-periodic-frequent patterns do not satisfy the *downward closure property*. That is, all non-empty subsets of a quasi-periodic-frequent pattern are not quasi-periodic-frequent. This increases the search space, which in turn increases the computational cost of mining the patterns. We introduce three pruning techniques to reduce the computational cost of mining the patterns.
- A pattern-growth algorithm, called Quasi-Periodic-Frequent pattern-growth (QPF-growth), has been proposed to discover the patterns.
- Experimental results show that the proposed patterns can provide useful information, and the QPF-growth is efficient.

The rest of the paper is organized as follows. Section 2 summarizes the efforts made in the literature to discover periodic-frequent patterns. In Section 3, we present the conceptual model of quasi-periodic-frequent patterns. In Section 4, we discuss the basic idea to reduce the computational cost of mining the patterns and describe the QPF-growth algorithm. Experimental results are presented in Section 5. Section 6 concludes with future research directions.

2 Related Work

The periodic behavior of patterns has been widely studied in various domains as temporal patterns [?], cyclic association rules [?] and periodic behavior of moving objects [?]. Since real-life patterns are generally imperfect, Han et al. [?] have investigated the concept of partial periodic behavior of patterns in time series databases with the assumption that period lengths are known in advance. The **partial periodic patterns** specify the behavior of time series at some but not at all points in time. Sheng and Joseph [?] have investigated the same with unknown periods. Walid et al. [?] have extended the Han's work in [?] to incremental mining of partial periodic patterns in time series databases. All of these approaches do not consider the effect of noise on the partial periodic behavior of patterns. Therefore, Yan et al. [?] have extended [?] by introducing information theory concepts to address the effect of noise on periodic behavior of patterns in time series. Sheng et al. [?] have introduced the concept of density to discover partial periodic patterns that may have occurred within small segments of time series data. The problem of mining frequently occurring periodic patterns with a gap has been investigated in sequence databases [?,?,?]. Although these works are closely related to our work, they cannot be directly applied for finding ppf-patterns from a transactional database because of two reasons. First, they consider either time series or sequential data; second, they do not consider the support threshold which needs to be satisfied by all frequent patterns.

Recently, the periodic occurrences of frequent patterns in a transactional database has been studied in the literature to discover periodic-frequent patterns [?,?,?]. We have observed that these approaches cannot be extended to discover ppf-patterns. It is because the ppf-patterns do not satisfy the anti-monotonic property.

The concept of finding frequent patterns in data streams uses the temporal occurrence of a pattern in a database [?]. However, it has to be noted that only the occurrence frequency was used to discover frequent patterns.

In [3], temporal occurrence behavior of the patterns in a transactional database has been used as an interestingness measure to identify a class of user-interest-based frequent patterns, called periodic-frequent patterns. A pattern-growth approach based on a tree structure, called Periodic-Frequent tree (PF-tree) has also been discussed for mining periodic-frequent patterns. In this paper, we refer to this approach as Periodic-Frequent growth (PF-growth).

A rare periodic-frequent pattern is a periodic-frequent pattern consisting of rare items i.e., items having low frequencies. It is difficult to mine rare periodic-frequent patterns with a single *minsup* constraint because low *minsup* can cause combinatorial explosion, producing too many periodic-frequent patterns in which some of them can be uninteresting. Hence, efforts have been made in [4, 5] to mine periodic-frequent patterns using multiple *minsup* constraints, where *minsup* of a pattern is represented with the *minimum item supports* of its items.

The proposed quasi-periodic-frequent patterns are different from the periodic-frequent patterns [3–5] in the following ways. First, quasi-periodic-frequent patterns need not appear periodically throughout the database. Therefore, the periodicity measure does not exist for the quasi-periodic-frequent patterns. Second, periodic-frequent patterns follow *downward closure property*, whereas quasi-periodic-frequent patterns do not. Third, every periodic-frequent pattern is a quasi-periodic-frequent pattern; however, vice versa is not true. In this paper, we are not focusing on mining quasi-periodic-frequent patterns involving rare items.

3 Proposed Model

In this section, we describe the model of quasi-periodic-frequent pattern mining using the periodic-frequent pattern mining model discussed in Section 1. We also introduce the basic notations and definitions in this regard.

Definition 1. Maximum period (*maxperiod*): *It is a user-specified constraint that describes the interestingness of a period. A period of X, $p_a^X \in P^X$ is interesting if $p_a^X \leq maxperiod$.*

Example 4. Continuing with the Example 3, if the user-specified $maxperiod = 2$, then p_1^{ab}, p_2^{ab}, p_3^{ab}, p_5^{ab} and p_6^{ab} are the interesting periods because their values are less than or equal to the user-specified $maxperiod$ constraint.

Definition 2. ***The periodic-ratio of a pattern X:*** *Let $IP^X = \{p_i^X, \cdots, p_j^X\} \subseteq P^X$, where $1 \leq i \leq j \leq r$ be the set of interesting periods for pattern X. The* periodic-ratio *of pattern X, denoted as $pr(X) = \frac{|IP^X|}{|P^X|}$. This measure captures the proportion of periods in which pattern X has appeared periodically in a database.*

For a pattern X, $pr(X) \in [0,1]$. If $pr(X) = 0$, it means X has not appeared periodically anywhere in the transactional database. If $pr(X) = 1$, it means X has appeared periodically throughout the transactional database. In other words, X is a periodic-frequent pattern. Thus, the proposed model generalizes the existing model of periodic-frequent patterns. The proportion of non-periodic occurrences of a pattern X in the transactional database is given by $1 - ps(X)$. Please note that the *periodic-ratio* of a pattern X can be measured in percentage of P^X.

Example 5. Continuing with the example 4, $IP^{ab} = \{p_1^{ab}, p_2^{ab}, p_4^{ab}\}$. Therefore, the periodic-ratio of '*ab*,' i.e., $pr(ab) = \frac{|IP^{ab}|}{|P^{ab}|} = \frac{3}{4} = 0.75$ $(= 75\%)$. The periodic-ratio of '*ab*' says that 0.75 proportion of its occurrences are periodic and 0.25 (=1-0.75) proportion of its occurrences are non-periodic within the database.

Definition 3. Minimum periodic-ratio (*minpr*): *It is a user-specified constraint that describes the interestingness of a pattern with respect to its periodic occurrences in a database. The pattern X is periodically interesting if $pr(X) \geq minpr$.*

Example 6. Continuing with the example 5, if the user-specified $minpr = 0.75$, then the pattern '*ab*' is interesting with respect to its periodic occurrence behaviour, because, $pr(ab) \geq minpr$.

Definition 4. ***Quasi-periodic-frequent pattern:*** *The pattern X is quasi-periodic-frequent if $S(X) \geq minsup$ and $pr(X) \geq minpr$. For a quasi-periodic-frequent pattern X, if $S(X) = a$ and $pr(X) = b$, then it is described as shown in Equation 1.*

$$X \; [support = a, periodic\text{-}ratio = b] \tag{1}$$

Example 7. Continuing with the example 6, if the user-specified $minsup = 3$, $maxperiod = 2$ and $minpr = 0.8$, then 'ab' is a quasi-periodic-frequent pattern because $S(ab) \geq minsup$ and $pr(ab) \geq minpr$. This pattern is described as follows:

$$ab \; [support = 5, \; periodic\text{-}ratio = 0.75]$$

The quasi-periodic-frequent patterns mined using the proposed model do not satisfy *downward closure property* (see, Lemma 1). That is, all non-empty subsets of a quasi-periodic-frequent pattern may not be quasi-periodic-frequent. The correctness of our argument is based on the Properties 1 and 2 and shown in Lemma 1.

Property 1. In the proposed model, the total number of periods for X i.e., $|P^X| = S(X) + 1 = |T^X| + 1$.

Property 2. Let X and Y be the two patterns in a transactional database. If $X \subset Y$, then $|P^X| \geq |P^Y|$ and $|IP^X| \geq |IP^Y|$ because $T^X \supseteq T^Y$.

Lemma 1. *In a transactional database T, the quasi-periodic-frequent patterns mined using minsup, maxperiod and minpr constraints do not satisfy* downward closure property.

Proof. Let $Y = \{i_a, \cdots, i_b\}$, where $1 \leq a \leq b \leq n$ be a quasi-periodic-frequent pattern with $S(Y) = minsup$ and $periodic\text{-}ratio(Y) = minpr$. Let Y be an another pattern such that $X \subset Y$. From Property **??**, we derive $|P^X| \geq |P^Y|$ and $|IP^X| \geq |IP^Y|$. Considering the scenario where $|P^X| > |P^Y|$ and $|IP^X| = |IP^Y|$, we derive $periodic\text{-}ratio(X) < periodic\text{-}ratio(Y) (= minpr)$. Hence, X is not a quasi-periodic-frequent pattern. Hence proved.

Problem Definition. Given a transactional database T, minimum support ($minsup$), maximum period ($maxperiod$) and minimum periodic-ratio ($minpr$) constraints, the objective is to discover the complete set of quasi-periodic-frequent patterns in T that have support and periodic-ratio no less than the $minsup$ and $minpr$ constraints, respectively.

4 QPF-growth: Idea, Design, Construction and Mining

4.1 Basic Idea

We have observed two issues while mining quasi-periodic-frequent patterns in the transactional databases. Now, we discuss each of these issues and our approaches to address the same.

Issue 1: The quasi-periodic-frequent patterns do not satisfy the *downward closure property*. This increases the search space, which in turn increases the computational cost of mining the patterns. We introduce the following pruning techniques to reduce the computational cost of mining the patterns.

1. The *minsup* constraint satisfies the *downward closure property* [1]. Therefore, if a pattern X does not satisfy *minsup*, then X and its supersets can be pruned because they cannot generate any quasi-periodic-frequent pattern.
2. Another interesting idea is as follows. Every quasi-periodic-frequent pattern will have support greater than or equal to *minsup*. Hence, every quasi-periodic-frequent pattern will have at least $(minsup+1)$[1] number of periods (Property **??**). That is, for a pattern X, $|P^X| \geq (minsup+1)$. Therefore, if $\frac{|IP^X|}{(minsup+1)} < minpr$, then X cannot be a quasi-periodic-frequent pattern. If $Y \supset X$, then Y cannot be a quasi-periodic-frequent pattern because $|IP^Y| \leq |IP^X|$ and $|P^X| \geq |P^Y| \geq (minsup+1)$. Therefore, if $\frac{|IP^X|}{(minsup+1)} < minpr$, then X and all of its supersets can be pruned because none of them can generate a quasi-periodic-frequent pattern.

   Example 8. Let the user-specified *minsup*, *maxperiod* and *minpr* values for the transactional database shown in Fig. 1(a) be 3, 2 and 0.8, respectively. Every quasi-periodic-frequent pattern will have support greater than or equal 3. Thus, every quasi-periodic-frequent pattern will have at least $4(=3+1)$ number of periods (Property **??**). For item 'e', $T^{\{e\}} = \{3,4,6,9\}$, $P^{\{e\}} = \{3,1,2,3,1\}$ and $IP^{\{e\}} = \{1,2,1\}$. Therefore, $\frac{|IP^{\{e\}}|}{minsup+1} = \frac{3}{4} = 0.75 < minpr$. The item '$e$' cannot be quasi-periodic-frequent because *periodic-ratio*$(e) < minpr$ as $|P^{\{e\}}| \geq (minsup+1)$. For any superset of '$e$', say '$\{e,f\}$', to be quasi-periodic-frequent, it should also have at least 4 number of periods. Using Property **??**, we derive $T^{\{e,f\}} \subseteq T^{\{e\}}$, $|P^{\{e,f\}}| \leq |P^{\{e\}}|$ and $|IP^{\{e,f\}}| \leq |IP^{\{e\}}|$. Since $|IP^{\{e,f\}}| \leq |IP^{\{e\}}|$, it turns out that $\frac{|IP^{\{e,f\}}|}{minsup+1} \leq \frac{|IP^{\{e\}}|}{minsup+1} < minpr$. Therefore, $\{e,f\}$ is also not a quasi-periodic-frequent pattern. In other words, all supersets of 'e' are also not quasi-periodic-frequent.

Definition 5. *Mai of X: Let IP^X be the set of interesting periodic occurrences for pattern X. The mai of X, denoted as $mai(X) = \frac{|IP^X|}{(minsup+1)}$.*

For a pattern X, $mai(X) \in [0,\infty)$. If $mai(X) \geq 1$, then $S(X) \geq minsup$ because $IP^X \subseteq P^X$. If $mai(X) < 1$, then $S(X) \geq minsup$ or $S(X) < minsup$. So, we have to consider both *mai* and support values of a pattern for generating quasi-periodic-frequent patterns. The **pruning technique** is as follows. If $mai(X) < minpr$, then X and its supersets cannot generate any quasi-periodic-frequent pattern. However, if $mai(X) \geq minpr$ and *periodic-ratio*$(X) < minpr$, then X must be considered for generating higher order patterns even though X is not a quasi-periodic-frequent pattern. The reason is that its supersets can still be quasi-periodic-frequent.

Definition 6. ***Potential pattern:*** *For a pattern X, if $mai(X) \geq minpr$ and $S(X) \geq minsup$, then X is said to be a potential pattern.*

[1] In this paper, *minsup* is discussed in terms of minimum support count.

A potential pattern containing only one item (1-itemset) is called a **potential item**. A potential pattern need not necessarily be a quasi-periodic-frequent. pattern. However, every quasi-periodic-frequent pattern is a potential pattern. Thus, we use potential patterns to discover the quasi-periodic-frequent patterns.

Issue 2: The conventional (frequent) pattern-growth algorithms that are based on FP-tree [2] cannot be used to discover these patterns. It is because the structure of FP-tree captures only the occurrence frequency and disregards the periodic behavior of a pattern. To capture both frequency and periodic behavior of the patterns, an alternative pattern-growth algorithm based on a tree structure, called Periodic-Frequent tree (PF-tree), has been proposed in the literature [3]. The nodes in PF-tree do not maintain the support count as in FP-tree. Instead, they maintain a list of *tids* (or a *tid*-list) in which the corresponding item has appeared in a database. These *tid*-lists are later aggregated to derive the final *tid*-list of a pattern (i.e., T^X for pattern X). A complete search on this *tid*-list gives the *support* and *periodic-ratio*, which are later used to determine whether the corresponding pattern is quasi-periodic-frequent or a non-quasi-periodic-frequent pattern. In other words, the pattern-growth technique has to perform a complete search on a pattern's *tid*-list to determine whether it is quasi-periodic-frequent or a non-quasi-periodic-frequent pattern.

In very large databases, the *tid*-list of a pattern can be very long. In such cases, the task of performing a complete search on a pattern's *tid*-list can be a computationally expensive process. To reduce the computational cost, we introduce another pruning technique based on the greedy search. The technique is as follows:

> Let t_{pos}^X denote the current position in T^X. For the user-defined *minpr*, if $t_{pos}^X > (|T^X| - ((|T^X|+1) \times minpr))$ and $|IP^X| = 0$, then X is not a quasi-periodic-frequent pattern.

The correctness of this technique is shown in Lemma 2.

Lemma 2. *Let t_{pos}^X denote the current position in T^X. For the user-defined minpr, if $t_{pos}^X > (|T^X| - ((|T^X|+1) \times minpr))$ and $|IP^X| = 0$, then X is not a quasi-periodic-frequent pattern.*

Proof. The total number of *periods* for pattern X is $(|T^X|+1)$ (see Property ?). If a frequent pattern X is quasi-periodic-frequent, then

$$\begin{aligned} pr(X) &\geq minpr \\ = \frac{|IP^X|}{|P^X|} &\geq minpr \\ = |IP^X| &\geq |P^X| \times minpr \end{aligned} \tag{2}$$

Since$|P^X| = |T^X| + 1$ (Property 1), Equation 2 can be expressed as

$$= |IP^X| \geq |T^X + 1| \times minpr \tag{3}$$

The Equation 3 is possible if and only if

$$t^X_{pos} \leq (|T^X| - ((|T^X| + 1) \times minpr)) \tag{4}$$

Therefore, if $t^X_{pos} > (|T^X| - ((|T^X| + 1) \times minpr)$ and $|IP^X| = 0$, then X is not a quasi-periodic-frequent pattern. Hence proved.

Using the above ideas, the proposed QPF-growth approach discovers quasi-periodic-frequent patterns by constructing a tree structure, called Quasi-Periodic-Frequent tree (QPF-tree). Now, we discuss the structure of QPF-tree.

4.2 Structure of QPF-tree

The QPF-tree consists of two components: QPF-list and a prefix-tree. QPF-list is a list with three fields: item (i), support or frequency (s) and number of interesting periods (ip). The node structure of prefix-tree in QPF-tree is same as the prefix-tree in PF-tree [3], which is as follows.

The prefix-tree in QPF-tree explicitly maintains the occurrence information for each transaction in the tree structure by keeping an occurrence transaction-id list, called *tid*-list, only at the last node of every transaction. Two types of nodes are maintained in a QPF-tree: ordinary node and *tail*-node. The ordinary node is similar to the nodes used in FP-tree, whereas the latter is the node that represents the last item of any sorted transaction. The structure of a *tail*-node is $N[t_1, t_2, \cdots, t_n]$, where N is the node's item name and $t_i, i \in [1, n]$, (n be the total number of transactions from the root up to the node) is a transaction-id where item N is the last item. Like the FP-tree [2], each node in a QPF-tree maintains parent, children, and node traversal pointers. However, irrespective of the node type, no node in a QPF-tree maintains support count value in it.

The QPF-growth employs the following three steps to discover quasi-periodic-frequent patterns.

1. Construction of QPF-list to identify potential items
2. Construction of QPF-tree using the potential items
3. Mining quasi-periodic-frequent patterns from QPF-tree

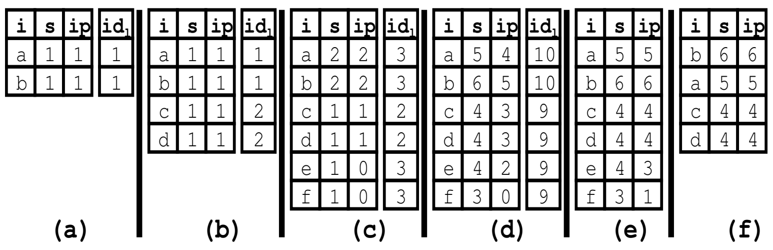
We now discuss each of these steps using the transactional database shown in Fig. 1(a). Let the user-specified *minsup*, *maxperiod* and *minpr* values be 3, 2 and 0.8, respectively.

4.3 Construction of QPF-list

Let id_l be a temporary array that explicitly records the $tids$ of the last occurring transactions of all items in the QPF-list. Let t_{cur} be the tid of current transaction. The QPF-list is, therefore, maintained according to the process given in Algorithm 1.

In Fig. 2, we show how the QPF-list is populated for the transactional database shown in Fig. 1(a). With the scan of the first transaction $\{a, b\}$ (i.e., $t_{cur} = 1$), the items 'a' and 'b' in the list are initialized as shown in Fig. 2(a) (lines 4 to 6 in Algorithm 1). The scan on the next transaction $\{c, d\}$ with $t_{cur} = 2$ initializes the items 'c' and 'd' in

QPF-list as shown in Fig. 2(b). The scan on next transaction $\{a,b,e,f\}$ with $t_{cur} = 3$ initializes QPF-list entries for the items '*e*' and '*f*' with $id_l = 3$, $s = 1$ and $ip = 0$ because $t_{cur} > maxperiod$ (line 6 in Algorithm 1). Also, the $\{s;ip\}$ and id_l values for the items '*a*' and '*b*' are updated to $\{2;2\}$ and 3, respectively (lines 8 to 11 in Algorithm 1). The QPF-list resulted after scanning third transaction is shown in Fig. 2(c). The QPF-list after scanning all ten transactions is given in Fig. 2(d). To reflect the correct number of interesting periods for each item in the QPF-list, the whole QPF-list is refreshed as mentioned from lines 16 to 20 in Algorithm 1. The resultant QPF-list is shown in Fig. 2(e). Based on the above discussed ideas, the items '*e*' and '*f*' are pruned from the QPF-list because their *mai* values are less than *minpr* (lines 22 to 24 in Algorithm 1). The items '*a*', '*b*', '*c*' and '*d*' are generated as quasi-periodic-frequent patterns (lines 26 to 28 in Algorithm 1). The items which are not pruned are sorted in descending order of their support values (line 31 in Algorithm 1). The resultant QPF-list is shown in Fig. 2(f). Let *PI* be the set of potential items that exist in QPF-list.



(a)

i	s	ip	id_l
a	1	1	1
b	1	1	1

(b)

i	s	ip	id_l
a	1	1	1
b	1	1	1
c	1	1	2
d	1	1	2

(c)

i	s	ip	id_l
a	2	2	3
b	2	2	3
c	1	1	2
d	1	1	2
e	1	0	3
f	1	0	3

(d)

i	s	ip	id_l
a	5	4	10
b	6	5	10
c	4	3	9
d	4	3	9
e	4	2	9
f	3	0	9

(e)

i	s	ip
a	5	5
b	6	6
c	4	4
d	4	4
e	4	3
f	3	1

(f)

i	s	ip
b	6	6
a	5	5
c	4	4
d	4	4

Fig. 2: QPF-list. (a) After scanning first transaction (b) After scanning second transaction (c) After scanning third transaction (d) After scanning entire transactional database (e) Reflecting correct number of interesting periods (f) compact QPF-list containing only potential items.

4.4 Construction of QPF-tree

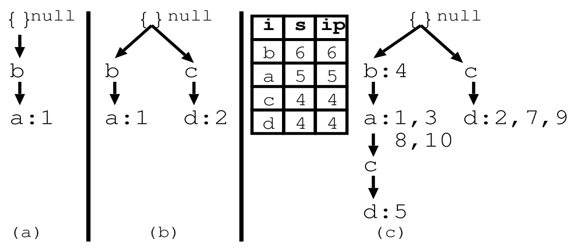
With the second database scan, the QPF-tree is constructed in such a way that it only contains nodes for items in QPF-list. We use an example to illustrate the construction of a QPF-tree.

Continuing with the ongoing example, using the FP-tree [2] construction technique, only the items in QPF list take part in QPF-tree construction. For simplicity of figures, we do not show the node traversal points in trees; however, they are maintained in a fashion like FP-tree does. The tree construction starts with inserting the first transaction $\{a,b\}$ according to the QPF-list order i.e., $\{b,a\}$, as shown in Fig. 3(a). The *tail*-node "$a:1$" carries the *tid* of the transaction. Fig. 3(b) shows the QPF-tree generated in the similar procedure after scanning the second transaction. Fig. 3(c) shows the resultant QPF-tree generated after scanning every transaction in the database.

Based on the QPF-list population technique and the above example, we have the following property and lemmas of a QPF-tree. For each transaction t in T, let $PI(t)$ be the set of potential items in t that exist in QPF-list, i.e., $PI(t) = item(t) \cap PI$, and is called the *potential item projection* of t.

Algorithm 1 QPF-list (T: Transactional database, I: set of items, $minsup$: minimum support, $maxperiod$: maximum period, $minpr$: minimum periodic-ratio)

```
1: Let $id_l$ be a temporary array that explicitly records the $tids$ of the last occurring transactions
   of all items in the QPF-list. Let $t_{cur}$ be the $tid$ of current transaction.
2: for each transaction $t_{cur} \in T$ do
3:     for each item $i$ in $t_{cur}$ do
4:         if $t_{cur}$ is $i$'s first occurrence then
5:             $s = 1$, $id_l = t_{cur}$;
6:             $ip = t_{cur} \leq maxperiod ? 1 : 0$;
7:         else
8:             if $t_{cur} - id_l \leq maxperiod$ then
9:                 $ip++$;
10:            end if
11:            $s++$, $id_l = t_{cur}$;
12:        end if
13:    end for
14: end for
15: /*Measuring correct number of interesting periods.*/
16: for each item $i$ in QPF-list do
17:     if $|T| - id_l \leq maxperiod$ then
18:         $ip++$;
19:     end if
20: end for
21: /* Identifying potential items in QPF-list. */
22: for each item $i$ in QPF-list do
23:     if $(s < minsup) || (\frac{ip}{(minsup+1)} < minpr)$ then
24:         remove $i$ from QPF-list;
25:     else
26:         if $\frac{ip}{(s+1)} \geq minpr$ then
27:             output $i$ as quasi-periodic-frequent item.
28:         end if
29:     end if
30: end for
31: Sort the remaining (potential) items in the QPF-list in descending order of their support
    values. Let $PI$ be the set of potential items.
```



i	s	ip
b	6	6
a	5	5
c	4	4
d	4	4

Fig. 3: QPF-tree. (a) After scanning first transaction (b) After scanning second transaction and (c) After scanning complete transactional database.

Property 3. A QPF-tree maintains a complete set of all *potential item* projection for each transaction in T only once.

Lemma 3. *Given a transactional database T, a minsup, a maxperiod and a minpr, the complete set of all* potential item *projections of all transactions in T can be derived from the QPF-tree.*

Proof. Based on Property 3, $PI(t)$ 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 the *tid*-list of the *tail*-node).

Lemma 4. *The size of a QPF-tree (without the root node) on a transactional database T for a minsup, a maxperiod and a minpr is bounded by* $\sum\limits_{t \in T} |PI(t)|$.

Proof. According to the QPF-tree construction process and Lemma 3, each transaction t contributes at best on path of the size $|PI(t)|$ to a QPF-tree. Therefore, the total size contribution of all transactions can be $\sum\limits_{t \in T} |PI(t)|$ at best. However, since there are usually a lot of common prefix patterns among the transactions, the size of a QPF-tree is normally much smaller than $\sum\limits_{t \in T} |PI(t)|$.

One can assume that the structure of a QPF-tree may not be memory efficient, since it explicitly maintains *tids* of each transaction. But, we argue that the QPF-tree achieves the memory efficiency by keeping such transaction information only at the *tail*-nodes and avoiding the support count field at each node. Moreover, keeping the *tid* information in tree can also been found in literature for efficient periodic-frequent pattern mining [3].

Therefore, the highly compact QPF-tree structure maintains the complete information for all periodic-frequent patterns. Once the QPF-tree is constructed, we use an FP-growth-based pattern growth mining technique to discover the complete set of periodic-frequent patterns from it.

4.5 Mining Quasi-Periodic-Frequent Patterns

Even though both of the QPF-tree and FP-tree arrange items in support-descending order, we cannot directly apply FP-growth mining on a QPF-tree. The reason is that, QPF-tree does not maintain support count at each node, and it handles the *tid*-lists at *tail*-nodes. Therefore, we devise a pattern growth-based bottom-up mining technique that can handle the additional features of QPF-tree. The basic operations in mining a QPF-tree for quasi-periodic-frequent patterns are (i) counting length-1 potential items, (ii) constructing the prefix-tree for each potential pattern, and (iii) constructing the conditional tree from each prefix-tree. The QPF-list provides the length-1 potential items. Before discussing the prefix-tree construction process we explore the following important property and lemma of a QPF-tree.

Property 4. A tail-node in a QPF-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 tid-list.

Lemma 5. *Let $B = \{b_1, b_2, \cdots, b_n\}$ be a branch in a QPF-tree where node b_n is the tail-node carrying the tid-list of the path. If the tid-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 tid-list without any loss.*

Proof. Based on Property 4, b_n maintains the occurrence information of the path B' at least in the transactions in its *tid*-list. Therefore, the same *tid*-list at node b_{n-1} exactly maintains the same transaction information for B' without any loss.

Using the feature revealed by the above property and lemma, we proceed to construct each prefix-tree starting from the bottom-most item, say i, of the QPF-list. Only the prefix sub-paths of nodes labeled i in the QPF-tree are accumulated as the prefix-tree for i, say PT_i. Since i is the bottom-most item in the QPF-list, each node labeled i in the QPF-tree must be a *tail*-node. While constructing the PT_i, based on Property 4 we map the *tid*-list of every node of i to all items in the respective path explicitly in a temporary array (one for each item). It facilitates the support and number of interesting periods' calculation for each item in the QPF-list of PT_i. Moreover, to enable the construction of the prefix-tree for the next item in the QPF-list, based on Lemma 5 the *tid*-lists are pushed-up to respective parent nodes in the original QPF-tree and in PT_i as well. All nodes of i in the QPF-tree and i's entry in the QPF-list are deleted thereafter. Fig. 4(a) shows the status of the QPF-tree of Fig. 3(c) after removing the bottom-most item '*d*'. Besides, the prefix-tree for '*d*', PT_d is shown in Fig. 4(b).

The conditional tree CT_i for PT_i is constructed by removing the items whose support is less than *minsup* or *mai* value is less than *minpr*. If the deleted node is a *tail*-node, its *tid*-list is pushed-up to its parent node. Fig. 4(c), for instance, shows the conditional tree for '*d*', CT_d constructed from the PT_d of Fig. 4(b). The contents of the temporary array for the bottom item '*j*' in the QPF-list of CT_i represent T^{ij} (i.e., the set of all *tids* where items i and j are occurring together). Therefore, it is rather simple calculation to compute $S(ij)$, $mai(ij)$ and $periodic\text{-}ratio(ij)$ from T^{ij} by generating P^{ij}. If $S(ij) \geq minsup$ and $periodic\text{-}ratio(ij) \geq minpr$, then the pattern "$ij$" is generated as a quasi-periodic-frequent pattern with support and periodic-ratio values of

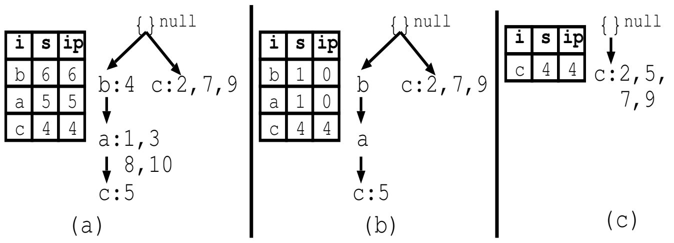


Fig. 4: Prefix-tree and conditional tree construction with QPF-tree. (a) QPF-tree after removing item '*d*' (b) Prefix-tree for '*d*' and (c) Conditional tree for '*d*'.

$S(ij)$ and $periodic\text{-}ratio(ij)$, respectively. The same process of creating prefix-tree and its corresponding conditional tree is repeated for further extensions of "*ij*". Else, if $mai(ij) \geq minpr$ and $S(ij) \geq minsup$, then the above process is still repeated for further extensions of "*ij*" even though "*ij*" is not a quasi-periodic-frequent pattern. The whole process of mining for each item is repeated until $QPF\text{-}list \neq \emptyset$.

For the transactional database in Fig. 1(a), the quasi-periodic-frequent patterns generated at $minsup = 3$, $maxperiod = 2$ and $minpr = 0.8$ are shown in Table 1. The above bottom-up mining technique on support-descending QPF-tree is efficient, because it shrinks the search space dramatically as the mining process progresses.

Table 1: Quasi-periodic-frequent patterns generated for the transactional database shown in Fig. 1(a).

S.No.	Patterns	Support	Periodic-ratio	S. No.	Patterns	Support	Periodic-ratio
1	{a}	6	0.85	4	{d}	4	0.8
2	{b}	5	0.83	5	{c,d}	4	0.8
3	{c}	4	0.8	6	{a,b}	5	0.83

4.6 Relation Between Frequent, Periodic-Frequent and Quasi-Periodic-Frequent Patterns

In a transactional database T, let F be the set of frequent patterns generated at $minsup = a$. In T, let PF be the set of periodic-frequent patterns generated at $minsup = a$ and $maxprd = b$. In T, let QPF be the set of quasi-periodic-frequent patterns generated at $minsup = a$, $maxperiod = b$ and $minpr = c$. The relationship between these patterns is $PF \subseteq QPF \subseteq F$. If $minpr = 1$, then $PF = QPF$. Else, $PF \subseteq QPF$.

4.7 Differences between QPF-tree and PF-tree

The prefix-structure of QPF-tree is similar to that of PF-tree. However, the differences between these two trees are as follows.

1. The list (i.e., PF-list) of PF-tree captures periodicity (maximum period) of a pattern. The QPF-list of QPF-tree captures the number of interesting periods of a pattern.

2. The periodic-frequent patterns follow *downward closure property*. Therefore, only the periodic-frequent items (or 1-patterns) participate in the construction of PF-tree. The quasi-periodic-frequent patterns do not follow *downward closure property*. Therefore, potential items participate in the construction of QPF-tree.
3. Mining procedure used for discovering periodic-frequent patterns from PF-tree and quasi-periodic-frequent patterns from QPF-tree are completely different. The reason is quasi-periodic-frequent patterns do not satisfy *downward closure property*.

In the next section, we present the experimental results on finding quasi-periodic-frequent patterns from the QPF-tree.

5 Experimental Results

In this section, we first investigate the interestingness of the quasi-periodic-frequent patterns with respect to the periodic-frequent patterns. Next, we investigate the compactness (memory requirements) and execution time of QPF-tree on various datasets with varied *minsup*, *maxperiod* and *minpr* values. Finally, we investigate the scalability of the QPF-tree. All programs are written in C++ and run with Ubuntu on a 2.66 GHz machine with 1 GB memory. The runtime specifies the total execution time, i.e., CPU and I/Os. The experiments are pursued on synthetic ($T10I4D100K$) and real-world datasets (retail, mushroom and kosarak). The $T10I4D100k$ dataset is a sparse dataset containing 1,00,000 transactions and 886 items. The retail dataset [6] is also a sparse dataset containing 88,162 transactions and 16,470 items. The mushroom dataset is a dense dataset containing 8,124 transactions and 119 items. The kosarak dataset is a huge sparse dataset with a large number of distinct items (41,270) and transactions (990,002). These datasets are widely used in frequent pattern mining literature and are available at Frequent Itemset MIning (FIMI) repository [7].

5.1 Experiment 1: Interestingness of Quasi-Periodic-Frequent Patterns

In this experiment, we study the interestingness of quasi-periodic-frequent patterns with reference to the periodic-frequent patterns. Note that at $minpr = 1$, all the quasi-periodic-frequent patterns discovered using the proposed model are periodic-frequent patterns.

The quasi-periodic-frequent patterns generated on the variations of *minsup*, *maxperiod* and *minpr* values over several datasets are reported in Table 2. The columns titled "A" and "B" respectively denote the number of quasi-periodic-frequent patterns and maximal length of the quasi-periodic-frequent patterns getting generated at different *minsup*, *maxperiod* and *minpr* values.

The data in the table demonstrate the following. First, increase in *minsup* (keeping other constraints fixed) decreases the number of quasi-periodic-frequent patterns. It is because many items fail to satisfy increased *minsup* constraint. Second, increase in *minpr* also decreases the number of quasi-periodic-frequent patterns. The reason is that many patterns have failed to appear periodically for longer time durations. Third, increase in *maxperiod* increases the number of quasi-periodic-frequent patterns. It is because of the increased interval range in which a pattern should reappear.

Table 2: Quasi-periodic-frequent patterns generated at different *minsup*, *maxperiod* and *minpr* values.

Database	*minsup*	$maxperiod_1 = 0.1\%$						$maxperiod_2 = 0.5\%$					
		minpr=0.5		*minpr*=0.75		*minpr*=1		*minpr*=0.5		*minpr*=0.75		*minpr*=1	
		A	B	A	B	A	B	A	B	A	B	A	B
T10I4D100k	0.1%	624	5	272	1	0	0	20748	10	6360	8	229	2
	1.0%	385	3	272	1	0	0	385	3	385	3	229	2
Retail	0.1%	837	5	293	5	4	2	5849	5	2177	5	15	3
	1.0%	159	4	102	4	4	2	159	4	159	4	15	3
Mushroom	10%	574,431	16	570,929	16	15	4	574,431	16	574,431	16	135	6
	20%	53,583	15	53,583	15	15	4	53,583	15	53,583	15	135	6

In Table 2, it can be observed that at $minpr = 1$, the number of quasi-periodic-frequent patterns (or periodic-frequent patterns) getting generated in different datasets (especially in sparse datasets) are very few, and also the maximal length of these patterns is very less. Whereas, by relaxing the *minpr*, we are able to generate more number of quasi-periodic-frequent patterns, and more importantly, the maximal length of the patterns is relatively large. This shows that relaxing the periodic occurrences of a pattern throughout a database, facilitates the user to find interesting knowledge pertaining to the patterns that are "mostly" occurring periodically in the database.

5.2 Experiment 2: Compactness and Execution Time of the QPF-tree

The memory consumptions of QPF-tree on the variations of *minsup*, *maxperiod* and *minpr* values over several datasets are reported in Table 3. It can be observed that similar observations that are drawn from Experiment 1 can also be observed with respect to the memory requirements of QPF-tree. The reason is that memory requirements of QPF-tree depends on the number of quasi-periodic-frequent patterns getting generated. More importantly, it is clear from the Table 3 that, the structure of QPF-tree can easily be handled in a memory efficient manner irrespective of the dataset type (dense or sparse) or size (large or small) and threshold values.

Table 3: Memory requirements for the QPF-tree. Memory is measured in *MB*.

Dataset	*minsup*	$maxperiod_1 = 0.1\%$			$maxperiod_2 = 0.5\%$		
		minpr=0.5	*minpr*=0.75	*minpr*=1	*minpr*=0.5	*minpr*=0.75	*minpr*=1
T10I4D100k	0.1%	11.077	10.975	10.828	11.226	11.197	11.179
	1.0%	8.255	7.820	6.656	8.255	8.255	8.255
Retail	0.1%	4.908	4.193	3.542	6.914	6.254	5.578
	1.0%	1.060	0.942	0.789	1.060	1.060	1.051
Mushroom	10%	0.250	0.250	0.235	0.250	0.250	0.250
	20%	0.131	0.131	0.122	0.131	0.131	0.127

The runtime taken by QPF-growth for generating quasi-periodic-frequent patterns at different *minsup*, *maxperiod* and *minpr* values on various datasets are shown in Table 4. The runtime encompasses all phases of QPF-list and QPF-tree constructions, and the corresponding mining operation. The runtime taken for generating quasi-periodic-frequent patterns depends upon the number of quasi-periodic-frequent patterns getting generated. Therefore, similar observations that were drawn in Experiment 1 can also be drawn from Table 4.

Table 4: Runtime requirements for the QPF-tree. Runtime is measured in seconds.

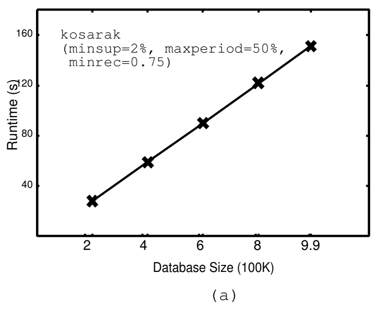
Dataset	*minsup*	$maxperiod_1 = 0.1\%$			$maxperiod_2 = 0.5\%$		
		minpr=0.5	*minpr*=0.75	*minpr*=1	*minpr*=0.5	*minpr*=0.75	*minpr*=1
T10I4D100k	0.1%	120.533	124.766	128.057	105.951	110.085	113.957
	1.0%	109.257	102.138	83.009	112.400	107.000	112.604
Retail	0.1%	43.780	35.982	30.527	67.782	58.759	50.289
	1.0%	15.441	15.182	14.352	15.780	15.909	15.894
Mushroom	10%	20.580	20.180	16.510	20.320	20.270	18.860
	20%	2.920	2.790	2.780	2.930	3.000	2.740

5.3 Experiment 3: Scalability of QPF-tree

We study the scalability of our QPF-tree on execution time and required memory by varying the number of transactions in database. We use real *kosarak* dataset for the scalability experiment, since it is a huge sparse dataset. We divided the dataset into five portions of 0.2 million transactions in each part. Then we investigated the performance of QPF-tree after accumulating each portion with previous parts and performing quasi-periodic-frequent pattern mining each time. We fix the $minsup = 2\%$, $maxperiod = 50\%$ and $minpr = 0.75$ for each experiment. The experimental results are shown in Fig. 5. The time and memory in y-axes of the left and right graphs in Fig. 5 respectively specify the total execution time and required memory with the increase of database size. It is clear from the graphs that as the database size increases, overall tree construction and mining time, and memory requirement increases. However, QPF-tree shows stable performance of about linear increase in runtime and memory consumption with respect to the database size. Therefore, it can be observed from the scalability test that QPF-tree can mine quasi-periodic-frequent patterns over large datasets and distinct items with considerable amount of runtime and memory.

6 Conclusion and Future Work

In this paper, we explored the notion "patterns that are mostly appearing periodically in a database are interesting," and introduced a new class of user-interest-based frequent patterns, called quasi-periodic-frequent patterns. This paper proposes a more flexible and powerful model. It facilitates the user to specify the minimum proportion in which a pattern should appear periodically throughout the database. This model enables the user to find periodically occurring patterns in the databases of longer time duration.



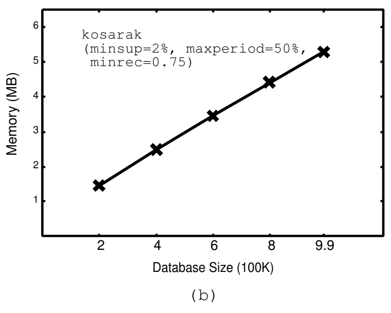


Fig. 5: Scalability of QPF-tree. (a) Runtime (b) Memory

Also, an efficient pattern-growth approach (i.e., QPF-growth) was proposed to mine these patterns. The quasi-periodic-frequent patterns do not follow *downward closure property*. Therefore, the proposed QPF-growth approach uses the introduced heuristic (*mai*) to minimize the search space. The experimental results from synthetic and real-world databases, demonstrate that quasi-periodic patterns can provide useful information. Experimental results also demonstrate that our QPF-growth can be time and memory efficient during mining the quasi-periodic-frequent patterns, and is highly scalable in terms of runtime and memory consumption.

As a part of future work, we have to investigate the mining of quasi-periodic-frequent patterns consisting of rare items.

References

1. Agrawal, R., Imielinski, T., and Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207-216 (1993)
2. Jiawei, H., Jian, P., Yiwen, Y., and Runying, M.: Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 53-87 (2004)
3. Tanbeer, S. K., Ahmed, C. F., Jeong, B., and Lee, Y.: Discovering Periodic-Frequent Patterns in Transactional Databases. In: Pacific Asia Knowledge Discovery in Databases (2009)
4. Uday Kiran, R., Krishna Reddy, P.: Mining Rare Periodic-Frequent Patterns Using Multiple Minimum Supports. In: 15th International Conference on Management of Data (2009)
5. Uday Kiran, R., Krishna Reddy, P.: Mining Rare Periodic-Frequent Patterns Using Maximum Item Constraints. In: ACM Compute (2010)
6. Brijs T., Swinnen G., Vanhoof K., and Wets G.: The use of association rules for product assortment decisions - a case study. In: Knowledge Discovery and Data Mining, 1999.
7. Frequent Itemset MIning Repository, `http://fimi.cs.helsinki.fi/data/`