Discovering Periodic Patterns in Non-uniform Temporal Databases

R. Uday Kiran[1](✉), J.N. Venkatesh[2], Philippe Fournier-Viger[3], Masashi Toyoda[1], P. Krishna Reddy[2], and Masaru Kitsuregawa[1,4]

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

[2] Kohli Center on Intelligent Systems (KCIS), International Institute of Information Technology Hyderabad, Hyderabad, India
jn.venkatesh@research.iiit.ac.in, pkreddy@iiit.ac.in

[3] Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
philfv8@yahoo.com

[4] National Institute of Informatics, Tokyo, Japan

Abstract. A temporal database is a collection of transactions, ordered by their timestamps. Discovering periodic patterns in temporal databases has numerous applications. However, to the best of our knowledge, no work has considered mining periodic patterns in temporal databases where items have dissimilar *support* and *periodicity*, despite that this type of data is very common in real-life. Discovering periodic patterns in such non-uniform temporal databases is challenging. It requires defining (*i*) an appropriate measure to assess the periodic interestingness of patterns, and (*ii*) a method to efficiently find all periodic patterns. While a pattern-growth approach can be employed for the second sub-task, the first sub-task has to the best of our knowledge not been addressed. Moreover, how these two tasks are combined has significant implications. In this paper, we address this challenge. We introduce a model to assess the periodic interestingness of patterns in databases having a non-uniform item distribution, which considers that periodic patterns may have different *period* and minimum number of cyclic repetitions. Moreover, the paper introduces a pattern-growth algorithm to efficiently discover all periodic patterns. Experimental results demonstrate that the proposed algorithm is efficient and the proposed model may be utilized to find prior knowledge about event keywords and their associations in Twitter data.

Keywords: Data mining · Periodic pattern · Non-uniform temporal database

1 Introduction

Temporal databases are commonly used in many domains. A temporal database is a collection of transactions, ordered by their timestamps. A temporal database is said to be non-uniform if it contains items with dissimilar *support*

Table 1. Some tweets produced during GEJE

Timestamp	Tweets
1301575750	#Discrimination n demagoguery 4 foreigners, mainly #Asian s by #Japanese in #Sendai. #earthquake #jishin http://htn.to/rhGtmd
1301575750	shhhhh dont tell @jimcramer El-Erian says recent Japanese earthquake is NOT like Kobe, won't have same V shaped recovery - Reuters #newsmkr
1301583131	Some people will never be able to go home. They lost their homes in the tsunami. #VOAAsiachat1
1301583579	Social media had a critical role in Japan during/after the quake/tsunami. Somewhat else can better assess overall than me. #VOAAsiachat1

and *periodicity*. Non-uniform temporal data is naturally produced in many real-world situations. For instance, disasters such as earthquakes and tsunami happen at irregular time intervals. Twitter data related to these disasters is thus non-uniform. For example, Table 1 shows a part of a temporal database generated from the tweets produced during the Great East Japan Earthquake (GEJE), which occurred on the 11th March 2011.

Discovering patterns in temporal databases is challenging because they not only allow time gaps between consecutive transactions, but also to have multiple transactions with the same timestamp. An important type of patterns that can be extracted from temporal databases is (Partial) periodic patterns. A periodic pattern is something persistent and predictable that appears in a database. Finding periodic patterns is thus useful to understand the data. For example, it was revealed in our present study on Twitter data related to the GEJE that over 80% of the event keywords found by a supervised event detection algorithm [1] can also be discovered as periodic patterns. The proposed study thus may be used as an unsupervised learning technique to generate some prior knowledge about event keywords and their associations in Twitter data.

The task of finding periodic patterns has two important sub-tasks: (*i*) determining the periodic interestingness of patterns, and (*ii*) finding all periodic patterns in a given database. While a variation of pattern-growth algorithms could be employed for the second sub-task, the first sub-task is non-trivial because of the following reasons:

1. Current periodic pattern models [2–4] do not take into account the information about the temporal occurrences of items in a dataset.
2. Since a temporal database allows transactions to share a common timestamp, the periodic interestingness of a pattern has to be determined by taking into account not only its *support*, but also its inter-arrival times in a database. Unfortunately, current measures assess the interestingness of a pattern by only taking its *support* into account. We need to investigate new measure(s) to assess the interestingness of patterns by taking into account both their *support* and *periodicity* in a database.

Moreover, how to combine the two aforementioned tasks has significant implications.

This paper addresses this challenge. It presents a model to discover periodic patterns in non-uniform temporal databases. The proposed model lets the user specify a different *maximum inter-arrival time* ($MIAT$) for each item. Thus, different patterns may satisfy different *period* depending on their items' $MIAT$ values. A new measure, *Relative Periodic-Support* (RPS), is proposed to determine the periodic interestingness of a pattern in a database. Unlike existing *support*-based measures, the proposed measure assess the interestingness of a pattern by taking into account its number of cyclic repetitions in the database. An inter-arrival time of a pattern is considered periodic (or cyclic) if it is no more than *period*. This measure satisfies the *null-invariant property* [5]. Thus, the usage of item specific $MIAT$ values and RPS allows the proposed model to capture the non-uniform distribution of items in a database. We also propose a pattern-growth algorithm that discovers the complete set of periodic patterns. Experimental results demonstrate that the proposed algorithm is efficient. We also demonstrate the usefulness of the proposed model by finding various event keywords and their associations in disaster related Twitter data.

The rest of paper is organized as follows. Section 2 describes the related work. Section 3 describes the proposed periodic pattern model. Section 4 introduces our algorithm to find all periodic patterns in a database. Section 5 reports on experimental results. Finally, Sect. 6 concludes the paper with future research directions.

2 Related Work

Frequent pattern mining is an important data mining task. Several *support* related measures have been discussed to determine the interestingness of a pattern in a transactional database. Each measure has a selection bias that justifies the significance of a knowledge pattern. As a result, there exists no universally acceptable best measure to judge the interestingness of a pattern in any given database. Researchers have proposed criteria to select an interestingness measure based on user and/or application requirements [5]. Recently, measures that satisfy the *null-invariant property* have became popular for finding frequent patterns. The reason is that this property guarantees finding genuine correlation patterns that are not influenced by object co-absence in a database. Unfortunately, current measures cannot be used to determine the periodic interestingness of a pattern in temporal databases. This is because these measures only take the *support* into account and completely ignore the temporal occurrence behavior of patterns in databases. We introduce a new null-invariant measure that assess the interestingness of a pattern by taking into account both the *support* and temporal occurrence information of patterns.

Periodic patterns are an important class of regularities that exist in a time series data. Since it was first introduced in [2], the problem of finding these patterns has received a great deal of attention [4]. A major limitation of these studies is that they consider time series as a symbolic sequence and ignore the temporal occurrence information about events in a series.

Similar to our problem, the mining of full periodic-frequent patterns in a transactional database has been studied in [6–8]. This problem of finding full periodic-frequent patterns greatly simplifies the design of the model because there is no need of any measure to determine the partial periodic interestingness of a pattern. More important, these studies also consider transactional database as a symbolic sequence of transactions (or itemsets) and ignore the temporal occurrence information of the transactions in a database. To the best of our knowledge, this is the first study that considers the problem of finding (partial) periodic patterns by taking into account the temporal occurrence information of the transactions in a database.

3 Proposed Model

Let $I = \{i_1, i_2, \cdots, i_n\}$ be the set of 'n' items appearing in a database. A set of items $X \subseteq I$ is called an itemset (or **a pattern**). A pattern containing k items is called a k-pattern. The length of this pattern is k. A transaction is a triplet $tr = (tid, ts, Y)$, where tid represents the transactional identifier, $ts \in \mathbb{R}$ represents the transaction time (or timestamp) and Y is an itemset. A temporal database TDB is an ordered set of transactions, i.e. $TDB = \{tr_1, tr_2, \cdots, tr_m\}$, where $m = |TDB|$ represents the database size (the number of transactions). Let ts_{min} and ts_{max} denote the minimum and maximum timestamps in TDB, respectively. For a transaction $tr = (tid, ts, Y)$, such that $X \subseteq Y$, it is said that X occurs in tr and such a timestamp is denoted as ts^X. Let $TS^X = (ts_a^X, ts_b^X, \cdots, ts_c^X)$, $a \leq b \leq c$, be the **ordered list of timestamps** of transactions in which X appears in TDB. The number of transactions containing X in TDB (i.e., the size of TS^X) is defined as the *support* of X and denoted as $sup(X)$. That is, $sup(X) = |TS^X|$.

Example 1. Table 2 shows a temporal database with $I = \{abcdefg\}$. The set of items 'a' and 'b,' i.e., 'ab' is a pattern. This pattern contains 2 items. Therefore, it is a 2-pattern. The length of this pattern is 2. In the first transaction, $tr_1 = (100, 1, ab)$, '100' represents the tid of the transaction, '1' represents the timestamp of this transaction and 'ab' represents the items occurring in this transaction. Other transactions in this database follow the same representation. The size of the database is $m = 12$. The minimum and maximum timestamps in this database are 1 and 14, respectively. Therefore, $ts_{min} = 1$ and $ts_{max} = 14$. The pattern 'ab' appears in the transactions whose timestamps are 1, 3, 6, 8, 10, 11 and 12. Therefore, $TS^{ab} = \{1, 3, 6, 8, 10, 11, 12\}$. The *support* of '$ab$,' i.e., $sup(ab) = |TS^{ab}| = 7$.

Definition 1 ***(Period of a pattern X).*** *Let $MIAT(i_j)$ be the user-defined* maximum inter-arrival time *(MIAT) specified for an item $i_j \in I$. The period of a pattern X, denoted as $PER(X)$, represents the largest $MIAT$ value of all items in X. That is, $PER(X) = max(MIAT(i_j) | \forall i_j \in X)$. The items' $MIAT$ values can also be expressed in percentage of $(ts_{max} - ts_{min})$.*

Table 2. Running example: temporal database

tid	ts	items	tid	ts	items	tid	ts	items	tid	ts	items
100	1	*ab*	103	4	*cd*	106	8	*abcd*	109	11	*abf*
101	3	*acdg*	104	6	*abcd*	107	9	*ce*	110	12	*abcd*
102	3	*abef*	105	7	*efg*	108	10	*abef*	111	14	*acdeg*

Example 2. Let the $MIAT$ values for the items a, b, c, d, e, f and g be 2, 2, 2, 2, 3, 4 and 4, respectively. The *period* of the pattern 'ab,' i.e., $PER(ab) = max(2, 2) = 2$.

The usage of items' $MIAT$ values enable us to achieve the goal of having lower *periods* for patterns that only involve frequent items, and having higher *periods* for patterns that involve rare items. The items' $MIAT$ values may be derived using the *period* determining functions, such as Fast Fourier Transformations (FFTs) and auto-correlation.

Definition 2 ***(Periodic occurrence of a pattern X).*** *Let ts_j^X, $ts_k^X \in TS^X$, $1 \leq j < k \leq m$, denote any two consecutive timestamps in TS^X. The time difference between ts_k^X and ts_j^X is referred as* ***an inter-arrival time*** *of X, and denoted as iat^X. That is, $iat^X = ts_k^X - ts_j^X$. Let $IAT^X = \{iat_1^X, iat_2^X, \cdots, iat_k^X\}$, $k = sup(X) - 1$, be the list of all inter-arrival times of X in TDB. An inter-arrival time of X is said to be* ***periodic*** *(or cyclic) if it is no more than $PER(X)$. That is, a $iat_i^X \in IAT^X$ is said to be* ***periodic*** *if $iat_i^X \leq PER(X)$.*

Example 3. The pattern 'ab' has initially appeared at the timestamps of 1 and 3. The difference between these two timestamps gives an inter-arrival time of 'ab.' That is, $iat_1^{ab} = 2\ (= 3 - 1)$. Similarly, other inter-arrival times of 'ab' are $iat_2^{ab} = 3\ (= 6-3)$, $iat_3^{ab} = 2\ (= 8-6)$, $iat_4^{ab} = 2\ (= 10-8)$, $iat_5^{ab} = 1\ (= 11-10)$ and $iat_6^{ab} = 1\ (= 12 - 11)$. Therefore, $IAT^{ab} = \{2, 3, 2, 2, 1, 1\}$. If $PER(ab) = 2$, then iat_1^{ab}, iat_3^{ab}, iat_4^{ab}, iat_5^{ab} and iat_6^{ab} are considered as the periodic occurrences of 'ab'. The iat_2^{ab} is considered as an aperiodic occurrence of 'ab' because $iat_2^{ab} \not\leq PER(ab)$.

Definition 3 ***(Relative periodic-support of a pattern X).*** *Let $\widehat{IAT^X}$ be the set of all inter-arrival times in IAT^X that have $iat^X \leq PER(X)$. That is, $\widehat{IAT^X} \subseteq IAT^X$ such that if $\exists iat_k^X \in IAT^X : iat_k^X \leq PER(X)$, then $iat_k^X \in \widehat{IAT^X}$. The relative periodic-support of X, denoted as $RPS(X) = \frac{|\widehat{IAT^X}|}{|IAT^{i_j}|}$, where i_j is an item that has the lowest support and maximum $MIAT$ value among all items in X. This measure satisfies the null-invariant property [5].*

Example 4. Continuing with the previous example, $\widehat{IAT^{ab}} = \{2, 2, 2, 1, 1\}$, the item '$b$' in the pattern '$ab$' has the lowest *support* and maximum $MIAT$ value. Therefore, the *relative periodic-support* of 'ab,' i.e., $RPS(ab) = \frac{|\widehat{IAT^{ab}}|}{|IAT^b|} = \frac{5}{6} = 0.83$.

For brevity, we call $\widehat{IAT^X}$ as ***periodic-frequency***. The *periodic-frequency* determines the number of cyclic repetitions of a pattern in the data. The proposed measure enables us to achieve the goal of specifying a higher number of cyclic repetitions for patterns that only involve frequent items, and a lower number of cyclic repetitions for patterns that involve rare items. For a pattern X, $RPS(X) \in [0,1]$. If all inter-arrival times of X are more than $PER(X)$, then $RPS(X) = 0$. In other words, X is an irregular pattern. If all inter-arrival times of X are within $PER(X)$, then $RPS(X) = 1$. In other words, X is a full periodic pattern.

In the proposed model, we have considered an inter-arrival time of X as interesting if $iat^X \leq PER(X)$. However, our model is flexible and allows other ways to consider an inter-arrival time of a pattern as interesting. For instance, we can consider an inter-arrival time of a pattern as interesting if $iat^X \leq PER(X) \pm \Omega$, where $\Omega > 1$ is a constant that denotes time tolerance. We stick to the above definition for brevity.

Definition 4 ***(Periodic pattern X).*** *The pattern X is a periodic pattern if $RPS(X) \geq minRPS$, where $minRPS$ is the user-specified minimum relative periodic-support.*

Example 5. Continuing with the previous example, if the user-specified $min\ RPS = 0.6$, then 'ab' is a periodic pattern because $RPS(ab) \geq minRPS$.

Definition 5 ***(Problem definition).*** *Given a temporal database (TDB), set of items (I), user-defined minimum interval times of the items (MIAT) and minimum relative periodic-support (minRPS), the problem of finding periodic patterns involve discovering all patterns in TDB that have* relative periodic-support *no less than minRPS.*

The periodic patterns generated by the proposed model satisfy the **convertible anti-monotonic property** [9].

Property 1. Let $Z = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq |I|$, be a pattern with $MIAT(i_1) \geq MIAT(i_2) \geq MIAT(i_k)$. If $Y \subset Z$ and $i_1 \in Y$, then $RPS(Y) \geq RPS(Z)$ as $\frac{|\widehat{IAT^Y}|}{|IAT^{i_1}|} \geq \frac{|\widehat{IAT^Z}|}{|IAT^{i_1}|}$.

4 Periodic Pattern-Growth Algorithm

In this section, we describe the proposed PP-growth algorithm that discovers the complete set of periodic patterns. Our algorithm involves the following two steps: (i) compress the database into a periodic pattern tree (PP-tree) and (ii) recursively mine the PP-tree to find all periodic patterns. Before we discuss these two steps, we describe the PP-tree structure.

Structure of PP-Tree Structure. A PP-tree has two components: a PP-list and a prefix-tree. The PP-list consists of each distinct *item* (i) with *minimum interval time* ($MIAT$), *support* (S), *periodic-frequency* (PF) and a pointer pointing to the first node in the prefix-tree carrying the item. The prefix-tree in a PP-tree resembles that of the prefix-tree in a FP-tree [10]. However, to capture both *support* and inter-arrival times of the patterns, the nodes in the PP-tree explicitly maintain the occurrence information for each transaction by keeping an occurrence timestamp list, called a ***ts*-list**. To achieve memory efficiency, only the last node of every transaction maintains the *ts*-list. We now explain the construction and mining of PP-tree.

Construction of PP-tree. The procedure for constructing a PP-tree is shown in Algorithm 1. We illustrate the working of this algorithm using the database

Algorithm 1. Construction of PP-Tree(TDB: Time series database, I: Set of items, $MIAT$: minimum interval time, $minRPS$: minimum relative periodic-support)

```
1: Insert all items in $TDB$ into the PP-list with their $MIAT$ values. Set the support
   and periodic-frequency values of all these items to 0. The timestamps of the last
   occurring transactions of all items in the PP-list are explicitly recorded for each
   item in a temporary array, called $ts_l$.
2: Let $t = \{ts_{cur}, X\}$ denote the current transaction with $ts_{cur}$ and $X$ representing
   the timestamp and pattern, respectively.
3: for each transaction $t \in TDB$ do
4:    for each item $i \in X$ do
5:       $S(i) + +;$
6:       if $((ts_l(i) \neq 0) \&\& (ts_{cur} - ts_l(i)) \leq MIAT(i))$ then
7:          $PF(i) + +;$
8:       $ts_l(i) = ts_{cur};$
9: All items in PP-list are sorted in ascending order of their $MIAT$ values. The items
   having a common $MIAT$ value are sorted in descending order of their support.
10: Measure the $RPS$ value for the bottom most item in the PP-list. If the $RPS$
    value of this item is less than $minRPS$, then prune this item from the PP-list
    and repeat the same step for the next bottom most item in the PP-list. Stop this
    pruning process once the $RPS$ value of the bottom most item in PP-list is no less
    than $minRPS$. Let $CI$ denote this sorted list of items.
11: Create a root node in the prefix-tree, $T$, and label it as "null."
12: for each transaction $t \in TDB$ do
13:    Sort the items in $X$ according to the order of $CI$. Let the sorted candidate
       item list in $t$ be $[p|P]$, where $p$ is the first item and $P$ is the remaining list.
       Call $insert\_tree([p|P], ts_{cur}, T)$, which is performed as follows. If $T$ has a child
       $N$ such that $N.item\text{-}name \neq p.item\text{-}name$, then create a new node $N$, Let its
       parent link be linked to $T$. Let its node-link be linked to nodes with the same
       item-name via the node-link structure. Remove $p$ from $P$. If $P$ is nonempty, call
       $insert\_tree(P, ts_{cur}, N)$ recursively; else add $ts_{cur}$ to the leaf node.
```

Algorithm 2. PP-growth($Tree$, α)

```
1: for each $a_i$ in the header of $Tree$ do
2:     if $\frac{PF(a_i)}{S(a_i)-1} \geq minRPS$ then
3:         Generate pattern $\beta = a_i \cup \alpha$. Traverse $Tree$ using the node-links of β, and
           construct an array, $TS^{\beta}$, which represents the timestamps at which β has
           appeared in $TDB$. Construct β's conditional pattern base and β's conditional
           PP-tree $Tree_{\beta}$ by calling calculateRPS(β, $TS^{\beta}$, $MIAT(a_i)$). The calculateRPS
           function calculates the *periodic-frequency* of β from $TS^{\beta}$, and returns $RPS$
           value by dividing the *periodic-frequency* with $S(a_i) - 1$.
4:         if $Tree_{\beta} \neq \emptyset$ then
5:             call PP-growth($Tree_{\beta}$, β);
6:     Remove $a_i$ from the $Tree$ and push the $a_i$'s ts-list to its parent nodes.
```

i	MIAT	S	PF	ts$_l$
a	2	0	0	0
b	2	0	0	0
c	2	0	0	0
d	2	0	0	0
e	3	0	0	0
g	4	0	0	0
f	4	0	0	0

(a)

i	MIAT	S	PF	ts$_l$
a	2	1	0	1
b	2	1	0	1
c	2	0	0	0
d	2	0	0	0
e	3	0	0	0
g	4	0	0	0
f	4	0	0	0

(b)

i	MIAT	S	PF	ts$_l$
a	2	9	7	14
b	2	7	5	12
c	2	7	5	14
d	2	6	4	14
e	3	5	2	14
f	4	4	3	11
g	4	3	0	14

(c)

i	MIAT	S	PF
a	2	9	7
b	2	7	5
c	2	7	5
d	2	6	4
e	3	5	2
f	4	4	3
g	4	3	0

(d)

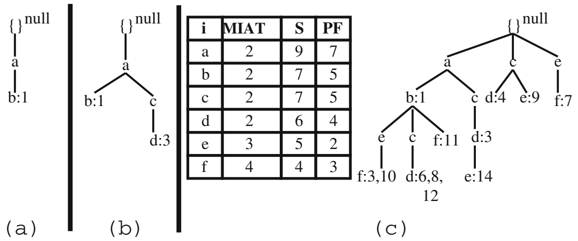
i	MIAT	S	PF
a	2	9	7
b	2	7	5
c	2	7	5
d	2	6	4
e	3	5	2
f	4	4	3

(e)

Fig. 1. Construction of PP-List. (a) Before scanning the database. (b) After scanning the first transaction. (c) After scanning the entire database. (d) Updated PP-list. (e) Final PP-list with sorted list of items

shown in Table 2. (**Please note that we ignore the *tid* information of transactions for brevity**).

For the construction of PP-list, we insert all items into the PP-list with their $MIAT$ values. The *support* and *periodic-frequency* of all these items are simultaneously set to 0. Figure 1(a) shows the PP-list generated before scanning the database (line 1 in Algorithm 1). The scan on the first transaction, "1:ab," updates the *support* and ts_l values of a and b to 1 and 1, respectively. Figure 1(b) shows the PP-list generated after scanning the first transaction. This process is repeated for other transactions in the database and PP-list is updated accordingly. Figure 1(c) shows the PP-list generated after scanning the entire database (lines 2 to 8 in Algorithm 1). The items in PP-list are sorted in ascending order of their $MIAT$ values. Items having a common $MIAT$ value are sorted in descending order of their *support* (to achieve memory efficiency). Figure 1(d) shows the sorted PP-list (line 9 in Algorithm 1). We calculate RPS for the item 'g,' which is the bottom-most item in the PP-list. As $RPS(g) \ngeq minRPS$, the item 'g' is pruned from the PP-list. Next, we calculate the RPS value for the item f, which is the current bottom-most item in the PP-list. As $RPS(f) \geq minRPS$, we consider f as a periodic 1-pattern and stop the process of pruning other aperiodic 1-patterns from the PP-list. Figure 1(e) shows the final PP-list after pruning some of the aperiodic 1-patterns whose supersets can never produce any



i	MIAT	S	PF
a	2	9	7
b	2	7	5
c	2	7	5
d	2	6	4
e	3	5	2
f	4	4	3

Fig. 2. Construction of PP-Tree. (a) After scanning the first transaction. (b) After scanning the second transaction. (c) After scanning the entire database

periodic pattern (line 10 in Algorithm 1). Let CI denote the sorted list of items in PP-list. That is, $CI = \{a, b, c, d, e, f\}$. Next, we create a root node in the prefix-tree of PP-tree, and label it as "null" (line 11 in Algorithm 1).

In the next step, we update the PP-tree by performing another scan on the database. The items in the first transaction, "1 : ab," are sorted in CI order and a first branch is constructed with two nodes $\langle a \rangle$ and $\langle b : 1 \rangle$, where 'a' is linked as a child of the root and 'b' is linked as the child node of 'a'. As 'b' represents the leaf node of the first transaction, this node carries the timestamp of 1. Figure 2(a) shows the PP-tree updated after scanning the first transaction. This process is repeated for the remaining transactions in the database and the PP-tree is updated accordingly. Figure 2(b) shows the PP-tree generated after scanning the second transaction. Figure 2(c). shows the PP-tree generated after scanning the entire database (lines 12 and 13 in Algorithm 1).

Recursive Mining of PP-Tree. The PP-tree is mined as follows. Start from length-1 pattern (as an initial suffix pattern). If the RPS value of this pattern satisfies the $minRPS$, then consider this pattern as a periodic item (or 1-pattern), construct its conditional pattern base (a sub-database, which consists of the set of prefix paths in the PP-tree with the suffix pattern), then construct its conditional PP-tree, and recursively mine that tree. Pattern-growth is achieved by concatening the suffix pattern with the periodic patterns generated from a conditional PP-tree. Next, the initial suffix pattern is pruned from the original PP-tree by moving its ts-lists to the corresponding parent nodes.

Algorithm 2 describes the procedure for finding periodic patterns in a PP-tree. We do not discuss this algorithm in detail as it is straightforward to understand. Mining the PP-tree is summarized in Table 3. It can be observed that conditional pattern bases have not been constructed for the item 'e,' because it is an aperiodic 1-pattern with $RPS(e) \not\geq minRPS$. The above bottom-up mining technique is efficient, because it shrinks the search space dramatically as the mining process progresses. Some of the improvements discussed for FP-growth [10] can be straight forward extended to PP-growth. We are unable to discuss these improvements due to the page limitation.

Table 3. Mining the PP-tree by creating conditional (sub-)pattern bases

item	*support*	*MIAT*	Conditional pattern base	Conditional PP-tree	Periodic patterns
f	4	4	$\{abe:3,10\}$, $\{ab:11\}$, $\{e:7\}$	$\langle e:3,7,10\rangle$	$\{ef:0.66\}$
e	5	3	–	–	–
d	6	2	$\{abc:6,8,12\}$, $\{ac:3,14\}$, $\{c:4\}$	$\langle c:3,4,6,8,12,14\rangle$	$\{cd:0.8\}$
c	7	2	$\{ab:6,8,12\}$, $\{a:3,14\}$	–	–
b	7	2	$\{a:1,3,6,8,10,11,12\}$	$\langle a:1,3,6,8,10,11,12\rangle$	$\{ab:0.83\}$

5 Experimental Results

Since there exists no algorithm to find periodic patterns in temporal databases, we only evaluate the proposed algorithm and show that our algorithm is memory and runtime efficient. We also show that PP-tree consumes less memory than the FP-tree for many databases. Finally, we discuss the usefulness of the proposed model by demonstrating that over 80% of the event keywords found by a supervised event detection system [1] in Twitter data can also be discovered as periodic patterns. (Similar to FP-growth [9,10], PP-growth also scales linearly with the increase of database size. Unfortunately, we are unable to present these results due to page limitation.)

The algorithms PP-growth and FP-growth are written in GNU C++ and run on a 2.66 GHz machine having 16 GB of memory. Ubuntu 14.04 is the operating system of our machine. The event detection system is written in python and java, and available for download at https://github.com/aritter/twitter_nlp. The experiments have been conducted using both synthetic (**T10I4D100K**) and real-world (**FAA-accidents** and **Twitter**) databases. The synthetic database, **T10I4D100K**, is generated by using the IBM data generator [11]. This data generator is widely used for evaluating association rule mining algorithms. The **T10I4D100K** database contains 870 items with 100,000 transactions. The **FAA-accidents** database is constructed from the accidents data recorded by the Federal Aviation Authority (FAA) from 1-January-1970 to 31-December-2014. Only categorical attributes have been taken into account while constructing the database. This database contains 9,290 items and 98,864 transactions. The Twitter database constitutes of 2,680,896 tweets collected from 10-march-2011 to 31-march-2011. These tweets are related to GEJE. We have created temporal database by considering top 4000 frequent english words.

Figure 3(a)–(c) present scatter plots about the inter-arrival times of items in the T10I4D100K, FAA-accidents and Twitter databases, respectively. The X-axis represents the items ranked in descending order of their *support* and

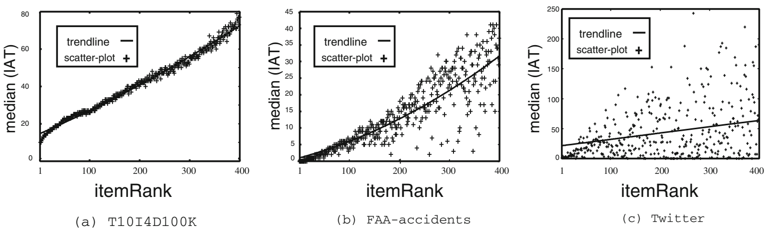


Fig. 3. The median of inter-arrival times of items in a database

Y-axis represents the median of inter-arrival times of an item in a database. The thick line in these figures denote the trend line. The equations of these trend lines and R^2 values are shown in Table 4. It can be observed from the trend lines that rare items not only have low *support*, but also have high inter-arrival times as compared against the frequent items. This experiment clearly demonstrates the importance of enabling every pattern to satisfy a different *period* and minimum number of cyclic repetitions to be a periodic pattern.

The performance of PP-growth has to be evaluated by varying the items' $MIAT$ values. Unfortunately, popular *period* identification functions (e.g. FFTs and auto-correlation) do not help us vary items' $MIAT$ values. In this context, we employ the following methodology to specify the items' $MIAT$ values. For each database, we use the equation of trend line as a reference, and specify the items' $MIAT$ values by multiplying the equation of the trend line with a constant β. That is, $MIAT(i_j) = \beta \times f(x)$, where $\beta \geq 1$ is a user-specified constant and $f(x)$ is the equation of trend line in which x denotes the rank of an item in support descending order.

Table 4. Trend line equations for various databases

Database	Equation of trend line ($f(x)$)	R^2
T10I4D100K	$y = 7.06E{-}05x^2 + 0.12x + 14.70$	0.9892
FAA-Accidents	$y = -9.67E{-}05x^2 + 0.04x + 1$	0.9122
Twitter	$y = 3.32E{-}06x^2 + 0.11x + 21.39$	0.0777

Figure 4(a)–(c) shows the number of periodic patterns generated for different $minRPS$ and β values in T10I4D100K, FAA-accidents and Twitter databases, respectively. The following two observations can be drawn from these figures: (i) Increase in β value may increase the number of periodic patterns. The reason is that higher β values tend to increase $MIAT$ values of items. (ii) Increase in $minRPS$ may decrease the number of periodic patterns. The reason is that increasing $minRPS$ increases the minimum number of cyclic repetitions necessary for a pattern to be a periodic pattern.

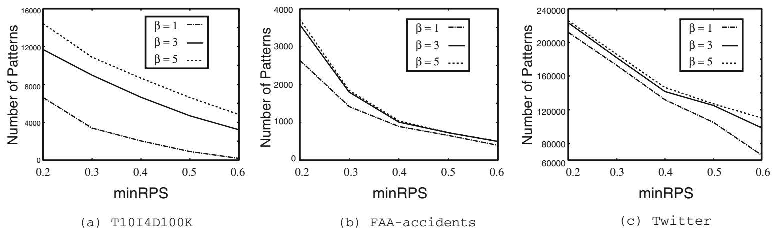


Fig. 4. The periodic patterns generated at different *minRPS* and β values

Figure 5(a)–(c) show the runtime requirements of PP-growth at different *minRPS* and β values in T10I4D100K, FAA-accidents and Twitter databases, respectively. It can be observed that varying the β and *minRPS* values has similar influence on runtime than on the generation of periodic patterns.

Table 5 lists the maximum memory usage of PP-tree and FP-tree on T10I4100K, FAA-accidents and Twitter databases, respectively. Both trees are constructed with every item in the database. It can be observed from the results that PP-tree consumes less memory than FP-tree if number of nodes in a *tree* exceed the database size, otherwise, PP-tree consumes more memory than FP-tree.

5.1 A Case Study: Evaluation of Periodic Patterns Discovered from Twitter Data

While investigating the usefulness of periodic patterns discovered from Twitter data, we have observed that many generated periodic 1-patterns (and their asso-

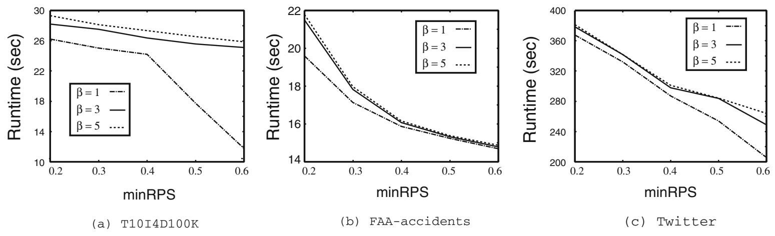


Fig. 5. Runtime requirements of PP-growth at different *minRPS* and β values

Table 5. Memory comparison of FP-tree and PP-tree

Data set	FP-tree (in MB)	PP-tree (in MB)	No. of nodes
T10I4D100K	10.906	8.561	714,739
FAA-accidents	5.898	4.801	316,935
Twitter	7.172	15.606	470,040

Table 6. Some of the interesting periodic patterns and tweets containing the patterns

Pattern	*RPS*	Tweets
still,death,alive,-rumors,celeb	0.83	S Yuko Yamaguchi (Hello Kitty) & Satoshi Tajiri (Pokemon) are still alive. Please stop spreading J-celeb death rumors. #earthquake
jishin,helpme,anpi,-hinan,nosg	0.86	twitter社より。統一のハッシュタグなどが発表になりました。情報の統合に協力しましょう。 #jishin: 地震一般に関する情報 #j_j_helpme :救助要請 #hinan :避難 #anpi :安否確認 #311care: 医療系被災者支援情報" #NOSG **(summary: users were tweeting the list of hashtags provided by Twitter for GEJE)**
tsunami,earthquake,-warning,massive,-widened	0.83	RT @bbcbreaking: #Tsunami warning is widened to incl rest of Pacific coast, incl #Australia and #South America massive #earthquake in #Japan

ciations) were interesting as they were referring to the event GEJE. This motivated us to study the following: (i) Do event keywords in Twitter exhibit periodic behavior? and (ii) If event keywords exhibit periodic behavior, then what would be their percentage? The significance of this study is that if we find many event keywords exhibiting periodic behavior, then one can use the proposed model as an unsupervised learning technique to derive some prior knowledge about event keywords and their associations in Twitter data.

Ritter et al. [1] discussed a supervised learning model to discover event keywords from tweets. We use this model for our experiment. This model annotates tweets using natural language processing techniques, generates a model from the training set of tweets and uses the model to extract event keywords from the test set of tweets. As the authors have already trained their model to identify event keywords in tweets, we have simply provided our Twitter data as the test set and extracted event keywords. A total of 325 event keywords have been extracted from the Twitter data. (We found that only 106 event keywords have appeared in top 500 frequent words. This clearly demonstrates that $frequency$ has less influence in determining a word as an event keyword.) When we compared these event keywords against the periodic 1-patterns generated at $\beta = 3$ and $minRPS = 0.6$, we found that 267 event keywords have been generated as periodic 1-patterns. In other words, 82.15% $(=\frac{267\times100}{325})$ of keywords have exhibited periodic behavior in Twitter data. This clearly demonstrates that periodic pattern mining can be used to find prior knowledge about event keywords and their associations in Twitter data. Table 6 lists some of the generated periodic patterns and their associated tweets.

6 Conclusions and Future Work

We have proposed a model to find periodic patterns in temporal databases. It enables every pattern to satisfy different *period* and minimum number of cyclic repetitions depending on its items. A null-invariant measure, *relative periodic-support*, was discussed to determine the periodic interestingness of a pattern in a database. A pattern-growth algorithm has also been presented to find periodic patterns. Experimental results show that the proposed model can find useful information and that the algorithm is efficient.

Our study has been confined to mining periodic patterns in a static temporal database. The method developed here can be extended to incremental mining of temporal databases.

References

1. Ritter, A., Mausam, Etzioni, O., Clark, S.: Open domain event extraction from twitter. In: KDD, pp. 1104–1112 (2012)
2. Han, J., Gong, W., Yin, Y.: Mining segment-wise periodic patterns in time-related databases. In: KDD, pp. 214–218 (1998)
3. Aref, W.G., Elfeky, M.G., Elmagarmid, A.K.: Incremental, online, and merge mining of partial periodic patterns in time-series databases. IEEE TKDE **16**(3), 332–342 (2004)
4. Chen, S.-S., Huang, T.C.-K., Lin, Z.-M.: New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports. J. Syst. Softw. **84**(10), 1638–1651 (2011)
5. Tan, P.-N., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: Knowledge Discovery and Data Mining, pp. 32–41 (2002)
6. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering periodic-frequent patterns in transactional databases. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS (LNAI), vol. 5476, pp. 242–253. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01307-2_24
7. Uday Kiran, R., Krishna Reddy, P.: Towards efficient mining of periodic-frequent patterns in transactional databases. In: Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) DEXA 2010. LNCS, vol. 6262, pp. 194–208. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15251-1_16
8. Venkatesh, J.N., Uday Kiran, R., Krishna Reddy, P., Kitsuregawa, M.: Discovering periodic-frequent patterns in transactional databases using all-confidence and periodic-all-confidence. In: Hartmann, S., Ma, H. (eds.) DEXA 2016. LNCS, vol. 9827, pp. 55–70. Springer, Cham (2016). doi:10.1007/978-3-319-44403-1_4
9. Pei, J., Han, J., Lakshmanan, L.V.: Pushing convertible constraints in frequent itemset mining. DMKD **8**, 227–252 (2004)
10. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. DMKD **14**(1), 55–86 (2007)
11. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216 (1993)