Discovering Chronic-Frequent Patterns in Transactional Databases

R. Uday Kiran[1,2] and Masaru Kitsuregawa[1,3]

[1]Institute of Industrial Science, The University of Tokyo, Tokyo, Japan.
[2]National Institute of Information and Communication Technology, Tokyo, Japan.
[3]National Institute of Informatics, Tokyo, Japan.
{uday_rage, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. This paper investigates the partial periodic behavior of the frequent patterns in a transactional database, and introduces a new class of user-interest-based patterns known as chronic-frequent patterns. Informally, a frequent pattern is said to be **chronic** if it has sufficient number of cyclic repetitions in a database. The proposed patterns can provide useful information to the users in many real-life applications. An example is finding chronic diseases in a medical database. The chronic-frequent patterns satisfy the anti-monotonic property. This property makes the pattern mining practicable in real-world applications. The existing pattern growth techniques that are meant to discover frequent patterns cannot be used for finding the chronic-frequent patterns. The reason is that the tree structure employed by these techniques' capture only the frequency and disregards the periodic behavior of the patterns. We introduce another pattern-growth algorithm which employs an alternative tree structure, called Chronic-Frequent pattern tree (CFP-tree), to capture both frequency and periodic behavior of the patterns. Experimental results show that the proposed patterns can provide useful information and our algorithm is efficient.

Keywords: Data mining, knowledge discovery in databases, frequent patterns and periodic patterns.

1 Introduction

A time series is a collection of events obtained from sequential measurements overtime. Periodic patterns are an important class of regularities that exist in a time series. Periodic pattern mining involves discovering all those patterns that have exhibited either complete or partial cyclic repetitions in a time series. Periodic pattern mining has several real-world applications including prediction, forecasting and detection of unusual activity. A classic application is market-basket analysis. It analyzes how regularly items are being purchased by the customers. For example, if the customers are purchasing '$Bread$' and 'Jam' together at every hour of a day, then the set $\{Bread, Jam\}$ represents a periodic pattern.

The problem of finding periodic patterns has been widely studied in [1–6]. The basic model used in all of these studies, however, remains the same and is as follows:

1. Split the given time series into distinct subsets (or periodic-segments) of a fixed length.
2. Discover all periodic patterns that satisfy the user-defined *minimum support* (*minSup*). The *minSup* controls the minimum number of periodic-segments in which a pattern must appear.

Example 1. Given the time series $TS = a\{bc\}baebace$ and the user-defined *period* as $3, TS$ 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 *periods* of TS_1 and TS_2. Therefore, its *support* count is 2. If the user-defined *minSup* count is 2, then $\{a \star b\}$ represents a periodic pattern. In the above time series, we have applied braces only for the events having more than one item for brevity. An event represents a set of items (or an itemset) having some occurrence order.

The popular adoption and successful industrial application of this basic model suffers from the following issues.

– The basic model considers time series as a symbolic sequence. As a result, the model fails to consider the actual temporal information of the events within a sequence.
– This model suffers from the sparsity problem. That is, most of the discovered patterns contain many wild characters with a very few number of events. For example, $a \star\star\star\star\star\star\star\star\star\star\star\star\star bc \star\star\star\star\star\star\star\star a \star\star\star\star\star\star\star\star b$. This problem makes the discovered patterns impracticable in applications.
– The periodic patterns satisfy the *anti-monotonic property* [7]. That is, all non-empty subsets of a periodic pattern are also periodic patterns. However, this property is insufficient to make the pattern mining practical or computationally inexpensive in the case of time series. The reason is number of frequent i-patterns shrink slowly (when $i > 1$) as i increases in a time series. The slow speed of decrease in the number of frequent i-patterns is due to the strong correlation between frequencies of patterns and their sub-patterns [2].

To confront these issues, researchers have introduced periodic-frequent pattern mining which involves discovering all those frequent patterns that have exhibited complete cyclic repetitions in a temporally ordered transactional database [8–11]. As the real-world is generally imperfect, we have observed that the existing periodic-frequent pattern mining algorithms cannot discover those interesting frequent patterns that have exhibited partial cyclic repetitions in a database.

With this motivation, this paper investigates the partial periodic behavior of the frequent patterns in a transactional database, and introduce a class of user-interest-based patterns known as **chronic-frequent patterns**. Informally, a frequent pattern is said to be **chronic-frequent** if it has sufficient number of cyclic repetitions in a database. A novel measure, called *periodic-recurrence*, has been introduced in this paper. This measure assess the periodic interestingness of a frequent pattern with respect to the number of cyclic repetitions in the entire database. The patterns discovered with this measure satisfy the anti-monotonic property. That is, all non-empty subsets of a chronic-frequent pattern are also chronic-frequent. This property makes the chronic-frequent pattern mining practicable in real-life applications. The existing pattern-growth techniques that are meant to discover frequent patterns in a transactional database [12] cannot be used for finding the chronic-frequent patterns. It is because the tree structure used by these techniques' capture only the frequency and disregard the periodic behavior of the patterns. In this paper, we have introduced another tree structure, called Chronic-Frequent Pattern Tree (CFP-tree), to capture both frequency and periodic behavior of the patterns. A pattern-growth algorithm, called Chronic-Frequent pattern-growth (CFP-growth), has been proposed to discover the patterns from CFP-tree. Experimental results show that CFP-growth is runtime efficient and scalable as well.

The rest of the paper is organized as follows. Section 2 describes the related work on periodic pattern mining. Section 3 introduces our model of chronic-frequent patterns. Section 4 describes the working of CFP-growth algorithm. The experimental evaluation of CFP-growth has been presented in Section 5. Finally, Section 6 concludes the paper with future research directions.

2 Related Work

Finding periodic patterns has been widely investigated in various domains as temporal patterns [13] and cyclic association rules [14]. These approaches discover all those patterns which are exhibiting complete cyclic repetitions in a time series data. Since the real-world is imperfect, Han et al. [1] have introduced a model to find periodic patterns which are exhibiting either complete or partial cyclic repetitions in a time series. Later, they have proposed the *max-subpattern hit set property* to reduce the computational cost of finding the periodic patterns [2]. Berberidis et al. [4] and Cao et al. [5] have tried to address an open problem of specifying the *period* using autocorrelation and other methods. Yang et al. [3] have used **information gain** to discover periodic patterns involving both frequent and rare items. All of these approaches consider time series as a symbolic sequence, and therefore, do not consider the actual temporal information of the events within a series.

Tanbeer et al. [8] have represented each event in time series as a pair constituting of an itemset and its timestamp. Next, they have modeled time series as a temporally ordered transactional database, and investigated the **full periodic behavior** of the frequent patterns to discover a class of user-interest-based patterns known as *periodic-frequent patterns*. The approach of representing time series as a transactional database has the following advantages:

- Symbolic sequences do not consider the temporal information of the events within a time series. On contrary, the same information is considered in temporally ordered transactional databases.
- The *anti-monotonic property* can not effectively reduce the search space in symbolic sequences [2]. However, the same property reduces the search space effectively in transactional databases.
- Fast algorithms, such as pattern-growth technique, can be employed to discover the patterns efficiently.

Recently, Chen et al. [15] have shown that by representing a symbolic sequence as a transactional database, one can employ a pattern-growth technique to outperform the *max-subpattern hit set algorithm* [2]. Thus, many researchers are extending Tanbeer's work to address the *rare item problem* [9, 10] and top$-k$ [11] periodic pattern mining. All of the above approaches try to discover those frequent patterns that are exhibiting complete cyclic repetitions in the entire database. On the contrary, our model focuses on finding the frequent patterns that are **exhibiting either complete or partial cyclic repetitions in a database**.

In [16, 17], we have introduced a measure known as *periodic-ratio* to assess the partial periodic behavior of a frequent pattern. Unfortunately, finding the patterns with this measure is a computationally expensive process because the discovered patterns do not satisfy the anti-monotonic property. In this paper, we have introduced an alternative interestingness measure which not only assess the partial periodic behavior of a frequent pattern, but also ensures that the discovered patterns satisfy the anti-monotonic property.

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

3 Proposed Model

Let $I = \{i_1, i_2, \cdots, i_n\}$ be the set of items. Let $X \subseteq I$ be a **pattern**. A pattern containing k number of items is called a **k-pattern**. A **transaction**, $tr = (tid, Y)$, is a tuple, where tid represents the transaction-identifier (or a **timestamp**) and Y is a pattern. A transactional database TDB over I is a set of transactions $T = \{t_1, t_2, \cdots, t_m\}$, $m = |TDB|$, where $|TDB|$ represents the size of TDB in total number of transactions. For a transaction $tr = (tid, Y)$, such that $X \subseteq Y$, it is said that X occurs in tr and such transaction-identifier is denoted as tid^X. Let $TID^X = \{tid_j^X, \cdots, tid_k^X\}$, $j, k \in [1, m]$ and $j \leq k$, be the set of all transaction-identifiers at which X has appeared in TDB. The size of TID^X is defined as the *support* of X, and denoted as $S(X)$. That is, $S(X) = |TID^X|$. The pattern X is said to be **frequent** if $S(X) \geq minSup$, where $minSup$ is the user-defined minimum support threshold.

Example 2. Consider the transactional database shown in Table 1. It contains 10 transactions. The tid of each transaction represents its sequential occurrence order with respect to a particular timestamp. The set of items, $I =$

$\{a, b, c, d, e, f, g, h\}$. The set of items, '$a$' and '$b$', i.e., '$\{a, b\}$' is known as an itemset (or a pattern). This pattern contains two items. Therefore, it is a 2-pattern. For brevity, we refer this pattern as 'ab'. The pattern 'ab' appears in the transactions having *tids* 1, 3, 5, 8 and 10. Therefore, $TID^{ab} = \{1, 3, 5, 8, 10\}$. The support of '$ab$' is the size of TID^{ab}. Therefore, $S(ab) = |TID^{ab}| = |\{1, 3, 5, 8, 10\}| = 5$. If the user-defined $minSup = 4$, then 'ab' is a frequent pattern as $S(ab) \geq minSup$.

Table 1. Transactional database.

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

Definition 1. *(A period of pattern X.) Let tid_p^X and tid_q^X, $p, q \in [1, m]$ and $p < q$, be the two consecutive transaction-ids where X has appeared in TDB. The number of transactions (or the time difference) between tid_p^X and tid_q^X can be defined as a* ***period*** *of X, say p_i^X. That is, $p_i^X = tid_q^X - tid_p^X$.*

Example 3. Continuing with Example 2, the pattern 'ab' has consecutively appeared in the *tids* of 1 and 3. Therefore, a period of 'ab,' i.e., $p_1^{ab} = 2$ $(= 3 - 1)$. Similarly, the other periods of 'ab' are as follows: $p_2^{ab} = 2$ $(= 5 - 3)$, $p_3^{ab} = 3$ $(= 8 - 5)$ and $p_4^{ab} = 2$ $(= 10 - 8)$.

Definition 2. *(An interesting period of pattern X.) Let $P^X = \{p_1^X, p_2^X, \cdots, p_k^X\}$, $k = S(X) - 1$, be the complete set of all periods of X in TDB. A $p_j^X \in P^X$ is said to be interesting iff $p_j^X \leq maxPrd$, where $maxPrd$ refers to the user-defined maximum period threshold. This definition captures the periodic occurrences of a pattern in the database.*

Example 4. The complete set of periods for 'ab', i.e., $P^{ab} = \{2, 2, 3, 2\}$. If the user-defined $maxPrd = 2$, then p_1^{ab} is an interesting period because $p_1^{ab} \leq maxPrd$. Similarly, p_2^{ab} and p_4^{ab} are interesting periods, however, p_3^{ab} is not an interesting period as $p_3^{ab} \nleq maxPrd$.

Definition 3. *(The periodic-recurrence of pattern X.) Let $IP^X \subseteq P^X$ be the set of periods such that $\forall p_j^X \in IP^X$, $p_j^X \leq maxPrd$. The size of IP^X gives the periodic-recurrence of X, say $PR(X)$. That is, $PR(X) = |IP^X|$.*

Example 5. The complete set of all interesting periods of 'ab', i.e., $IP^{ab} = \{p_1^{ab}, p_2^{ab}, p_4^{ab}\}$. Therefore, the *periodic-recurrence* of 'ab', i.e., $PR(ab) = |IP^{ab}| = 3$.

The above definition measures the number of periodic occurrences of a pattern X in TDB. Now, we define chronic-frequent patterns using the *support* and *periodic-recurrence* measures.

Definition 4. *(The chronic-frequent pattern X.) The frequent pattern X is said to be* ***chronic-frequent*** *if its periodic-recurrence is no less than the user-defined minimum periodic-recurrence threshold (minPR). That is, X is a chronic-frequent pattern if $S(X) \geq minSup$ and $PR(X) \geq minPR$.*

Example 6. If the user-defined $minPR = 3$, then the frequent pattern 'ab' is a chronic-frequent pattern as $PR(ab) \geq minPR$.

The *support* and a *period* of a pattern can be normalized to the scale of [0%, 100%] by expressing them in the percentage of $|TDB|$. Similarly, the *periodic-recurrence* of pattern X can also be normalized to the same scale by expressing it in percentage of $|TDB|-1$, where $|TDB|-1$ represents the maximum number of periods a pattern can have in a database (see Property 3). The patterns discovered with this normalization method satisfy the anti-monotonic property. The correctness of our argument is based on the Properties 1, 2 and 3 and shown in Lemma 1.

Property 1. The total number of *periods* for a pattern X, i.e., $|P^X| = S(X)-1$.

Property 2. (Apriori property [7]) If $X \subset Y$, then $TID^X \supseteq TID^Y$.

Property 3. The maximum support a pattern X can have in a database is $|TDB|$. From Property 1, it turns out that the maximum number of *periods* a pattern X can have in a database is $|TDB|-1$.

Lemma 1. *Let X and Y be the patterns such that $X \subset Y$. If $S(X) < minSup$ and $PR(X) < minPR$, then $S(Y) < minSup$ and $PR(Y) < minPR$.*

Proof. If $X \subset Y$, then $TID^X \supseteq TID^Y$ (see Property 2). Thus, $P^X \supseteq P^Y$, $IP^X \supseteq IP^Y$, $S(X) \geq S(Y)$ *and* $PR(X) \geq PR(Y)$ $\left(= \frac{|IP^X|}{|TDB|-1} \geq \frac{|IP^Y|}{|TDB|-1}\right)$. Therefore, if $S(X) < minSup$ and $PR(X) < minPR$, then $S(Y) < minSup$ and $PR(Y) < minPR$. Hence proved.

Definition 5. *(**Problem definition.**) Given a transactional database (TDB) and the user-defined* minimum support *(minSup),* maximum period *(maxPrd) and* minimum periodic-recurrence *(minPR) thresholds, discover the complete set of chronic-frequent patterns having* support *and* periodic-recurrence *no less than the minSup and minPR, respectively.*

In the next section, we discuss our algorithm to discover the complete set of chronic-frequent patterns from a transactional database.

4 The CFP-growth Algorithm

The CFP-growth algorithm involves two steps: (i) compressing the database into a tree-structure, called CFP-tree and (ii) recursive mining of CFP-tree to discover the patterns. Before describing these two steps, we explain the structure of CFP-tree.

4.1 Structure of CFP-tree

The CFP-tree includes a prefix-tree and a chronic-frequent item (or 1-pattern) list, called CFP-list. The CFP-list consists of four fields – *item name* (I), *total support* (f), *periodic-recurrence* (pr) and a pointer pointing to the first node in the prefix-tree carrying the item.

The prefix-tree in CFP-tree resembles the prefix-tree in FP-tree. However, to capture both $frequency$ and *chronic* behaviour of the patterns, the nodes in CFP-tree explicitly maintains the occurrence information for each transaction by keeping an occurrence transaction-id list, called *tid*-list. **To achieve memory efficiency, only the last node of every transaction maintains the *tid*-list**. Hence, there are two types of nodes maintained in a CFP-tree: ordinary node and *tail*-node. The former is the type of nodes similar to that used in FP-tree, whereas the latter is the node that represents the last item of any sorted transaction. The structure of *tail*-node is $I[tid_1, tid_2, \cdots, tid_m]$, where I is the node's item name and tid_i, $i \in [1, m]$, (m be the total number of transactions from the *root* up to the node) is a transaction-id where item I is the last item. The conceptual structure of CFP-tree is shown in Figure 1. Like in FP-tree, each node in a CFP-tree maintains parent, child and node traversal pointers. However, irrespective of the node type, no node in a CFP-tree maintains support value in it.

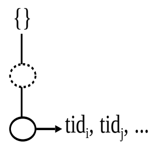

Fig. 1. The conceptual structure of prefix-tree in CFP-tree. The dotted ellipse represents the ordinary node, while the other ellipse represents the tail-node of sorted transactions with tids.

To facilitate high degree of compactness, items in a CFP-tree are arranged in support-descending item order. It has been proved in [18] that such tree can provide a highly compact tree structure, and an efficient mining phase using pattern-growth technique.

One can assume that the structure of prefix-tree in CFP-tree may not be memory efficient as it explicitly maintains tids of each transaction. However, it has been argued in the literature [8] 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. Furthermore, CFP-tree avoids the *complicated combinatorial explosion problem of candidate generation* as in Apriori-like algorithms [7]. In the literature, keeping the information pertaining to transactional-identifiers in a tree can also been found in efficient frequent pattern mining [19, 20].

```
for each transaction t_cur in TDB do
    for each item i in t_cur do
        if t_cur is i's first occurrence then
            Set f=1, pr=0 and id_l=t_cur
        else
            if t_cur-id_l <= maxPrd then
                Set ++pr.
            endif
            Set ++f and id_l=t_cur.
        endif
    endfor
endfor
Prune the items in the CFP-list that
have f<minsup or pr <minPR.
```

(a)

I	f	pr	id_l
a	1	0	1
b	1	0	1
h	1	0	1

(b)

I	f	pr	id_l
a	1	0	1
b	1	0	1
h	1	0	1
e	1	0	2
f	1	0	2

(c)

I	f	pr	id_l
a	2	1	3
b	2	1	3
h	1	0	1
e	1	0	2
f	1	0	2
g	1	0	3

(d)

I	f	pr	id_l
a	5	3	10
b	5	3	10
h	2	0	4
e	4	3	7
f	4	3	7
g	3	0	9
c	5	4	10
d	5	4	10

(e)

I	f	pr
a	5	3
b	5	3
c	5	4
d	5	4
e	4	3
f	4	3

(f)

Fig. 2. Construction of CFP-list. (a) Procedure (b) After scanning first transaction (c) After scanning second transaction (d) After scanning third transaction (e) After scanning every transaction and (f) Sorted list of chronic-frequent items.

4.2 Construction of the CFP-Tree

Since chronic-frequent patterns satisfy the anti-monotonic property, chronic-frequent items (or 1-patterns) play a key role in efficient mining of these patterns. Using the CFP-list, we perform a scan on the database to discover these items. Let t_{cur} denote the *tid* of current transaction. Let id_l be a temporary array that explicitly records the *tids* of last occurring transactions of all items in the CFP-list. Figure 2(a) shows the procedure followed to discover the chronic-frequent items. We illustrate this procedure using the database shown in Table 1.

The scan on the first transaction '$1 : a, b, h$', with $t_{cur} = 1$, inserts the items 'a', 'b' and 'h' into the CFP-list with $f = 1$, $pr = 0$ and $id_l = 1$ (see Figure 2(b)). The scan on the second transaction '$2 : e, f$', with $t_{cur} = 2$, inserts the items 'e' and 'f' into the CFP-list with $f = 1$, $pr = 0$ and $id_l = 2$ (see Figure 2(c)). The scan on the third transaction '$3 : a, b, g$', with $t_{cur} = 3$, adds the item 'g' into the CFP-list with $f = 1$, $pr = 0$ and $id_l = 3$. Simultaneously, the 'f', 'pr' and 'id_l' values of 'a' and 'b' are updated to 2, 1 and 3, respectively. Figure 2(d) shows the CFP-list constructed after scanning the third transaction. Similar approach is followed for the remaining transactions and CFP-list is updated accordingly. Figure 2(e) shows the CFP-list constructed after scanning all transactions in the database. The items having *support* less than the $minSup$ or *periodic-recurrence* less than the $minPR$ are pruned from the CFP-list. The remaining items are sorted in descending order of their frequencies. Figure 2(f) shows the sorted list of chronic-frequent items in CFP-list. Let CF denote this sorted list of items.

Using the FP-tree construction technique, only the items in the CF will take part in the construction of CFP-tree. The *tree* construction starts by inserting the first transaction, '$1 : a, b, h$', according to CFP-list order, as shown in Figure 3(a). The tail-node 'b : 1' carries the *tid* of the transaction. Please note that the item 'h' was not considered in the construction of CFP-tree as it is not a chronic-frequent item. Similar process is repeated for other transactions in the database. Figure 3(b), (c) and (d) respectively show the CFP-tree constructed after scanning second transaction, third transaction and entire database. For the simplicity of figures, we do not show the node traversal pointers in trees, however, they are maintained in a fashion like FP-tree does.

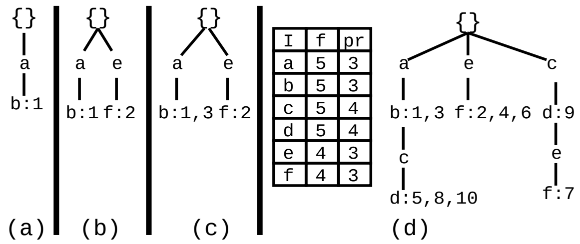

Fig. 3. Construction of CFP-tree. (a) After scanning first transaction (b) After scanning second transaction (c) After scanning third transaction and (d) After scanning entire database.

The CFP-tree explicitly maintains *tids* of each transaction at the nodes. As a result, one can argue that the structure of a CFP-tree may not be memory efficient. We argue that the CFP-tree achieves the memory efficiency by keeping such transaction information only at the *tail*-nodes and avoiding the support count field at the each node. It was also shown in the literature [8] such trees are memory efficient. Moreover, keeping the *tid* information in tree can also be found in the literature for efficient mining of frequent patterns [19].

4.3 Mining CFP-tree

Even though both CFP-tree and FP-tree arrange items in support-descending order, we can not directly apply the FP-growth mining on a CFP-tree. The reason is that, CFP-tree does not maintain the support count at each node, and it handles the *tid*-lists at *tail*-nodes. Therefore, we devise an alternative pattern growth-based bottom-up mining technique that can handle the additional features of CFP-tree.

The basic operations in mining CFP-tree involves (*i*) counting length-1 chronic-frequent items, (*ii*) constructing the prefix-tree for each chronic-frequent patterns, and (*iii*) constructing the conditional tree from each prefix-tree. The CFP-list provides the length-1 chronic-frequent items. Before discussing the prefix-tree construction process we explore the following important property and lemma of a CFP-tree.

Property 4. A tail-node in a CFP-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 2. *Let* $Z = \{a_1, a_2, \cdots, a_n\}$ *be a path in a CFP-tree where node* a_n *is the tail-node carring the tid-list of the path. If the tid-list is pushed-up to the* *node* a_{n-1}*, then* a_{n-1} *maintains the occurrence information of the path* $Z' = \{a_1, a_2, \cdots, a_{n-1}\}$ *for the same set of transactions in the tid-list without any loss.*

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

Choosing the last item 'i' in the CFP-list, we construct its prefix-tree, say PT_i, with the prefix sub-paths of nodes labeled 'i' in the CFP-tree. Since 'i' is the bottom-most item in the CFP-list, each node labeled 'i' in the CFP-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. It facilitates the calculation of *support* and *periodic-recurrence* for each item in the CFP-list of PT_i. Moreover, to enable the construction of the prefix-tree for the next item in the CFP-list, based on Lemma 2, the *tid*-lists are pushed-up to respective parent nodes in the original CFP-tree and in PT_i as well. All nodes of i in the CFP-tree and i's entry in the CFP-list are deleted thereafter. Figure 4 (a) shows the prefix-tree of 'f', i.e., PT_f. Figure 4(c) shows the status of the CFP-tree of Figure 3(d) after removing the bottom-most item 'f'.

The conditional tree CT_i for PT_i is constructed by removing all non-chronic-frequent items from the PT_i. If the deleted node is a tail-node, its *tid*-list is pushed-up to its parent node. Figure 4(b) shows the conditional tree for 'f', CT_f constructed from the PT_f of Figure 4(a). The contents of the temporary array for the bottom item 'j' in the CFP-list of CT_i represent the TID^{ij} (i.e., the set of all *tids* where item i and j occur together in the database). Therefore, it is rather simple calculation to compute $S(ij)$ and $PR(ij)$ from TID^{ij}. If $S(ij) \geq minSup$ and $PR(ij) \geq minPR$, then the pattern 'ij' is generated as a chronic-frequent pattern. The same process of creating prefix-tree and its corresponding conditional tree is repeated for further extensions of 'ij'. The whole process of mining for each item is repeated until CFP-list$\neq \emptyset$.

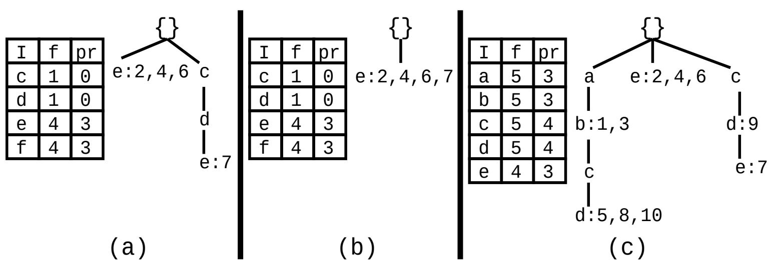

Fig. 4. Prefix-tree and conditional tree construction with CFP-tree. (a) Prefix-tree for 'f' (b) Conditional tree for 'f' and (c) CFP-tree after removing item 'f'.

5 Experimental Results

Since there is no existing approach to discover chronic-frequent patterns, we only investigate the performance of CFP-growth algorithm. In addition, we also discuss the usefulness of proposed patterns using a real-world database.

The CFP-growth algorithm was written in Java and run on Ubuntu on a 2.66 GHz machine having 4GB of memory. The databases used for our experiments are as follows:

- **T10I4D100K and T10I4D1000K databases.** These two databases are synthetic transactional databases generated using the procedure given in [7]. The T10I4D100K dataset contains 100,000 transactions and 941 distinct items. The T10I4D1000K contains 983,155 transactions with 30,387 items.
- **Shop-14 database.** A Czech company has provided clickstream data of seven internet shops in ECML/PKDD 2005 Discovery challenge [21]. In this paper, we have considered the click stream data of product categories visited by the users in "Shop 14" (www.shop4.cz), and created a transactional database with each transaction representing the set of web pages visited by the people at a particular *minute interval.* The transactional database contains 59,240 transactions (i.e., 41 days of page visits) and 138 product categories (or items).
- **BMS-WebView-1 database.** This is a real-world database containing 59,602 transactions with 497 items [22].
- **Kosarak database.** This is a very large real-world database containing 990,002 transactions with 41,270 distinct items.

The *Kosarak* and *BMS-WebView-1* databases have been downloaded from the Frequent Itemset MIning (FIMI) repository (`http://fimi.ua.ac.be/data/`).

5.1 Generation of Chronic-Frequent Patterns

Table 2 shows the different *minSup*, *maxPrd* and *minPR* values used for finding chronic-frequent patterns in T10I4D100K, Shop-14 and BMS-WebView-1 datasets. It can be observed that we have we have set low *minSup* and *minPR* values to discover the patterns involving both frequent and relatively infrequent (or rare) items.

Table 2. The user-defined *minSup*, *maxPrds* and *minPR* values in different datasets. The Greek letters α, β and γ represent the *minSup*, *maxPrd* and *minPR* thresholds, respectively.

Datasets	minSup (α)			maxPrd (β)			minPR (γ)		
	α_1	α_2	α_3	β_1	β_2	β_3	γ_1	γ_2	γ_3
T10I4D100K	0.1%	0.3%	0.5%	1%	5%	10%	0.1%	0.2%	0.3%
Shop-14	0.1%	0.3%	0.5%	1%	5%	10%	0.1%	0.2%	0.3%
BMS-WebView-1	0.1%	0.3%	0.5%	1%	5%	10%	0.1%	0.2%	0.3%

Table 3. The number of chronic-frequent patterns generated at different *minSup*, *maxPrd* and *minPR* threshold values.

Dataset	α	γ_1 β_1	γ_1 β_2	γ_1 β_3	γ_2 β_1	γ_2 β_2	γ_2 β_3	γ_3 β_1	γ_3 β_2	γ_3 β_3
T10I4D100K	α_1	20077	26384	26511	10115	12643	12644	3768	4432	4432
	α_2	4476	4476	4476	4476	4476	4476	3768	4432	4432
	α_3	1069	1069	1069	1069	1069	1069	1069	1069	1069
Shop-14	α_1	1215	27320	30382	4268	6377	6537	2428	3000	3058
	α_2	3089	3089	3089	3089	3089	3089	2428	3000	3058
	α_3	1244	1244	1244	1244	1244	1244	1244	1244	1244
BMS-WebView-1	α_1	1410	3227	3680	572	777	796	362	431	432
	α_2	435	435	435	435	435	435	362	431	432
	α_3	201	201	201	201	201	201	201	201	201

Table 4. Runtime requirements of CFP-growth. The runtime is expressed in seconds.

Dataset	α	γ_1 β_1	γ_1 β_2	γ_1 β_3	γ_2 β_1	γ_2 β_2	γ_2 β_3	γ_3 β_1	γ_3 β_2	γ_3 β_3
T10I4D100K	α_1	207	263	268	105	126	136	37	44	44
	α_2	45	45	45	45	45	45	37	43	43
	α_3	19	19	19	19	19	19	19	19	19
Shop-14	α_1	121	220	303	42	63	65	24	30	32
	α_2	30	30	30	30	30	30	24	32	33
	α_3	14	14	14	14	14	14	14	14	14
BMS-WebView-1	α_1	103	257	287	97	189	234	182	166	156
	α_2	78	91	251	58	79	165	38	43	98
	α_3	71	92	124	55	69	83	20	34	79

Table 3 shows the number of chronic-frequent patterns generated in different datasets at various *minSup*, *maxPrd* and *minPR* threshold values. The following observations can be drawn from this table. (*i*) At a fixed *maxPrd* and *minPR*, increase in *minSup* has decreased the number of chronic-frequent patterns. (*ii*) At a fixed *minSup* and *minPR*, increase in *maxPrd* has increased the number of chronic-frequent patterns. It is because the occurrences of a frequent pattern which were earlier (i.e., at low *maxPrd* threshold) considered as aperiodic have been considered as periodic with in *maxPrd* threshold. (*iii*) At a fixed *minSup* and *maxPrd*, increase in *minPR* has decreased the number of chronic-frequent patterns. The reason is that many frequent patterns were unable to occur periodically for longer time durations in a database.

Table 4 shows the runtime taken by CFP-growth to discover chronic-frequent patterns in T10I4D100K, Shop-14 and BMS-WebView-1 datasets. The runtime involves both the construction and mining of CFP-tree. The changes on the *minSup*, *minPR* and *maxPrd* shows the similar effect on runtime consumption as that of the generation of chronic-frequent patterns. It can be observed that the proposed algorithm discovers the complete set of chronic-frequent patterns at a reasonable runtime even at low *minSup* and *minPR* thresholds.

Table 5 shows some of the chronic-frequent patterns discovered in Shop-14 dataset at $minSup = 1\%$, $maxPrd = 5\%$ and $minPR = 1\%$. It can be observed that none of these patterns were appearing periodically throughout the database, however, there were periodically appearing in distinct subsets of the database. Using the approach discussed in [8], we have made an effort to find periodic-frequent patterns with $minSup = 1\%$ and $maxPrd = 5\%$. Unfortunately, no pattern was discovered at these threshold values. It because all frequent patterns have failed to reappear at very short intervals throughout the database. Thus, the proposed model was able to discover useful patterns.

Table 5. The chronic-frequent patterns discovered in Shop-14 dataset.

Chronic-frequent patterns	Range of *tids* containing the pattern
{{TV's}, {Analog camcorders}}	[9,4447], [6591,15843], [16964,25508][26649,32654]
{{Speakers for home cinemas}, {Home cinema systems-components}}	[18, 5970], [7971, 11473], [18905, 24096]
{{Washer dryers}, {Refrigerators, freezers, show cases}, {built-in ovens, hobs, grills}}	[4,4655], [13824, 19589], [40232, 45721]
{{Built-in dish washers}, {Refrigerators, freezers, show cases}}	[13639,19544], [48495, 53310]

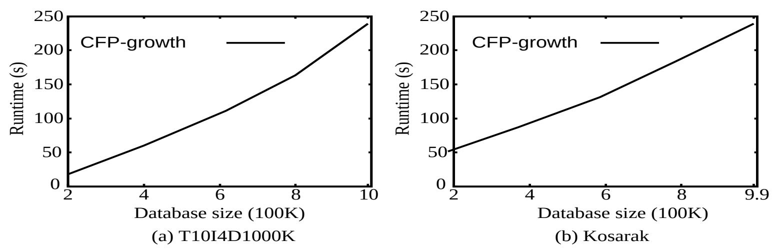

Fig. 5. Scalability of CFP-growth. (a) T10I4D1000K dataset and (b) Kosarak dataset.

5.2 The Scalability Test

We study the scalability of our CFP-growth algorithm on execution time by varying the number of transactions in $T10I4D1000K$ and $Kosarak$ datasets. In the literature, these two datasets were widely used to study the scalability of algorithms [23, 8]. The experimental setup was as follows. Each dataset was divided into five portions with 0.2 million transactions in each part. Then we investigated the performance of CFP-growth after accumulating each portion with previous parts. For each experiment, we set $minSup = 10\%$, $maxPrd = 1\%$ and $minPR = 10\%$.

Figure 5 (a) and (b) respectively show the runtime requirements of CFP-growth on the T10I4D1000K and Kosark datasets with the increase of dataset size. It is clear from the graphs that as the database size increases, overall tree construction and mining time increases. However, CFP-growth shows stable performance of about linear increase of runtime with respect to the database size. Therefore, it can be observed from the scalability test that CFP-growth can mine the patterns over large databases and distinct items with considerable amount of runtime.

6 Conclusions and Future Work

In this paper, we have investigated the partial periodic behavior of a frequent pattern in a transactional database, and proposed a practicable model to discover a class of user-interest-based patterns known as chronic-frequent patterns. We have provided a CFP-tree, a highly compact tree structure to capture the database content, and a pattern-growth technique to mine the complete set of chronic-frequent patterns. The experimental results demonstrate that our CFP-growth can be runtime efficient, and highly scalable as well.

As a part of future work, we would like to extend our work to improve the performance of association rule-based recommender systems. Furthermore, it is interesting to investigate the chronic behavior of the patterns in time-series databases, sequential databases, and data streams.

References

1. J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases.," in *KDD*, pp. 214–218, 1998.
2. J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *ICDE*, pp. 106–115, 1999.
3. R. Yang, W. Wang, and P. Yu, "Infominer+: mining partial periodic patterns with gap penalties," in *ICDM*, pp. 725–728, 2002.
4. C. Berberidis, I. Vlahavas, W. Aref, M. Atallah, and A. Elmagarmid, "On the discovery of weak periodicities in large time series," in *PKDD*, pp. 51–61, 2002.
5. H. Cao, D. Cheung, and N. Mamoulis, "Discovering partial periodic patterns in discrete data sequences," in *Advances in Knowledge Discovery and Data Mining*, vol. 3056, pp. 653–658, 2004.
6. W. G. Aref, M. G. Elfeky, and A. K. Elmagarmid, "Incremental, online, and merge mining of partial periodic patterns in time-series databases," *IEEE TKDE*, vol. 16, pp. 332–342, Mar. 2004.
7. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD*, pp. 207–216, 1993.
8. S. K. Tanbeer, C. F. Ahmed, B. S. Jeong, and Y. K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *PAKDD*, pp. 242–253, 2009.
9. R. U. Kiran and P. K. Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *DEXA (2)*, pp. 194–208, 2010.

10. A. Surana, R. U. Kiran, and P. K. Reddy, "An efficient approach to mine periodic-frequent patterns in transactional databases," in *PAKDD Workshops*, pp. 254–266, 2011.
11. K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-k periodic-frequent pattern from transactional databases without support threshold," in *Advances in Information Technology*, vol. 55 of *Communications in Computer and Information Science*, pp. 18–29, 2009.
12. J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions," *DMKD*, vol. 14, no. 1, 2007.
13. C. M. Antunes and A. L. Oliveira, "Temporal data mining: An overview," in *Workshop on Temporal Data Mining, KDD*, 2001.
14. B. Özden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *ICDE*, pp. 412–421, 1998.
15. S.-S. Chen, T. C.-K. Huang, and Z.-M. Lin, "New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports," *J. Syst. Softw.*, vol. 84, pp. 1638–1651, Oct. 2011.
16. R. U. Kiran and P. K. Reddy, "An alternative interestingness measure for mining periodic-frequent patterns," in *Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I*, pp. 183–192, 2011.
17. R. U. Kiran and M. Kitsuregawa, "Discovering quasi-periodic-frequent patterns in transactional databases," in *Big Data Analytics - Second International Conference, BDA 2013, Mysore, India, December 16-18, 2013, Proceedings*, pp. 97–115, 2013.
18. J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Min. Knowl. Discov.*, vol. 8, pp. 53–87, Jan. 2004.
19. M. J. Zaki and C.-J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, pp. 462–478, Apr. 2005.
20. X. Zhi-jun, C. Hong, and C. Li, "An efficient algorithm for frequent itemset mining on data streams," in *Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining*, vol. 4065, pp. 474–491, 2006.
21. "Weblog dataset." `http://web.archive.org/web/20070713202946rn_1/lisp.vse.cz/challenge/CURRENT/`.
22. Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms," in *KDD '01*, pp. 401–406, 2001.
23. R. U. Kiran and P. K. Reddy, "An alternative interestingness measure for mining periodic-frequent patterns," in *DASFAA (1)*, pp. 183–192, 2011.