Expert Systems with Applications 42 (2015) 7882–7894

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Mining top-k frequent-regular closed patterns

Komate Amphawan [a,*], Philippe Lenca [b]

[a] Computational Innovation Laboratory, Informatics, Burapha University, Thailand
[b] Institut Mines Telecom, Telecom Bretagne, UMR CNRS 6285 Lab-STICC, France

ARTICLE INFO

Article history:
Available online 24 June 2015

Keywords:
Frequent pattern
Regular pattern
Closed pattern
Bit-vector

ABSTRACT

Frequent-regular pattern mining has attracted recently many works. Most of the approaches focus on discovering a complete set of patterns under the user-given support and regularity threshold constraints. This leads to several quantitative and qualitative drawbacks. First, it is often difficult to set appropriate support threshold. Second, algorithms produce a huge number of patterns, many of them being redundant. Third, most of the patterns are of very small size and it is arduous to extract interesting relationship among items. To reduce the number of patterns a common solution is to consider the desired number k of outputs and to mine the top-k patterns. In addition, this approach does not require to set a support threshold. To cope with redundancy and interestingness relationship among items, we suggest to focus on closed patterns and introduce a minimal length constraint. We thus propose to mine the *top-k frequent-regular closed patterns with minimal length*. An efficient single-pass algorithm, called TFRC-Mine, and a new compact bit-vector representation which allows to prune uninteresting candidate, are designed. Experiments show that the proposed algorithm is efficient to produce longer – non redundant – patterns, and that the new data representation is efficient for both computational time and memory usage.

1. Introduction

Frequent and regular (or periodic) pattern mining aims to detect the occurrence behavior of patterns *i.e.* whether a pattern occurs frequently and regularly, or mostly in a specific time interval in a database (Tanbeer, Ahmed, Jeong, & Lee, 2009). It can be applied to many fields such as genetic data analysis (Glynn, Chen, & Mushegian, 2006), manufacturing (Engler, 2008), behavior analysis of moving objects (Li, Ding, Han, Kays, & Nye, 2010), elderly daily habits' monitoring (Soulas, Lenca, & Thépaut, 2013), medical data (Khaleel, Dash, Choudhury, & Khan, 2015; Luth et al., 2008), game player behavior (Soulas & Lenca, 2015). Within this framework, a pattern is interesting if it is frequent and regular. The user has thus to specify a support threshold (as for association rules mining) and a regularity threshold (*i.e.* the maximum interval at which the pattern appear/disappear).

However, it is well-known that the classical frequent based framework leads to two problems. First, setting the minimum support threshold is not easy. Second, it tends to generate a large number of patterns, most of them being of poor interest. To cope with these issues, asking the number of desired outputs is considered easier (Fu, Kwong, & Tang, 2000). In addition, with such an approach, additional constraints (such as closed property and minimum pattern length) can be incorporated (Han, Wang, Lu, & Tzvetkov, 2002). It is benefit from both usability and efficiency point of views.

In the applications mentioned above, the user may prefer to focus first on regularity of the patterns and on a reduced set of results. Amphawan, Lenca, and Surarerks (2009) thus introduced the problem of mining top-k frequent-regular patterns where the user does not need to specify a support threshold but may only focus on the number k of desired results and the regularity of patterns. However, this approach may generate redundant and/or short patterns. Thus, interesting patterns may be lost mainly because of redundant patterns, and especially the small ones. Indeed, these patterns tend to have more chance to appear in the top-k patterns. An efficient solution consists of mining closed patterns of minimum length (Han et al., 2002). Closed patterns will reduce redundancy. Minimum length constraint will increase the number of items in a pattern and thus may reveal interesting relations among them. In addition, it will tend to favor patterns with lower support which are often more interesting than very frequent –well known– patterns. Last, both parameters k and the minimal length of patterns are easier to set than the support threshold.

[*] Corresponding author.
E-mail addresses: komate@gmail.com (K. Amphawan), philippe.lenca@telecom-bretagne.eu (P. Lenca).

http://dx.doi.org/10.1016/j.eswa.2015.06.021
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

We thus propose to mine the *top-k frequent-regular closed patterns with minimal length*. An efficient single-pass algorithm, called TFRC-Mine, and a new compact bit-vector representation which allows to prune uninteresting candidate, are designed. Experiments show that the proposed algorithm is efficient to produce longer –non redundant– patterns, and that the new data representation is efficient for both computational time and memory usage.

The rest of the paper is organized as follows. Related works are presented in Section 2. Preliminary definitions and problem statement are given in Section 3. The new concise *bit-vector* representation is described in Section 4. TFRC-Mine algorithm is detailed in Section 5. Experiments and discussion are given in Section 6. Last, we conclude in Section 7.

2. Related works

Since Tanbeer et al. (2009) pointed out that regularity of occurrence can be of great interest, regular pattern mining has attracted several works particularly in transactional database and data streams. In addition, several ways of characterizing regularity (sometimes called periodicity by some authors) have been proposed. We here briefly review main related works.

Rashid, Karim, Jeong, and Choi (2012) proposed a pattern-growth based approach to mine frequent patterns that occur after regular intervals in a transactional database. In this work, the temporal regularity measure is based on the variance of interval time between pattern occurrences. Sreedevi and Reddy (2013) proposed a method to mine regular-closed itemsets in transactional database by a using vertical data format. It generates the complete set of regular-closed patterns for the user-given regularity and support thresholds.

Focusing on rare periodic-frequent patterns, Kiran and Reddy (2010) and Surana, Kiran, and Reddy (2012) proposed an efficient pattern-growth based algorithm and a methodology to dynamically specify the maximum periodicity threshold for each pattern. They thus use multiple support and regularity thresholds, one for each item. Kiran and Kitsuregawa (2015) proposed a new class of user-interest-based patterns named chronic-frequent patterns. A frequent pattern is said to be chronic if it has sufficient number of cyclic repetitions in a temporally ordered transactional database. To assess the interestingness of a frequent pattern the authors propose an anti-monotone periodic-recurrence measure, based on the number of cyclic repetitions in the database, which allows to make chronic-frequent pattern mining practicable with large databases. However, three thresholds, one for the support, one for the periodicity and one to decide if a pattern is chronic or not, are required.

Tanbeer, Ahmed, and Jeong (2010a) proposed a single-pass tree structure to capture streams contents in a compact manner and a pattern-growth based mining technique on the tree to mine the regular patterns in stream data. The temporal regularity measure is here based on a user-given regularity threshold as defined in Tanbeer et al. (2009). Kumar and Kumari (2012) proposed to mine regular frequent patterns in data streams with the sliding window technique using the vertical data format. The proposed algorithm satisfies downward closure property and is thus efficient. Also, because of the occurrence characteristic of patterns may change with the update of a database, Tanbeer, Ahmed, and Jeong (2010b) proposed a tree structure and pattern-growth based approaches to mine regular patterns in incremental transactional databases. They use a regularity threshold but do not consider the support. Kiran and Kitsuregawa (2014) proposed a greedy search on a pattern's tid-list to determine the periodic interestingness of a pattern. Pruning of the non-periodic-frequent patterns is done with a sub-optimal solution, while the periodic-frequent patterns are found with a global optimal solution. The algorithm requires both support and regularity thresholds.

As shown in Table 1 most of algorithms to mine regular and/or periodic patterns described above need to set a support threshold. To avoid this difficult task and to control the number of patterns mined Amphawan et al. (2009) introduced the problem of mining top-k frequent-regular patterns where the user needs only to specify the number k of desired patterns and the regularity threshold. A compressed tid-sets representation was used in Amphawan, Lenca, and Surarerks (2012a). An efficient pruning strategy based on support estimation and on a database partitioning technique is proposed in Amphawan, Lenca, and Surarerks (2012b). All the above algorithms mine a top-k list with a best-first strategy.

Table 1 summarizes the principal characteristics of related approaches described above. It shows that most of them produce redundant patterns and do not consider their minimal length. They thus limit the overall quality of the set of mined patterns.

Table 1
Details of previous approaches.

References	σ_s	σ_t	σ_μ	Closed property	Minimum length
Rashid et al. (2012)	Yes	Yes (variance)	variance	No	No
Sreedevi and Reddy (2013)	Yes	Yes	No	Yes	No
Kiran and Reddy (2010)	Yes (multiple)	Yes (multiple)	No	No	No
Surana et al. (2012)	Yes (multiple)	Yes (multiple)	No	No	No
Kiran and Kitsuregawa (2015)	Yes	Yes	periodic-recurrence	No	No
Tanbeer et al. (2010a)	No	Yes	No	No	No
Kumar and Kumari (2012)	No	Yes	No	No	No
Tanbeer et al. (2010b)	No	Yes	No	No	No
Kiran and Kitsuregawa (2014)	Yes	Yes	No	No	No
Amphawan et al. (2010)	Yes	Yes	No	No	No
Kiran and Kitsuregawa (2013)	Yes	Yes	periodic-ratio	No	No
Amphawan et al. (2009)	No (top-k)	Yes	No	No	No
Amphawan et al. (2012a)	No (top-k)	Yes	No	No	No
Amphawan et al. (2012b)	No (top-k)	Yes	No	No	No
Our proposition	No (top-k)	Yes	No	Yes	Yes

σ_s: require a support threshold.
σ_t: require a temporal (regularity or periodicity) threshold.
σ_μ: use an additional measure μ (with or without a threshold for μ).
Closed property: remove redundancy with the closed patterns.
Minimum length: take into account the minimum length of patterns for interestingness consideration.

Table 1 also shows that very few works consider other measures beyond the support and regularity. A perspective will be to consider interestingness measures for evaluating and selecting temporal patterns, but not necessary regular or periodic, as proposed in Chang (2011), Railean, Lenca, Moga, and Borda (2013) and Luo, Yuan, and Luo (2013). Last, although out of scope of this work we would like to point out that approximate regularities and periodicities has been considered by Amphawan, Surarerks, and Lenca (2010), Amir and Levy (2012) and Kiran and Kitsuregawa (2013).

By considering top-k closed patterns with minimal length constraint, we here propose to cope with all the issues mentioned above. In addition, a key point when taking into account temporal behavior is to maintain the occurrence data related to the items. For that purpose, we also propose an new compact representation of the data which allows to apply a pruning property to discover the regular patterns.

3. Preliminaries and problem definition

We here first present the basic concepts and notations used to discover top-k frequent-regular patterns. For the sake of consistency, we use similar definitions and notations as in Amphawan et al. (2012b). Last, we introduce the problem of top-k frequent-regular closed patterns with minimal length.

3.1. Frequent and regular patterns

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items. A set $X \subseteq I$, is called a pattern (an itemset) or a l-*pattern* if X contains l items. A transactional database $TDB = \{t_1, t_2, \ldots, t_m\}$ is a set of m transactions. Each transaction is a 2-tuple (j, Y) where j is a unique transaction identifier (tid for short), and $Y \subseteq I$ is an itemset.

The pattern X is said to occur in the transaction $t_j = (j, Y)$, and is denoted t^X_j, if $X \subseteq Y$. Obviously, it is equivalent to say that the transaction t_j contains X. The set $T^X = \{t^X_j, \ldots, t^X_k\}, j, k \in [1, m]$ and $j \leqslant k$, is the ordered set w.r.t. tids of transactions that contain X.

Definition 1 (*Regularity of pattern* X). Let t^X_j and t^X_k be two consecutive transactions in T^X, *i.e.* where $j < k$ and there is no transaction $t_i, j < i < k$, such that t_i contains X. Thus, $k - j$ is the gap of occurrence of X between t^X_j and t^X_k (notice that the number of transactions not containing X between t^X_j and t^X_k is $k - j - 1$). We define the regularity r^X of X to be the maximal gap of occurrence of X between any two consecutive transactions in T^X.

There are two special cases to consider: if X does not appears in the first transaction t_1 then its first regularity is initialized with the tid of the first transaction that contains X; if X does not appears in the last transaction t_m, then its last regularity is defined by $m - v$ where v is the tid of the last transaction that contains X. So, if one consider any subset of k transactions in T^X, where $k \geqslant r^X$, then the itemset X should appear at least one times.

3.2. Top-k frequent-regular closed patterns with minimum length

In the frequent and regular framework, the interestingness of a pattern X is evaluated by the support $s^X = |T^X|$ and the regularity r^X of X (Tanbeer et al., 2009). Thus, the problem of top-k frequent-regular patterns mining can be defined as below (Amphawan et al., 2009):

Definition 2 (*Top-k frequent-regular patterns* (*Amphawan et al., 2009*)). Let's regard the list of patterns sorted by descending support values. Let s_k be the support of the kth pattern in the sorted list. The top-k frequent-regular patterns mining problem is then to discover the set of first k patterns having a support greater or equal to s_k, a regularity no greater than a user-given maximum regularity threshold σ_r, where k is the number of user desired patterns.

However, without the closed property of patterns, the top-k frequent-regular patterns mining approach may generate redundant and/or uninteresting patterns. There may also be a large portion of short patterns included in the results in which users cannot well extract interesting relationship among items. To cope with these issues, we here propose to consider the closed property on frequent-regular patterns and to constrain the minimum number of items in each pattern as proposed by Han et al. (2002) for top-k frequent closed patterns.

Definition 3 (*Closed property on a pattern* X (*Pasquier, Bastide, Taouil, & Lakhal, 1999*)). A pattern X is closed if there is no proper superset with the same support.

Definition 4 (*Top-k frequent-regular closed patterns of minimal length*). The top-k frequent-regular closed patterns of minimal length mining problem is to discover the set of k closed patterns having highest support, a regularity no greater than a user-given maximum regularity threshold σ_r and length no smaller than a user-given minimum length threshold min_l, where k is the number of user desired patterns.

4. PDBV: Partitioned Dynamic Bit-Vector

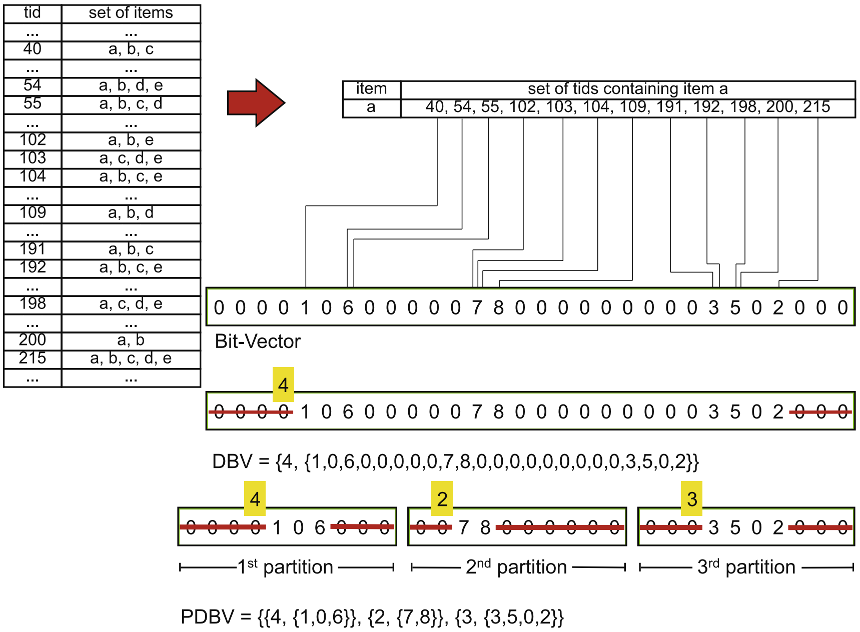
We here introduce the Partitioned Dynamic Bit-Vector, a new bit-vector representation which allows to reduce memory consumption and fast tids intersection, support and regularity computations to generate and evaluate candidate patterns (see the TFRC-Mine algorithm described 5).

4.1. Towards Partitioned and Dynamic Bit-Vector

Bit-vector is commonly applied to mine frequent and/or closed patterns for vertical representation of the database (Burdick, Calimlim, & Gehrke, 2001; Shenoy et al., 2000). A *bit-vector* is a vector of m bits (m being the number of transactions) and there is one *bit-vector* for each considered pattern. The ith bit is 1 (respectively 0), if the transaction of tid i contains the corresponding pattern (respectively does not contain). This representation allows to scan the database once and also to reduce memory consumption. Several efficient algorithms have been proposed and results are significant (*e.g.* Dong & Han, 2007; Song, Yang, & Xu, 2008). However, *bit-vector* always have a fixed size equal to m. In addition, many *bit-vector*s may contain many consecutive 0 bits. This can be particularly the case with sparse databases that are frequently considered. So, there is a room for improvement of *bit-vector* representation.

Vo, Hong, and Le (2012) thus proposed a concise *bit-vector* namely *Dynamic Bit-Vector* (DBV). The main idea of DBV is based on the use of a positive integer to recognize the number of consecutive 0 from the head of the vector and then applies normal *bit-vector* to store all of information after the first appearance of the considered pattern. A similar compression is done after the last occurrence of the pattern.

For example, let's consider the pattern a occurring in the set of transactions $T^a = \{t^a_{40}, t^a_{54}, t^a_{55}, t^a_{102}, t^a_{103}, t^a_{104}, t^a_{109}, t^a_{191}, t^a_{192}, t^a_{198}, t^a_{200}, t^a_{215}\}$, where TDB contains 240 transactions ($m = 240$). Let us suppose that a *bit-vector* is represented by bytes of 8 bits. So, one needs here 30 bytes. Fig. 1 shows the *bit-vector* of pattern a in decimal notation. The first four bytes are 0



tid	set of items
...	...
40	a, b, c
...	...
54	a, b, d, e
55	a, b, c, d
...	...
102	a, b, e
103	a, c, d, e
104	a, b, c, e
...	...
109	a, b, d
...	...
191	a, b, c
192	a, b, c, e
...	...
198	a, c, d, e
...	...
200	a, b
215	a, b, c, d, e
...	...

item	set of tids containing item a
a	40, 54, 55, 102, 103, 104, 109, 191, 192, 198, 200, 215

Fig. 1. Example of *bit-vector*, DBV, and PDBV for an item 'a' (expressed in Bytes).

(corresponding to the 32 first transactions) and the fifth one is 1 (*i.e.* 00000001) corresponding to transactions $\mathtt{t}_{33}$ to $\mathtt{t}_{39}$, where a does not occur, and to $\mathtt{t}_{40}$ which contains a. The last three bytes are also equal to 0 because a does not appear in transactions $\mathtt{t}_{217}$ to $\mathtt{t}_{240}$, meanwhile the absence on $\mathtt{t}_{216}$ is contained in the last non-zero byte of DBV. DBV will then compress the *bit-vector* (7 bytes can be removed, and 1 byte is used to store this information).

DBV applies only at the head and at the tail of the vectors. In many situations, a large part of the patterns may appear in the very first transactions and also in the last ones. For these cases, the DBV approach will not be efficient. In addition, compression can be done in a similar way between the head and the tail of the vectors. We thus propose a *Partitioned Dynamic Bit-Vector* (PDBV) representation to tackle these issues.

4.2. Structure of Partitioned Dynamic Bit-Vector

Remember that a pattern is regular, if it occurs at least once in every $\sigma_{\mathtt{r}}$ consecutive transactions. So, if the database is split into $\mathtt{p} = \lceil \mathtt{m}/\sigma_{\mathtt{r}} \rceil$ consecutive partitions, a regular pattern should appear at least once in each partition. This remark is the main idea behind the definition of the *Partitioned Dynamic Bit-Vector* (PDBV). A PDBV is simply a set of DBVs of equal size $\{\mathtt{DBV}_1, \mathtt{DBV}_2, \ldots, \mathtt{DBV}_{\mathtt{p}}\}$ where each $\mathtt{DBV}_{\mathtt{i}}$ is build (following the traditional approach) on partition i.

Let's consider the example depicted in Fig. 1 where a *bit-vector* of 30 bytes represents the occurrences of item a. If we split the database into three partition ($\mathtt{p} = 3$), then each partition contains 10 bytes. The corresponding PDBV removes 21 bytes of 0 and add for that 3 bytes. PDBV needs less memory than DBV (12 against 24 bytes).

Based on DBV, PDBV will have similar performance in worst case. However in many cases it will need less memory, especially on sparse databases.

Lemma 1. *For any pattern X, the number of bytes contained in* $PDBV^X$ *is less or equal than* DBV *(and obviously than bit-vector).*

Proof. $\mathtt{PDBV} = \{\mathtt{DBV}_1, \mathtt{DBV}_2, \ldots, \mathtt{DBV}_{\mathtt{p}}\}$ compresses the vector at any head if there at least two bytes of 0 and at the tail if there at least one byte of 0 for each $\mathtt{DBV}_{\mathtt{i}}$ corresponding to the ith partition. DBV compresses the vector only at the head (*i.e.* corresponding to $\mathtt{DBV}_1$) and at the tail of the non partitioned vector (*i.e.* corresponding to $\mathtt{DBV}_{\mathtt{p}}$). So, PDBV is equal to DBV when a pattern occurs in the first and in the last transactions of each partition. In all other cases, PDBV has a higher compression rate. □

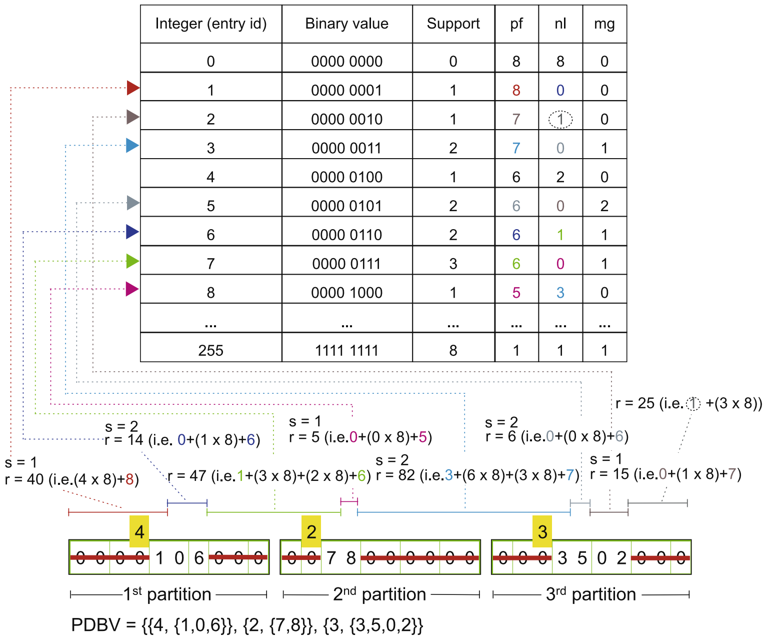
4.3. An efficient method for fast computing support and regularity from a PDBV

Computing the support and the regularity of each –candidate– pattern are costly. Vo et al. (2012) proposed to use a look-up table, an additional-static table (completely independent from the contents of database) which contains the support of all possible values of each byte, to efficiently count the number of bits 1 of a *bit-vector* (*i.e.* its support). We here also use a similar look-up table where we add three information to compute the regularity values from each byte of PDBV:

- pf: the position of first appearance of bit 1 (if there are only bits 0, $\mathtt{pf} = 8$).
- nl: the number of bits 0 after the last occurrence of a bit 1 to the end of the byte (if there are only bits 0, $\mathtt{nl} = 8$).
- mg: the maximum position gap between two bits 1 (if there is only one or zero bit 1, $\mathtt{mg} = 0$).

With a look-up table, the time complexity for computing support and regularity values of any pattern is then $\mathcal{O}(\mathtt{b})$, where b is the maximal number of bytes contained in PDBV. The worst case is when no compression can be done *i.e.* as for DBV and *bit-vector*.

As an example, let's consider the look-up table of Fig. 2 where each byte is a sequence of 8 bits. This table has thus $2^8 = 256$ entries. The third entry corresponds to the sequence 00000010 (which values 2) where there is only one bit 1 (at the 7*th* position): $\mathtt{pf} = 7, \mathtt{nl} = 1, \mathtt{mg} = 0$. The support of this byte is 1.



Integer (entry id)	Binary value	Support	pf	nl	mg
0	0000 0000	0	8	8	0
1	0000 0001	1	8	0	0
2	0000 0010	1	7	1	0
3	0000 0011	2	7	0	1
4	0000 0100	1	6	2	0
5	0000 0101	2	6	0	2
6	0000 0110	2	6	1	1
7	0000 0111	3	6	0	1
8	0000 1000	1	5	3	0
...	...	...	...	...	...
255	1111 1111	8	1	1	1

Fig. 2. Example of regularity calculation.

The regularity of each pattern X can then be computed by sequentially considering each $\mathtt{DBV}_i$ of the corresponding PDBV. Let $\mathtt{hO}_i$ (respectively $\mathtt{tO}_i$) be the number of consecutive bytes whose value 0 from the head (respectively to the tail) of each $\mathtt{DBV}_i$. Indeed, one have to take into account only 3 situations:

1. the first appearance of X in the current partition: this is determined by the gap of appearance between the last non-zero byte of $\mathtt{DBV}_{i-1}$ and the first non-zero byte of $\mathtt{DBV}_i$,
2. the appearance of X between two bytes: this is determined by the gap of disappearance between any two consecutive non-zero bytes, and
3. the last appearance of X in the database: this is determined by the gap between the last bit '1' of the last non-zero byte in $\mathtt{DBV}_p$ and the end of database.

Let's analyze the calculation of regularities in $\mathtt{DBV}_3 = \{3, \{3, 5, 0, 2\}\}$ (Fig. 1). It has three non-zero bytes. Four local regularities need to be computed: one for the first non-zero byte value 3, one for the gap between the first and the second non-zero bytes values 3 and 5, one for the gap between the second and the third non-zero bytes values 5 and 2, and one for the gap between the last non-zero byte value 2 and the end of the partition. These four situations correspond respectively to case 1, case 2, case 2 and case 3 of Lemma 3. Each calculation is illustrated in Fig. 2.

Lemma 2. *The look up table and PDBV^X provide an efficient way of computing the support of a pattern X.*

Proof. Following the definition provided in Vo et al. (2012), the jth element of the lookup table contains the number of bits 1 of the byte value j. Computation of the support of X with $\mathtt{PDBV}^{\mathtt{X}}$ is done by summation of the support of the jth element of the lookup table for the non-zero bytes value j in each $\mathtt{DBV}_i^{\mathtt{X}}$. These operations on bytes are in $\mathcal{O}(\lceil \mathtt{m}/\mathtt{bytesize} \rceil)$. The computation of the support with the $\mathtt{PDBV}^{\mathtt{X}}$ representation is thus correct and efficient. □

Lemma 3. *The look up table and PDBV^X provide an efficient way of computing the regularity of a pattern X.*

Proof. Based on the three cases mentioned above the regularity of a pattern is:

$$max\begin{cases} max(\mathtt{nl}_{\mathtt{lo}_{i-1}} + (\mathtt{tO}_{i-1} \times 8) + (\mathtt{hO}_i \times 8) + \mathtt{pf}_{\mathtt{fo}_i}, \mathtt{mg}_{\mathtt{fo}_i}) & \mathrm{case:1} \\ max(\mathtt{nl}_{\mathtt{lo}_i} + (\mathtt{nz} \times 8) + \mathtt{pf}_{\mathtt{c}_i}, \mathtt{mg}_{\mathtt{c}_i}) & \mathrm{case:2} \\ \mathtt{nl}_{\mathtt{lo}_p} + (\mathtt{tO}_p \times 8) & \mathrm{case:3} \end{cases}$$

where: $\mathtt{nl}_{\mathtt{lo}_i}$ is the number of bits 0 after the last bit 1 of the last non-zero byte of $\mathtt{DBV}_i$; $\mathtt{tO}_{i-1}$ is the number of byte 0 at the tail of previous $\mathtt{DBV}_{i-1}$; $\mathtt{hO}_i$ is the number of bytes 0 at the head of the current $\mathtt{DBV}_i$; $\mathtt{pf}_{\mathtt{fo}_i}$ is the position of first bit 1 in the first non-zero byte of $\mathtt{DBV}_i$; $\mathtt{mg}_{\mathtt{fo}_i}$ is the maximum position gap between two bits 1 in the first non-zero byte of $\mathtt{DBV}_i$; $\mathtt{nz}$ is the number of bytes 0 between two non-zero bytes; $\mathtt{pf}_{\mathtt{c}_i}$ is the position of first bit 1 in considered non-zero byte of $\mathtt{DBV}_i$; $\mathtt{mg}_{\mathtt{c}_i}$ is the maximum position gap between two bits 1 in considered non-zero byte of $\mathtt{DBV}_i$; $\mathtt{nl}_{\mathtt{lo}_p}$ is the number of bits 0 after the last bit 1 of the last non-zero byte of $\mathtt{DBV}_p$; $\mathtt{tO}_p$ is the number of bytes 0 at the tail of $\mathtt{DBV}_p$.

The values $\mathtt{nl}_{\mathtt{lo}_i}, \mathtt{pf}_{\mathtt{fo}_i}, \mathtt{mg}_{\mathtt{fo}_i}, \mathtt{pf}_{\mathtt{c}_i}, \mathtt{mg}_{\mathtt{c}_i}$ and $\mathtt{nl}_{\mathtt{lo}_p}$ are easily extracted from the look-up table. The value $\mathtt{tO}_{i-1}, \mathtt{hO}_i, \mathtt{nz}$, and $\mathtt{tO}_p$ are also easily retrieved by reading the bytes of $\mathtt{DBV}_{i-1}$ and $\mathtt{DBV}_i$. The operations on bytes are in $\mathcal{O}(\lceil \mathtt{m}/\mathtt{bytesize} \rceil)$. The computation of the regularity with the $\mathtt{PDBV}^{\mathtt{X}}$ representation is thus efficient. □

4.4. Method for PDBVs intersection

The classical intersection operations on PDBVs are done to generate new candidate patterns in the same manner as for DBVs (Vo et al., 2012). The main difference is here that we can consider to sequentially intersect each $\mathtt{DBV}_i$ contained in the set of considered PDBVs (as shown in Algorithm 1).

Algorithm 1. PDBVs' intersection

```
Input: set of n PDBVs: $\{PDBV_1, PDBV_2, \ldots, PDBV_n\}$
Output: a new $PDBV = \{DBV_1, DBV_2, \ldots, DBV_n\}$, its
  support s, its regularity r
  for each jth partition from all p partitions do
    • $hO_j^{PDBV} = \max(hO_j^{PDBV_1}, hO_j^{PDBV_2}, \ldots, hO_j^{PDBV_n})$
  //find the maximum position
    for each $i = 1$ to n do
      • $pos_i = hO_j^{PDBV} - hO_j^{PDBV_i}$ //find the first byte
  of $DBV_j^{PDBV_i}$ to intersect
    • $count = \max(|DBV_j^{PDBV_1}| - pos_1, |DBV_j^{PDBV_2}| - pos_2, \ldots,$
  $|DBV_j^{PDBV_n}| - pos_n)$ //compute number of bytes to intersect
    while $count > 0$ and $(b = b_{pos_1}^{DBV_j^{PDBV_1}} \cap b_{pos_2}^{DBV_j^{PDBV_2}}$ \hskip
  0.35em
  $\cap \ldots \cap b_{pos_n}^{DBV_j^{PDBV_n}}) = 0$ do
      • decrease count by 1 and increase $pos_1,$
  $pos_2, \ldots, pos_n$ by 1
      • increase $hO_j^{PDBV}$ by 1
    • calculate s and r from b (case: 1 of Lemma 3)
    • collect b as a member of $DBV_j$ of PDBV
    while $count > 0$
      • $b = b_{pos_1}^{DBV_j^{PDBV_1}} \cap b_{pos_2}^{DBV_j^{PDBV_2}} \cap \ldots \cap b_{pos_n}^{DBV_j^{PDBV_n}}$
      • decrease count by 1 and increase $pos_1,$
  $pos_2, \ldots, pos_n$ by 1
      • collect b as a member of $DBV_j$ of PDBV
      if $b > 0$ then
        • calculate s and r from b (case: 2 of Lemma 3)
    • delete sequence of bytes 0 at the end of $DBV_j$ of PDBV
  • calculate regularity r from the last occurrence of the
  last partition and the end of database (case: 3 of Lemma 3)
```

5. TFRC-Mine: top-k frequent-regular closed patterns mining based on minimum length

The TFRC-Mine algorithm mines top-k frequent-regular closed patterns with length no less than min_l as described in Definition 4. It uses two lists, one for frequent and regular items and one top-k list for the output patterns. The proposed PDBV representation is used to collect the information of patterns. TFRC-Mine has two main steps. First, it scan once the database to initialize the first list. Second, frequent and regular items (*i.e.* member of the first list) are merged to generate patterns of minimal length.

5.1. List of frequent and regular items

The initialization of the frequent and regular items list FRI-list is done with one database scan. Each entry of FRI-list contains 5 information: an item i, its support s^i, its regularity r^i, its last known occurrence tid lo^i and its *Partitioned Dynamic Bit Vector* $PDBV^i$.

The database is then sequentially scanned by group of σ_r consecutive transactions in order to build the p DVBs of PDBV of all items. The first σ_r transactions will help to initialize the FRI-list: a new entry is created for any item i that firstly occurs ($s^i = 1$, lo^i = tid of the transaction, and the other informations are initialized as described in Section 4); otherwise, the entry for item i is updated (s^i is increased by 1, lo^i is set to the new tid, etc.). The following groups of σ_r transactions are then scanned and used to only update the entries of items already occurring in the first group of transactions. Indeed, if an item does not appear in the σ_r first transactions, then this item will not be regular (see the more general regularity property described in Lemma 4). Finally, items with regularity greater than σ_r are deleted from the FRI-list and the list is sorted in support descending order. Details of FRI-list initialization are shown in Algorithm 2.

Lemma 4. *A pattern X is not a regular pattern if there is one DBV_i^X, except the last DBV_p^X, in $PDBV^X$ where each byte is 0.*

Proof. Each DBV_i^X in $PDBV^X$ represents a set of X's occurrence information in σ_r consecutive transactions. So, if each byte of any DBV_i^X, except the last one (DBV_p^X), is 0 then X does not occur for σ_r consecutive transactions and thus can not be regular. If each byte of DBV_p^X is 0, then X is not regular if and only if X does not occur in the last transaction of the previous DBV^X. Notice that if $m \div \sigma_r$ is not an integer, DBV_p^X only contains occurrence information of X for the last $m\% \ \sigma_r$ transactions. For this case, it cannot guarantee, whether X is regular or non-regular, because X can be regular even if DBV_p^X contains only byte 0. However, the last regularity of X can be calculated by case 3 of Lemma 3. When a pattern X is detected as non-regular, then it can be definitively pruned. □

Algorithm 2. Scanning of database

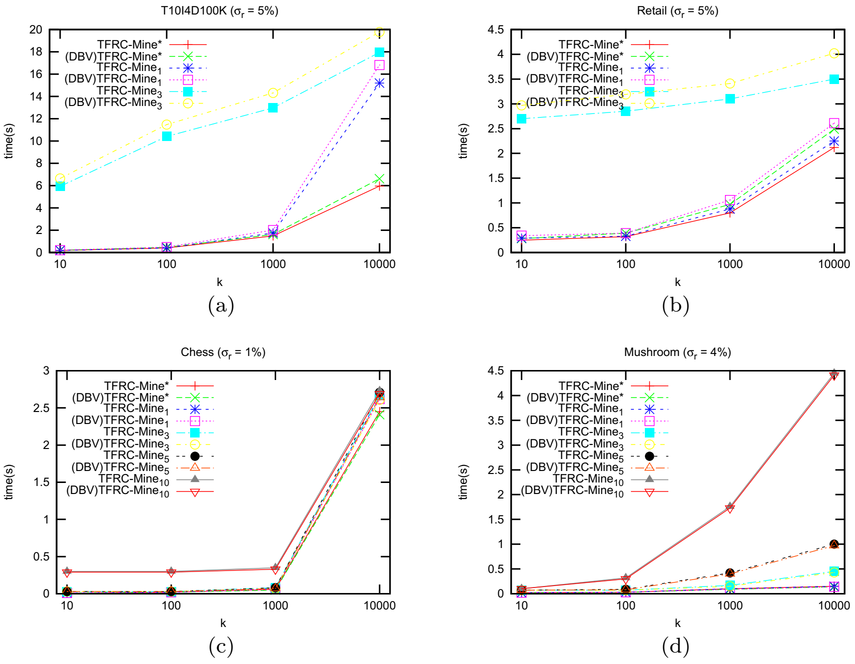
```
Input: $k, \sigma_r$, TDB
Output FRI-list
  • split TDB into small partitions and initialize FRI-list
  for each transaction $t_j$ in the first partition do
    for each item i in transaction $t_j$ do
      if item i does not have an entry in the FRI-list then
        • create a new entry for item i with
  $s^i = 1, r^i = j, lo^i = j$
        • create and update $DBV_1^i$ with j
      else
        • add the support $s^i$ by 1
        • calculate regularity $r^i$ by j ($r^i = j - lo^i$ if
  $(j - lo^i) > r^i$)
        • update $lo^i$ and $DBV_1^i$ with j
  for each partition $m = 2$ to the last partition do
    for each transaction $t_j$ in mth partition do
      for each item i in transaction $t_j$ do
        if item i has an entry in the FRI-list then
          if $t_j$ is the first occurrence of i in mth partition
  then
            • add the support $s^i$ by 1
            • calculate regularity $r^i$ by j ($r^i = j - lo^i$ if
  $(j - lo^i) > r^i$)
            • create $DBV_m^i$ and update $lo^i$ and $DBV_m^i$ with j
          else
            • add the support $s^i$ by 1
            • calculate regularity $r^i$ by j ($r^i = j - lo^i$ if
  $(j - lo^i) > r^i$)
            • update $lo^i$ and $DBV_m^i$ with j
  for each item i in the top-k list do
    • calculate regularity $r^i$ by $|TDB| - lo^i$
    if $r^i > \sigma_r$ then
      • remove entry of i out of the FRI-list
  • sort FRI-list by support descending order
```

5.2. Mining a complete set of results

The mining process of the top-k frequent-regular closed patterns is mainly performed with a top-k list. Each entry of the top-k list contains 5 information: a pattern X of $\mathtt{min}_1$ length at least, its support $\mathtt{s}^{\mathtt{X}}$, its regularity $\mathtt{r}^{\mathtt{X}}$, its set $\mathtt{at}^{\mathtt{X}}$ of items that always appear with pattern X, and its *Partitioned Dynamic Bit Vector* $\mathtt{PDBV}^{\mathtt{X}}$.

The algorithm TFRC-Mine then apply a best-first search strategy w.r.t. the support order in the FRI-list and the constraint on the number of items to generate candidate patterns. Indeed, items with high support have more chance to frequently and/or regularly appear together.

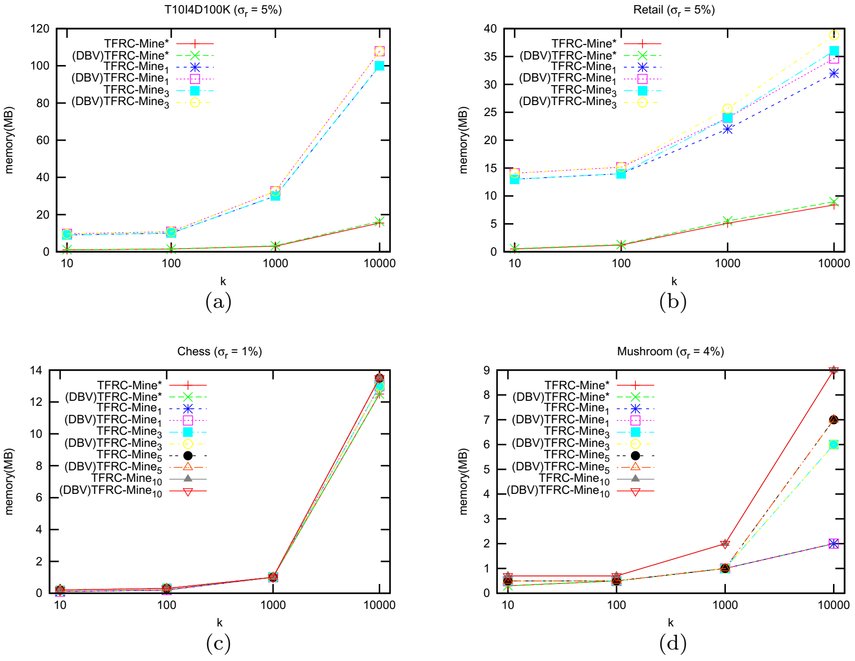
As described in Algorithm 3, TFRC-Mine thus consider the first group of $\mathtt{min}_1$ items of size 1 and join them. Their PDBVs are intersected in order to compute the support, the regularity and the PDBV of the candidate pattern. The candidate pattern's entry is inserted at the first position of the top-k list, if the regularity is no greater than $\sigma_{\mathtt{r}}$. Then, the item at position $\mathtt{min}_1 + 1$ is considered to be joined with previous items using the same constraints (*i.e.* the pattern is formed by $\mathtt{min}_1$ items, it has a regularity no greater than $\sigma_{\mathtt{r}}$, and it is inserted in the top-k at its right position w.r.t. descending support order). In case of insertion of a pattern, its neighborhood needs to be examined (Yan, Han, & Afshar, 2003): if there is no pattern in its neighborhood with the same prefix and the same support, then the inserted pattern is closed and the kth entry is deleted; otherwise, there is a chance that it is not closed, and the top-k list has to store more than k patterns for a while in order to be sure that any closed patterns with highest support will not be missed. This process continue with the remaining items in the FRI-list. At the end of this process, the top-k list is initialized with at most k frequent-regular patterns of length $\mathtt{min}_1$.



(a) (b)

(c) (d)

Fig. 3. Runtime of PDBV against DBV.



(a) (b)

(c) (d)

Fig. 4. Memory usage of PDBV against DBV.

The initialization of the top-k list is done with a same process. However, an additional constraint is applied: only couple X and Y of patterns with the same prefix are considered to generate longer candidate pattern $Z = XY$. This constraint reduce the number of unsuccessful candidate and the number of merging operations to generate a successful candidate (Yan et al., 2003).

Then, TFRC-Mine has to check whether X and/or Y are closed or not. One has here to consider three cases:

1. if $s^Z = s^X = s^Y$ then X and Y can not be closed; the last item of pattern Y is added to at^X (*i.e.* the entry of X contains the new candidate pattern $Z = X \cup at^X$), and the entry of Y is deleted (at this time, the top-k list does not contain patterns X and Y anymore);
2. if $s^Z = s^X$ and $s^Z \neq s^Y$ then X can not be closed; the process is then similar to case 1 except that the entry of Y is not deleted;
3. if $s^Z \neq s^X$ and $s^Z \neq s^Y$ then X and Y can be closed; the candidate pattern Z is inserted into the top-k list by taking into account its neighborhood as described above.

This merging step is repeated to all pairs of patterns in the top-k list. The process will end, thanks to regularity and prefix constraints. Finally, all the patterns after the kth pattern are eliminated from the top-k list, since they cannot be a result.

Algorithm 3. Mining a set of k patterns

```
Input: FRI-list, $\sigma_r$, k, $min_1$
Output: A set of top-k frequent-regular closed patterns
  for a group of $min_1$ items (e.g. $\{i_1, i_2, \ldots, i_{min_1}\}$) in the FRI-list
    • merge all items in the group to generate pattern Z
    • intersect PDBVs: $PDBV_1 \cap PDBV_2, \ldots, \cap PDBV_{min_1}$ to compute support $s^Z$, regularity $r^Z$ and collect $PDBV^Z$ (as described in Section 4.4)
    if $r^Z \leqslant \sigma_r$ and $s^Z \geqslant s_{kth}$ then
    • create an entry for pattern Z with $s^Z, r^Z$ and $PDBV^Z$ and insert into the top-k list by support descending order
    if there is no neighbor of Z with the same prefix items, support, and regularity then
      • remove the entry of the kth entry from the top-k list
  for each pattern X in the top-k list do
    for each pattern Y in the top-k list $(X \neq Y)$ do
      if X and Y have the same prefix then
        • merge patterns X and Y to be pattern Z
        • intersect $PDBV^X$ and $PDBV^Y$ to compute support of $s^Z$, regularity $r^Z$, and collect $PDBV^Z$
        if $r^Z \leqslant \sigma_r$ and $s^Z \geqslant s^k$ then
          if $s^Z = s^X = s^Y$ then
            • add the last item of pattern Y to $at^X$ and remove the entry of Y from the top-k list
          else if $s^Z = s^X$ ($s^Z \neq s^Y$) then
            • add the last item of Y to $at^X$
          else if $s^Z = s^Y$ ($s^Z \neq s^X$) then
            • add the last item of X to $at^Y$
          else
            • create an entry for pattern Z with $s^Z, r^Z$ and $PDBV^Z$ and insert into the top-k list by support descending order
            if there is no neighbor of Z with the same prefix items, support, and regularity then
              • remove the entry of the kth entry from the top-k list
  • remove all the patterns after the kth pattern out of the top-k list
```

Lemma 5. TFRC-Mine *only produces the top-k frequent-regular closed patterns of length min_l.*

Proof. The top-k list is initialized only with patterns of min_1 items. Each candidate pattern is build by merging two patterns in the top-k list, so each final pattern contains at least min_1 items. The use of Lemma 4 guarantees that only regular patterns are considered. Each candidate or new pattern is inserted at its right place in the top-k list w.r.t. support descending order. So obviously, the top-k list contains only frequent-regular patterns of length min_1.

Let's now show that the final patterns are closed. For each candidate pattern X, TFRC-Mine first check whether it is closed. If there is no pattern with the same prefix as X, then X is closed (since there is no possibility to generate a superset of X that have the same support of X). The closed pattern X will then belongs to the final result if there is no more than $k - 1$ patterns with greater support. Otherwise, the pattern X is merged with all the patterns with same prefix to generate new longer candidate patterns (case 3). If none of these patterns have the same support as X, then X is now closed. Otherwise, X is non-closed and will be eliminated by case 1 or case 2. This process is repeated until at most k (it can be less depending of the regularity values) closed patterns are found. Each iteration either remove non closed patterns and/or add a closed pattern. The number of iteration is bounded as k is a finite number. □

6. Experimental study

This section reports the performance study of TFRC-Mine. To the best of our knowledge, this is no competitive algorithm which also aims to avoid redundancy for the task of mining the top-k frequent-regular patterns. The closest algorithms are MTKPP, TR-CT, TKRIMPE which aim to mine top-k frequent-regular patterns (Amphawan et al., 2009, 2012a, 2012b). We thus also analyze TFRC-Mine* which corresponds to TFRC-Mine with the minimum length min_1 of patterns to be mined set to 1 and without closed property constraint. In that way, TFRC-Mine* is similar to MTKPP, TR-CT, and TKRIMPE except that each algorithm has its own representation to maintain occurrence information.

We also compare the efficiency (runtime and memory usage), average length of patterns and compactness of results of TFRC-Mine, TFRC-Mine*, MTKPP, TR-CT, and TKRIMPE, respectively.

6.1. Datasets and test environment

Comparison has been done with four databases available at (Goethals & Saki, 2003). PDVB and thus TFRC-Mine are expected to be efficient when databases are sparse. Dense databases are also considered to show that TFRC-Mine is still efficient in that case:

- *Dense databases*: *Chess* database which consists of 3,196 transactions with an average length of 37 items; *Mushroom* database which consists of 8,124 transactions with an average length of 23 items;
- *Sparse databases*: *T10I4D100K* which consists of 100,000 transactions with an average length of 10 items; *Retail* which consists of 88,162 transactions with an average length of 10.3.

All experiment were performed on a Intel® Xeon 2.4 GHz with 8 GB of memory, running Linux system. Both algorithms are implemented in C.

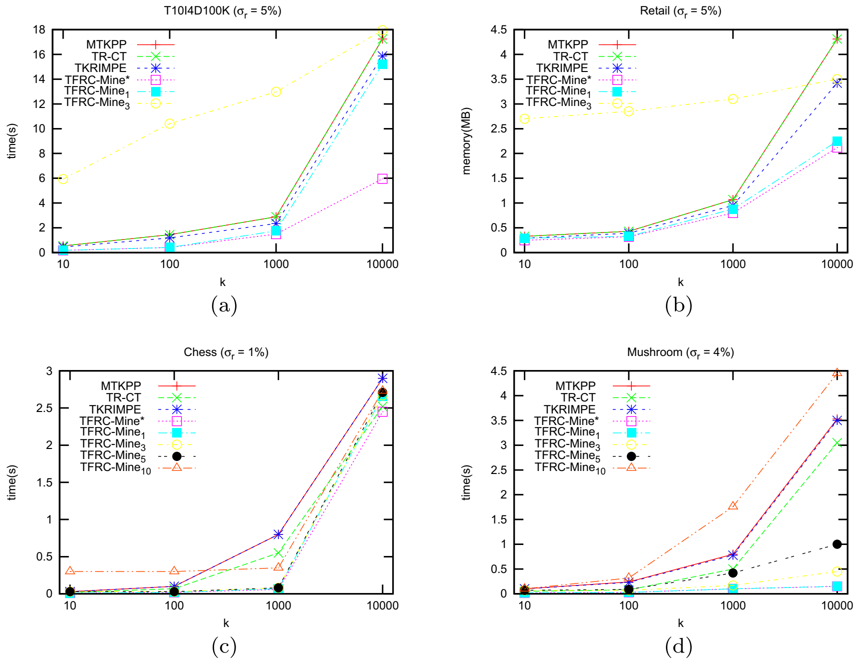
The parameter values are set so that the behavior of the algorithms can be analyzed in several meaningful situations.

Comparison is done for $\mathtt{k} = 10, 100, 1000$ and $10\,000$, for several values of σ_r between 1% and 16% depending on the density of the database and several values of $\mathtt{min}_l$ from 1 to 10 depending on the average length of transactions and σ_r.

6.2. Performance on PDBV

The efficiency of PDBV against DBV can be observed in terms of runtime and memory usage. Fig. 3 shows the runtime of both representations for several values of k and $\mathtt{min}_l$. It shows that PDBV is faster for sparse datasets and has similar performance as DBV for dense datasets. These experimental results are consistent with what is expected theoretically: with sparse datasets, there are many sequences of byte 0 which can be removed by PDBV and thus runtime for storing and intersection occurrence information processing can be reduced (in average PDBV is 10% faster than DBV); however, with dense datasets, this benefit is reduced and PDBV tends to have similar runtime than that of DBV.

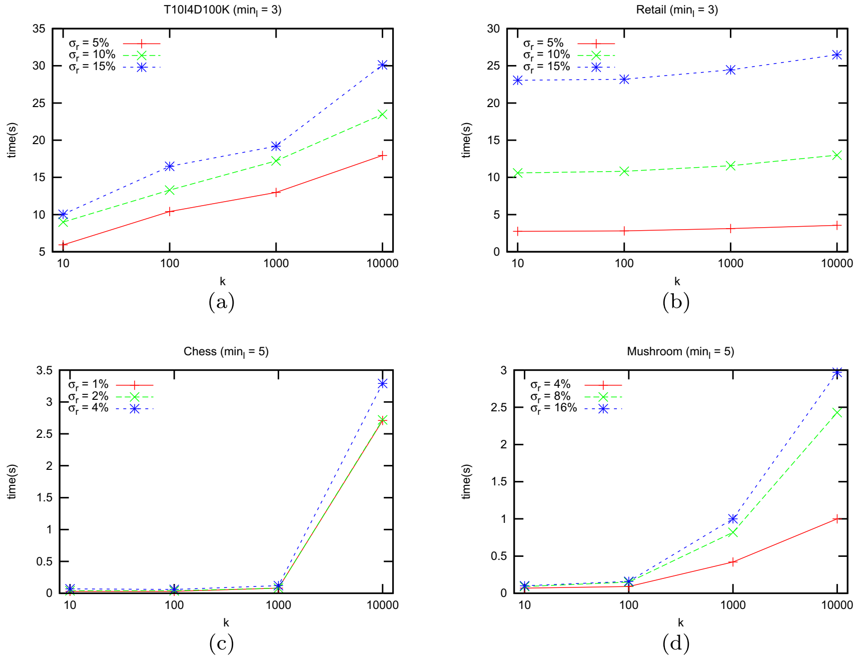
Fig. 4 shows the memory usage of both representations. The result is also consistent with what is expected theoretically. The figure shows that PDBV uses less memory on sparse datasets (about 8 to 10% less) but uses nearly the same amount of memory than DBV on dense datasets. The reduction of memory usage made by PDBV also contributes to decrease the runtime.



(a) (b)

(c) (d)

Fig. 5. Runtime with the variation of minimum length.



(a) (b)

(c) (d)

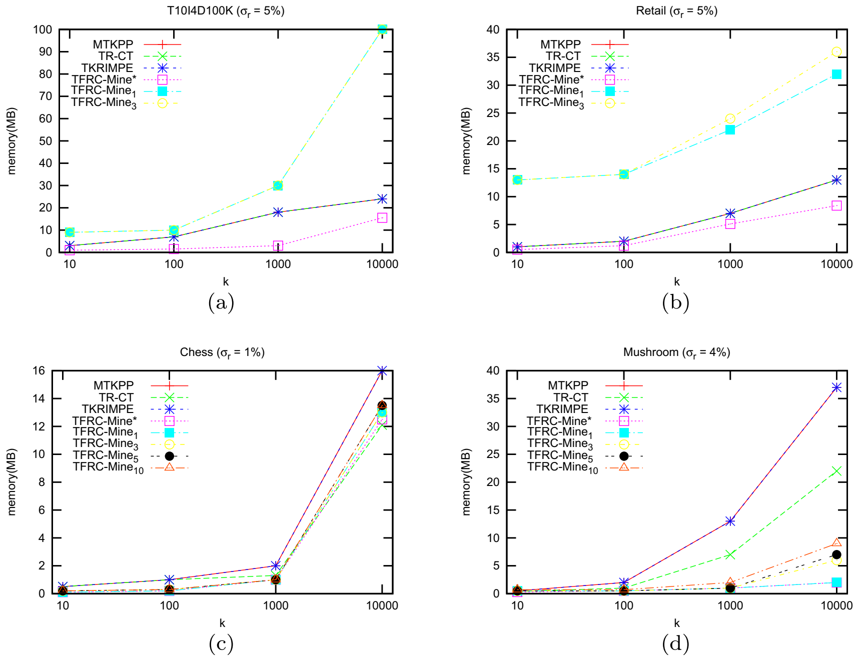
Fig. 6. Runtime with the variation of regularity threshold.

6.3. Computational time

The comparison between TFRC-Mine*, MTKPP, TR-CT and TKRIMPE aims to illustrate the advantage for runtime and memory of the new bit vector representation PDBV used to collect tids (occurrence information). Runtime w.r.t. minimum length of these algorithms is presented Fig. 5 (where $\mathtt{TFRC} - \mathtt{Mine}_1$ indicates that TFRC-Mine is run with $\mathtt{min}_1 = 1$).

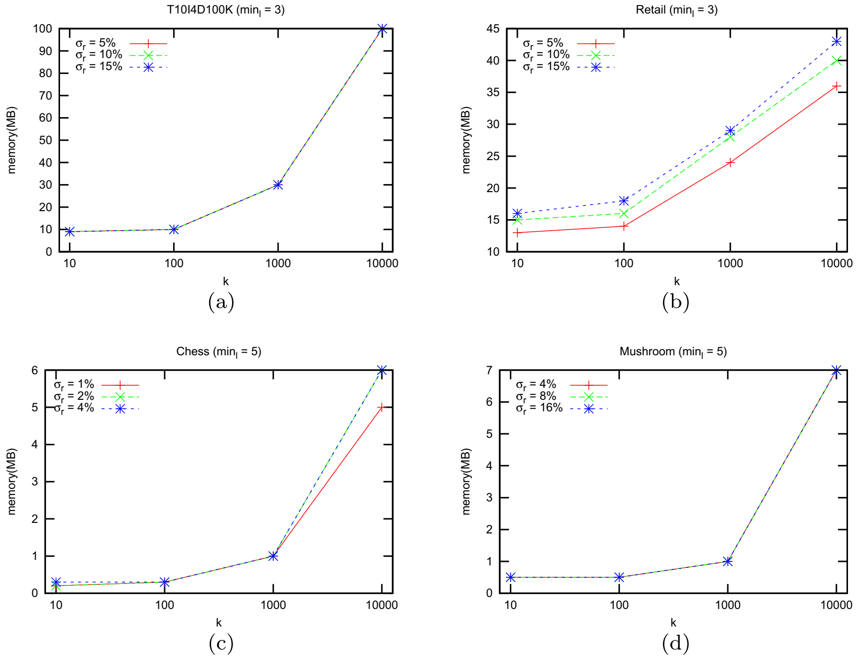
For each situations, TFRC-Mine* is faster than MTKPP (as for TR-CT and TKRIMPE). TFRC-Mine is still faster when $\mathtt{min}_1 = 1$. Obviously, mining larger patterns requires more computations, especially more combinations between candidates and more intersections of tids are needed. These operations are numerous when databases are sparse. Thus, TFRC-Mine is slower than MTKPP in most cases. However, TFRC-Mine is still very competitive for larger $\mathtt{min}_1$ especially when databases are dense. In addition, it is competitive for large k whatever is the database type.

The runtime of TFRC-Mine, when minimum length is fixed, mostly depends on the regularity threshold σ_r and on the type of the database (sparse/dense, number of transactions, and number of items). The higher σ_r is the larger the set of candidate regular patterns is. TFRC-Mine thus needs more times. However, it is still efficient for large σ_r as shown in Fig. 6, even for the biggest



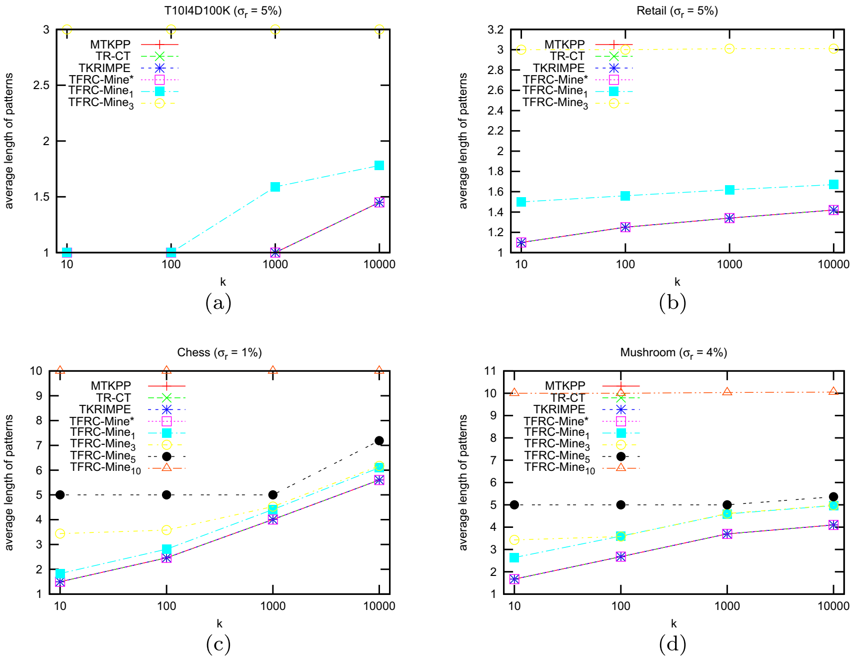
(a) (b) (c) (d)

Fig. 7. Memory usage with the variation of minimum length.



(a) (b) (c) (d)

Fig. 8. Memory usage with the variation of regularity threshold.



(a)

(b)

(c)

(d)

Fig. 9. Average length of results with the variation of minimum length.

database (T10I4D100K) and for the database with a large amount of items (Retail).

6.4. Memory consumption

Memory usage w.r.t. minimum length is given in Fig. 7. Memory usage of TFRC-Mine consists of storing the FRI-list and the top-k list and there is no advantage for sparse databases. Indeed, with sparse data, there is more chance to have long and non-closed patterns with low support and thus TFRC-Mine has to maintain more than k patterns for a while. However, as expected, TFRC-Mine* is more efficient than MTKPP, TR-CT and TKRIMPE for every database. In addition, for dense databases TFRC-Mine is more efficient than other algorithms thanks to the PDBV representation. Indeed, the support of the kth pattern in the top-k list has more chance to be high and there is less patterns with the same support. Thus, there is more chance to maintain a list as short as possible *i.e.* with k entries. This is particularly true for very dense database like the Mushroom database.

Fig. 8 illustrates that the variation of the regularity threshold has limited impact on TFRC-Mine memory consumption. As discussed for runtime (Fig. 6) high regularity may lead to large set of candidate regular patterns. In particular, more memory is used to store single items in the FRI-list.

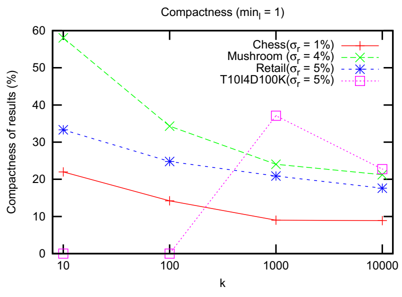


Fig. 10. Compactness of Top-k frequent-regular closed patterns.

6.5. Average length of results

Fig. 9 shows the average length of the discovered patterns. It is clear that TFRC-Mine* behave like MTKPP. Both algorithms tend to mine short patterns. Fig. 9 illustrates also the ability of TFRC-Mine to mine longer patterns. It should be noticed that the average length of patterns is relatively stable when $\texttt{min}_1$ is high (whatever the regularity threshold). Indeed, patterns with more than $\texttt{min}_1$ items tend to be less and less frequent and thus can not be a result thanks to the support constraint.

6.6. Compactness of top-k frequent-regular closed patterns

Fig. 10 shows the compactness factor of the closed-based approach. Thus, comparison is done between MTKPP and $\mathrm{TFRC}-\mathrm{Mine}_1$. Compactness corresponds to the percentage of top-k frequent-regular patterns that are covered by the top-k frequent-regular closed patterns (the higher the better).

In most cases, compactness is high. The closed-based approach can significantly reduce redundancy in the set of top-k patterns. Compactness decreases when k increases. Indeed, when k increases, there is more chance to mine non-redundant patterns. When compactness is very low it means that there is very few closed patterns in the top-k frequent-regular patterns (*e.g.* for the *T10I4D100K* database with $k \leqslant 100$).

7. Conclusion

Frequent and regular or periodic pattern mining is an important task in many applications. We here considered frequent and regular patterns where a pattern is said to be regular, given a user-given regularity threshold σ_r, if it appears at least once every σ_r transactions. Most of the similar approaches require to set a minimum support threshold which is known to be difficult, few of them consider the redundancy issue and/or the relationship between items of a pattern. None of them consider at the same time all these aspects.

We have proposed TFRC-Mine, a new algorithm to mine the top-k frequent-regular closed patterns of minimal length. It has several advantages and practical implications. The user has to set the values of meaningful parameters only (k, the regularity threshold and the minimum length). The top-k approach avoids to generate a large amount of patterns. The closed property aims to remove redundant patterns and also allows to recover interesting patterns, those that could have been removed because of redundant ones. Last, the constraint on the pattern's length aims to enhance the interestingness of the discovered patterns. As a whole, our proposal reduce the number of patterns and enhance the quality of each pattern.

TFRC-Mine uses a new compressed bit-vector which allows to prune candidates which can not be regular, and fast computation of the support and the regularity values. In addition, the algorithm scan the database only once. Its efficiency, for both running time and memory consumption, is formally demonstrated and is verified through several experiments. Clearly, TFRC-Mine is very efficient for sparse databases.

However, as formally discussed (and illustrated by the experiments) a first limitation of TFRC-Mine is that there is very few gain for dense databases. In addition, the top-k list is in memory until the end of the process (that is also why the database is scanned only once) and thus TFRC-Mine can manage only databases such that their compressed bit-vector representation hold in memory.

Perspectives of this work are of three kinds. The first one is qualitative. It includes establishing connection between regularity and periodicity. The later may force stronger constraints on when a pattern should appear. At the opposite, depending of the application domain, one can be interested in additional interestingness measures allowing some soft temporal constraints. Items' utility is obviously to be considered, especially with monotone constraints. The second perspective is to extend the algorithm for incremental databases and data streams. Last because each partition can be managed locally a third perspective is to develop a parallel and distributed version of TFRC-Mine.

Acknowledgment

The authors would like to thank the Junior Research Fellowship Program –French Embassy of Thailand–, the Faculty of Informatics of the University of Burapha –Thailand– and the UMR 6285 Lab-STICC –France– for providing support to this research.

References

Amir, A., & Levy, A. (2012). Approximate period detection and correction. In *Proceedings of the 19th international symposium on string processing and information retrieval, October 21–25, Cartagena de Indias, Colombia. Lecture notes in computer science* (Vol. 7608, pp. 1–15). Springer.

Amphawan, K., Lenca, P., & Surarerks, A. (2012). Efficient mining top-k regular-frequent itemset using compressed tidsets. In *Proceedings of international workshops on new frontiers in applied data mining, May 24–27, 2011, Shenzhen, China. Lecture notes in computer science* (Vol. 7104, pp. 124–135).

Amphawan, K., Lenca, P., & Surarerks, A. (2009). Mining top-k periodic-frequent patterns without support threshold. In *Proceedings of the 3rd international conference on advances in information technology, December 1–5, Bangkok, Thailand. Communications in computer and information science* (Vol. 55, pp. 18–29). Springer.

Amphawan, K., Lenca, P., & Surarerks, A. (2012b). Mining top-k regular-frequent itemsets using database partitioning and support estimation. *Expert Systems with Applications, 39*(2), 1924–1936.

Amphawan, K., Surarerks, A., & Lenca, P. (2010). Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree. In *Proceedings of the 3rd international conference on knowledge discovery and data mining, January 9–10, Phuket, Thailand* (pp. 245–248). IEEE Computer Society.

Burdick, D., Calimlim, M., & Gehrke, J. (2001). MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th international conference on data engineering, April 2–6, 2001, Heidelberg, Germany* (pp. 443–452). IEEE Computer Society.

Chang, J. (2011). Mining weighted sequential patterns in a sequence database with a time-interval weight. *Knowledge-Based Systems, 24*(1), 1–9.

Dong, J., & Han, M. (2007). BitTableFI: An efficient mining frequent itemsets algorithm. *Knowledge-Based Systems, 20*(4), 329–335.

Engler, J. (2008). Mining periodic patterns in manufacturing test data. In *International conference I EEE Southeast Con* (pp. 389–395). IEEE Computer Society.

Fu, A. W. -C., Kwong, R. W. -w, & Tang, J. (2000). Mining n-most interesting itemsets. In *Proceedings of the 12th international symposium on foundations of intelligent systems, October 11–14, Charlotte, USA. Lecture notes in computer science* (pp. 59–67).

Glynn, E. F., Chen, J., & Mushegian, A. R. (2006). Detecting periodic patterns in unevenly spaced gene expression time series using lombscargle periodograms. *Bioinformatics, 22*(3), 310–316.

Goethals, B., & Saki, M. J. (2003). FIMI'03: Workship on frequent itemset mining implementations. In *Proceedings of the ICDM 2003 workshop on frequent itemset mining implementations, December 19, Melbourne, Florida, USA. CEUR workshop proceedings*. CEUR-WS.org.

Han, J., Wang, J., Lu, Y., & Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. In *Proceedings of 2002 IEEE international conference on data mining (ICDM 2002), 9–12 December 2002, Maebashi City, Japan* (pp. 211–218).

Khaleel, M., Dash, G., Choudhury, K., & Khan, M. (2015). Medical data mining for discovering periodically frequent diseases from transactional databases. In *Computational intelligence in data mining – Vol. 1. Smart innovation, systems and technologies* (Vol. 31, pp. 87–96).

Kiran, R. U., & Kitsuregawa, M. (2013). Discovering quasi-periodic-frequent patterns in transactional databases. In *Proceedings of the 2nd international conference on big data analytics, December 16–18, Mysore, India. Lecture notes in computer science* (Vol. 8302, pp. 97–115).

Kiran, R. U., & Kitsuregawa, M. (2014). Novel techniques to reduce search space in periodic-frequent pattern mining. In *Proceedings of the 19th international conference on database systems for advanced applications, April 21–24, Bali, Indonesia. Lecture notes in computer science* (vol. 8422, pp. 377–391).

Kiran, R., & Kitsuregawa, M. (2015). Discovering chronic-frequent patterns in transactional databases. In *Databases in networked information systems. Lecture notes in computer science* (Vol. 8999, pp. 12–26).

Kiran, R. U., & Reddy, P. K. (2010). Towards efficient mining of periodic-frequent patterns in transactional databases. In *Proceedings of the 1st international conference on database and expert systems applications, August 30–September 3, 2010, Bilbao, Spain* (Vol. 6262, pp. 194–208).

Kumar, G., & Kumari, V. (2012). Sliding window technique to mine regular frequent patterns in data streams using vertical format. In *Proceedings of 2012 IEEE international conference on computational intelligence computing research (ICCIC), December 18–20, Tamilnadu, India* (pp. 1–4).

Li, Z., Ding, B., Han, J., Kays, R., & Nye, P. (2010). Mining periodic behaviors for moving objects. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, July 25–28, Washington, DC, USA* (pp. 1099–1108).

Luo, X., Yuan, H., & Luo, Q. (2013). On discovering feasible periodic patterns in large database. In *Proceedings of the 11th IEEE international conference on dependable, autonomic and secure computing, December 21–22, Chengdu, China* (pp. 344–351).

Luth, S., Herkel, J., Kanzler, S., Frenzel, C., Galle, P. R., Dienes, H. P., et al. (2008). Serologic markers compared with liver biopsy for monitoring disease activity in autoimmune hepatitis. *Journal of Clinical Gastroenterology, 42*(8), 926–930.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th international conference on database theory, January 10–12, Jerusalem, Israel* (Vol. 1540, pp. 398–416).

Railean, I., Lenca, P., Moga, S., & Borda, M. (2013). Closeness preference - a new interestingness measure for sequential rules mining. *Knowledge-Based Systems, 44*, 48–56.

Rashid, M., Karim, M., Jeong, B. -S., Choi, H .-J. (2012). Efficient mining regularly frequent patterns in transactional databases. In *Proceedings of the 17th international conference on database systems for advanced applications, April 15–19, Busan, South Korea* (Vol. 7238, pp. 258–271).

Shenoy, P., Haritsa, J. R., Sudarshan, S., Bhalotia, G., Bawa, M., Shah, D. (2000). Turbo-charging vertical mining of large databases. In *Proceedings of ACM SIGMOD international conference on management of data, May 16–18, Dallas, USA* (pp. 22–33).

Song, W., Yang, B., & Xu, Z. (2008). Index-BitTableFI: An improved algorithm for mining frequent itemsets. *Knowledge-Based Systems, 21*(6), 507–513.

Soulas, J., & Lenca, P. (2015). Periodic episode discovery over event streams. In *Proceedings of the 17th portuguese conference on artificial intelligence, September 8–11, Coimbra, Portugal. Lecture notes in computer science*.

Soulas, J., Lenca, P., & Thépaut, A. (2013). Monitoring the habits of elderly people through data mining from home automation devices data. In *Proceedings of the 16th Portuguese conference on artificial intelligence, September 9–12, Angra do Heroísmo, Azores, Portugal. Lecture notes in computer science* (Vol. 8154, pp. 343–354).

Sreedevi, M., & Reddy, L. (2013). Mining regular closed patterns in transactional databases. In *Proceedings of the 7th international conference on intelligent systems and control, Jan 4–5, Coimbatore, India* (pp. 380–383).

Surana, A., Kiran, R. U., Reddy, P. K., 2012. An efficient approach to mine periodic-frequent patterns in transactional databases. In *Proceedings of international workshops on new frontiers in applied data mining, May 24–27, 2011, Shenzhen, China. Lecture notes in computer science* (Vol. 7104, pp. 254–266).

Tanbeer, S. K., Ahmed, C. F., Jeong, B. -S., & Lee, Y. -K. (2009). Discovering periodic-frequent patterns in transactional databases. In *Proceedings of 13th Pacific–Asia* *conference on advances in knowledge discovery and data mining, April 27–30, Bangkok, Thailand. Lecture notes in computer science* (pp. 242–253).

Tanbeer, S. K., Ahmed, C. F., Jeong, B. -S. (2010). Mining regular patterns in data streams. In *Proceedings of the 15th international conference on database systems for advanced applications, April 1–4, Tsukuba, Japan. Lecture notes in computer science* (Vol. 5981, pp. 399–413).

Tanbeer, S. K., Ahmed, C. F., Jeong, B. -S. (2010). Mining regular patterns in incremental transactional databases. In *Proceedings of the 12th international Asia–Pacific web conference, April 6–8, Buscan, Korea* (pp. 375–377).

Vo, B., Hong, T.-P., & Le, B. (2012). DBV-Miner: A dynamic bit-vector approach for fast mining frequent closed itemsets. *Expert Systems with Applications, 39*(8), 7196–7206.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the 3rd SIAM international conference on data mining, May 1–3, San Francisco, CA, USA* (pp. 166–177).