Periodic event sets detection in temporal databases

© *Ekaterina Ivannikova*
Saint Petersburg State University
ivannikovae@gmail.com

Abstract. Temporal data regularity is an important data property that can be used in different applications. Such regularity is explored in the field of periodic pattern mining. In this paper, we raise a problem of periodic sets detection and suggest the method for its solution. The existing algorithms for the periodic event mining are considered in detail and a new approach is proposed in the paper. The comparison of the algorithms and their performance are demonstrated through a series of experiments.

Keywords: periodic set; pattern mining; temporal mining; data regularity

1. Introduction

Pattern mining is an important area of data mining, and it has been growing rapidly over the two past decades. The initial stimulus for the development of methods was the problem of market basket analysis, that requires to determine which products are usually purchased together by using a transaction database of supermarket buying. The new knowledge could be used for the correct placement of goods or for the advertising purpose. The first efficient algorithm for finding frequent patterns has been proposed in [1]. Now there are many algorithms for detecting the patterns that are applied in various fields. For example, they are used in biology to identify the sequences of nucleotides, in the analysis of log files for the detection of failures or attacks, in medicine for diagnosis, in marketing for advertising or tips for users, etc.

Many kinds of the data in the real world are time series or temporal databases. Periodic pattern mining is the problem that regards temporal regularity. Periodic patterns are characterized by a certain persistence and predictability. Information about such regularity can be used for many tasks: for the purpose of personal promotion to the users, for the timely reminders about the future events or forecasting.

By the periodic pattern we mean the set of items that frequently occur together at regular time intervals. There are two ways for prediction if the pattern and its period p are known. If the pattern has occurred during some time interval (t_{beg}, t_{end}) then it is likely to repeat during (t_{beg}+ p, t_{end} + p). And if several events of the pattern have

occurred then other events of the pattern are likely to happen soon. Therefore, it is important to identify and describe the periodicity.

There are several types of periodic patterns considered in literature. Most of the studies on their mining apply the Apriori property heuristic and adopt some variations of Apriori-like mining methods [9]. Apriori property is used to reduce the search space and formulated as "All nonempty subsets of a frequent itemset must also be frequent", i.e. an itemset is frequent only if all of its sub-itemsets are frequent. This observation applies to construct k+1-patterns based on the found k-patterns set. But 1-patterns are generally detected by a simple search and are not consider in detail. In addition, in most of the previous works the 1-pattern consists of a single item while the pattern that consists of a number of events may be interested in some tasks. In this work we present an algorithm for such periodic 1-patterns finding and discuss several approaches to periodic event detection. We propose a new algorithm to the periodic event mining as well. The experiments show that our method is the more advantageous in some cases.

The remainder of this paper is organized as follows. In section 2 the previous works on the frequent and periodic pattern mining will be discussed. In section 3 the notation used throughout the paper and the formal definition of periodic patterns are introduced. Section 4 describes the several possible ways of patterns detection and section 5 presents the experimental results. The conclusion and future research directions are contained in section 6.

2. Related works

Our study combines frequent pattern mining and periodic pattern mining in periodic sets finding. We will review the related works in these areas below.

2.1. Frequent pattern mining

There are a huge number of studies in this field in literature. The existing algorithms can be classified into three main groups:

- iterative Apriori-like level-wise mining techniques
- pattern mining methods without candidate generation
- mining techniques with the vertical data format

As mentioned above, the concept of frequent itemset and the first algorithm for finding frequent patterns by using downward closure property, called Apriori, were introduced by Agrawal and Srikant in [1]. This approach requires candidate set generation and scanning the database to check each candidate. Such techniques as hashing [14], partitioning [15], upper bound of the number of candidate patterns that can be generated in the level-wise approach [5] were developed for improvements and extensions of algorithms in that category [7].

In paper [10] a problem of finding long frequent patterns is considered and a new FP-growth approach that mines frequent itemsets without candidate generation was

devised. At the first step items are ordered in frequency-descending order and according to this list, the database is compressed into a special FP-tree and after that the tree is mined in some way. An alternative approach is proposed in [6].

These methods are used to discover patterns from a set of transactions in a horizontal data format (i.e. {Transaction_id :itemset}). An example of alternative class of the algorithms that first transforms the data into a vertical data format (i.e. {item :Transaction_id set}) is Eclat [18]. These methods don't require scanning the database to count the support of (k+1)-candidates.

2.2. Periodic pattern mining

The problem of mining periodic patterns can be viewed from different perspectives [9]: depending on the coverage of the pattern there are full- and partial-periodic patterns. Basing on the precision of the periodicity synchronous and asynchronous pattern can be identified. And a pattern can also be precise or approximate.

Generally, in the works related with our, the following notation of periodic pattern is used. Let S be the sequence $S = S_1, S_2, \ldots, S_n$. A pattern p is the nonempty sequence $p = p_1 \ldots p_k$, where $p_i \subset S_j \bigcup \{*\}$. The additional event {*} – symbol that matches any event and is used to represent the "don't care" position in the pattern. The frequency of a pattern p is the count of j such that the string s is true in $S_{i|p|+1} \ldots S_{i|p|+|p|}$. If the frequency of the p exceeds a minimum support threshold, then this pattern is named periodic. A pattern with the k non-{*} positions is also called an i-pattern.

Cyclic association rules that display regular cyclic variation over time were first introduced in [13]. These rules are based on partial-periodic patterns with perfect periodicity in the sense that each pattern reoccurs in every cycle. In the article several techniques called cycle-pruning, cycle-skipping and cycle-elimination were proposed to optimize mine periodic patterns with 100% support.

General partial-periodic patterns with imperfect periodicity studied in [3, 4, 8] are more common in real world. In work [3] a new structure named abbreviated list table (ALT) that maintains the occurrence counts of all symbols and corresponding periods was proposed. Using this structure only a small number of data sequence passes are required to compute the periods and the patterns of size 1. Han et al [8] explored interesting properties such as the Apriori property and the max-subpattern hit set property related to partial periodicity and proposed several methods for efficient mining of k-patterns. Another original algorithms for symbol and segment periodicity detection based on mapping scheme and convolution may be found in [4].

The above methods were used to discover potential periods from the entire time-series data. In paper [16] the authors presented the dense periodic patterns that may exist only in a limited range of the time-series.

Most of the works are concentrated on the k-patterns constructing and use a base line algorithm for 1-patterns mining. To the best of our knowledge only paper [3]

represents the way to improve 1-patterns mining. In our work we propose a new approach to the 1-patterns mining and perform a comparison analysis with the existing base line and ATL approaches. In addition, most of the existing studies focused on the detection of such patterns where S_i and p_j are single events while we explore the patterns where p_i may be a set of events.

Another group of works [11, 12, 17] is focused on the regular activities that occur with a calendar-based periodicity. In [12] the calendar schema is proposed to construct periodic patterns. Calendar-based approach allows to specify multiple time granularities. For example, schema (2010, *, 1) means that the pattern occurred on the first day of each month in 2010. A fuzzy periodic calendar and an algorithm for mining fuzzy periodicity are developed in [11].

3. Problem definition

Let E be a set of events. Time-series S is a sequence of records (t_i, e_i) with an event $e_i \in E$ and a time stamp t_i when the event occurred.

We assume that the pattern time interval is limited by user-specified window W, i.e. all events in pattern occur within the interval W. Time-series S can be presented as $S = S_1\ S_2 \ldots S_n$, where S_i are sequential disjoint sets of the events happening during the window. For example, for W equal to an hour, S_1 may contain the events which occurred from 12 am to 13 am, S_2 contains the events that took place from 13 to 14 am, and so on.

Let T be a subset of E. The set T is contained in S_i, if all the events of T are in S_i. The ***binary sequence*** for itemset T ($BinSeq_T$) is constructed as follow: $BinSeq_T[i] = 1$ if T is contained in S_i, and $BinSeq_T[i] = 0$ otherwise. The ***candidate pair*** (p, o) with period p and offset o has a ***support*** Sup of the period (p, o) for the set T:

$$Sup(p,o) = \frac{|\, i : BinSeq_T[i*p+o] = 1 \,|}{|\, BinSeq_T[i*p+o] \,|} * 100\% \quad \text{, where}$$

$$|\, BinSeq_T[i*p+o] \,| \;=\; \frac{Size(BinSeq_T)}{p} \qquad , i*p+o \;\le Size(BinSeq_T)$$

The notation (p, o, sup) will be used if it is known that the candidate pair (p, o) has the support sup. We say that the set T is a ***periodic 1-pattern*** with period p and offset o if Sup(p, o) is not less than a specified minimum support. The length of a 1-pattern is the number of elements in the pattern. The 1-pattern of length 1 we will also be called a ***periodic event*** and the 1-pattern of length k greater than 1 we will be referred to as a ***periodic k-set*** for short. Our goal is to find all possible periodic sets in a given time-series.

4. Algorithm

Our algorithm belongs to a group of methods using an iterative approach known as a level-wise search. This approach uses k-patterns to generate (k+1)-patterns. At the first step of the algorithm the set of periodic events is found by searching the binary sequences. Further, the periodic sets with two items can be constructed and tested for the periodicity (i.e. which of them satisfy a minimum support), and so on, until no more periodic k-sets can be detected.

The following predefined parameters are used in the algorithms: minimum period (P_min), maximum period (P_max), minimum support (Sup_min).

4.1. Periodic items

Three methods for the periodic events detection will be discussed below. The first method is the base line algorithm that is used in most of existing works. In the second approach we adopt the idea of ATL structure described in [3]. The third approach represents a new way to periodic item detection. The periodic 1-sets or periodic events are found using binary sequences of events. So, for each event it is needed to construct the binary sequence as described in section 3.

Base line approach.

This method requires counting the support value for all possible periods-candidates as shown in fig. 1.

```
forP_min ≤ p ≤P_max
for 0 ≤ o < p
       if (Sup(p, o) ≥ Sup_min) then
       Result.AddPeriod(p, o)
```

Fig. 1

Such an approach requires the full scan of the binary sequence for the testing of some period. To check all the candidates *(P_max – P_min + 1)* passes are needed for each event. If n is the length of the binary sequence, *⌊n/p⌋* steps are required to check one candidate (p, o). There are p candidate pairs with period p and different offsets. To test them $p * \lfloor n/p \rfloor \sim n$ steps are required. And the number of interesting periods is *(P_max-P_min+1)*. So, the algorithm is performed in time *O((P_max-P_min+1) * n)*.

ATL-based approach.

In this case only one full scan of the sequence is required. The number of times each candidate pair occurs is counted by scanning.

Initially, count is equal to 0 for all the candidates. When a position i is considered, if BinSeq[i] = 1 then the count of the occurrences increases by one for all the cycles (p, o = i mod p).

For example, consider a binary sequence: BinSeq = 01001011001, P_min = 2, P_max = 5.

Let the algorithm scan sequence at position i=4: BinSeq [4]=1 => (2, 0).count++, (3, 1).count++, (4, 0).count++, (5, 4).count++, and so on while i less than the sequence length.

After the occurrences numbers of candidate periods have been counted the supports of pairs are calculated as follow:

$$Sup(p, o) = (p, o).counts / \frac{Size(BinSeq)}{p} * 100\%$$

This method is described below in Fig 2.

```
Result; // the set of finding periods
for  0 ≤ I < Size(BinSeq)
if(BinSeq[i]=1) then
       forP_min ≤ p ≤ P_max
        (p, i mod p).count++;
forP_min ≤ p ≤P_max
for 0 ≤ I < p
       if(sup(p, o) ≥ Sup_min) then
       Result.Add(p, o);
```

Fig. 2

Time complexity of this approach is *O(n).*

Divider-based pruning approach.

This method is based on the following observation: if there is a triple (p, o, sup), then a triple (p_1, d_1, sup_1) = (p/d, o, sup/d) with smaller period and support also exists. To be more precise d is a divider of p and sup_1 no less than sup/d. So, in order for a triple (p, o, sup) to exist, the existence of triple (p_1 = p/d, o, $sup_1 \geq$ sup/d) calculated at the previous steps is necessary. Support values that were counted for smaller periods can be used to reduce computing at the next iterations for larger periods mining. If for some candidate (p, o) the support is equal to *sup* and *sup* < *Sup_min*, then the candidate (p*i, o) is not suitable in case *p*i* < *P_max* and *sup*i* < *Sup_min.*

For example, at some iteration of the algorithm a candidate pair (4, 3) is tested on periodicity with the minimum support 80 %. If the support of the candidate is 30% then candidate (8, 3) may be excluded from further consideration.

The second observation we will use is that if the period (p, o) is found, then the multiple periods (p*i, o) are not so interesting.

At the beginning of the algorithm we assume that there are all possible periods, i.e. pairs (p, o). Denote this set as Cand. Pseudocode of the algorithm is given in Figure 3.

```
Result; // the set of finding periods
while (not Cand.Empty())
  (p, o) = Cand.GiveNextPair();
sup = Sup(p, o);
if (sup ≥ Sup_min) then
Result.Add(p, o);
i = 1;
    while (p*i ≤ P_max)
    Cand.Remove(p*i, o);
else
       i = 1;
       while (p*i ≤ P_max and
sup*I < Sup_min)
       Cand.Remove(p*i, o);
       i++;
```

Fig. 3

Algorithm evaluation:

1) In the algorithm a check of all candidates with the prime periods is required. Such pairs couldn't be removed from the set of candidates at the previous steps of the algorithm.

The number of primes less than N is estimated approximately as *N/ln(N)*. Consequently, the number of prime periods in the range of P_min to P_max equal to *K = P_max/ln(P_max) – P_min/ln(P_min)*. To compute the support of pairs with period p and all possible offsets the scan of the entire binary sequence is required. Thus, to test all prime periods, we need to perform *K*n* steps.

2)Let's consider what candidates with the composite periods should be treated on the average. There are *(P_min + P_max)/2 * (P_max - P_min +1)* pairs with period p from P_min to P_max and the corresponding offset from 0 to p-1.

The number of prime periods is equal to K. The mean period is *(P_min + P_max)/2* and the number of candidates with this period and different offsets is *(P_min + P_max)/2* also. We estimate the count of candidates with prime periods as *K*(P_min + P_max)/2*. Therefore, the number of candidates with the composite periods may be computed like that:

$$\frac{(P_{\min} + P_{max})}{2} * (P_{max} - P_{min} + 1) - K * \frac{(P_{\min} + P_{max})}{2} =$$

$$\frac{(P_{\min} + P_{max})}{2} * (P_{max} - P_{min} + 1 - K)$$

We assume that a half of these pairs on the average are excluded from consideration, i.e. from the candidate set, at the previous steps of the algorithm. To check a candidate with period p it is required to scan n/p elements of the binary sequence. Consequently, to check a candidate with the mean period it is required *n / (P_min+P_max) / 2 = 2*n / (P_min+P_max)* elements. So, to check the candidates with composite periods it is needed to take

$$\frac{1}{2} * \frac{(P\_\min + P\_max)}{2} * (P_{max} - P_{min} + 1 - K) * \frac{2 * n}{(P_{min} + P_{max})} =$$

$$= *(P\_max - P\_min + 1 - K)*n \text{steps on the average.}$$

3) So, summing the results in 1) and 2) for evaluating prime and composite periods respectively we conclude that the algorithm perform
*K*n + *(P_max – P_min + 1 - K)*n = *(P_max – P_min + 1+K)*n* steps. And the time complexity of this approach is *O(*(P_max – P_min + 1+K)*n)* in the mean.

4.2. Periodic k-sets

The Apriori property can be adapted to periodic sets: All nonempty subsets of a periodic set must also be periodic sets with the same periods and offsets. So, for k+1-sets generation we will use the k-sets.

Let $P_k(p, o)$ be the collection of k-sets having period p with offset o. This set contains the patterns with their binary sequences. The order of elements in the patterns is not significant, and we will keep items in lexicographic order. In this case we apply the join step proposed by Agrawal, etc. in [1] to the candidate k+1-

sets (denote this set as $C_{k+1}(p, o)$) generation. P_k is joined with P_k in the following way:

$$\text{Let } p = p_1 p_2 \ldots p_k,\ q = q_1 q_2 \ldots q_k \in P_k$$

$$\text{If p, q such that } p_1 = q_1, \ldots, p_{k-1} = q_{k-1}, p_k < q_k, \text{ then}$$

$$c = p_1 p_2 \ldots p_k q_k \in C_{k+1}$$

We will store k-sets with their binary sequences. $BinSeq_p$ is the binary sequence for k-pattern P and $BinSeq_{qk}$ is the binary sequence of item q_k. Support of the candidate set $c = p_1 p_2 \ldots p_k q_k$ is calculated as:

$$Sup_c(p, o) = \frac{|\, i : BinSeq_p[i * p + o] = 1 \;\&\; BinSeq_{qk}[i * p + o] = 1 \,|}{|\, BinSeq_p[i * p + o] \,|} * 100\%$$

$$|\, BinSeq_p[i * p + o] \,| = \frac{Size(BinSeq_p)}{p}$$

If the support of the candidate c is more than the given minimum support, then $c \in P_{k+1}(p, o)$. All k+1-candidates are tested and the set of k+1-sets is composed. Similarly, the collection of k+2-sets is obtained from the set of k+1-patterns and so on.

Let's estimate the time required to generate $P_{k+1}(p, o)$ from the set $P_k(p, o)$. Let m be the number of the patterns in $P_k(p, o)$. The count of the candidate k+1-sets deriving at the join step can be evaluated as (m-1) + (m-2) + … + 1 = m(m-1)/2 in the worst case. If n is the binary sequence length, then 2*n/p steps are required to check the one candidate. So, the algorithm perform m(m-1)*n/p steps during the construction of the $P_{k+1}(p, o)$ set from $P_k(p, o)$.

So, we have shown the method for periodic k-sets mining above. Note that the periodic k-patterns can be obtained from the found periodic sets using known methods for periodic pattern mining.

5. Experiments

5.1. Data generation

For our experiments we used synthetic data. The time-series were generated by tuning the following parameters: the beginning date and the end date of the sequence, the number of different events (|E|), the length of time-series (i.e. total number of entries in a file), the count of periodic sets in series, the minimum and maximum periods, the minimum and maximum window for periodic sets, the minimum and maximum support of periodic sets.

Note that in addition to the known generated periodic sets the time-series may contain a number of others patterns. These periods are formed by noise events, which correspond to the real data. Some of the periods may be obvious, well-known or uninteresting, but the task of the revelation of interesting and useful periodic patterns is not in the scope of this work.

The events in the data have different frequency. |E| is the number of different (noisy) events. We have an ordered list of the events. At some moment of the time an event with a sequential number N occurs. The number N is calculated as *N = random.Next(random.Next(|E|))*.

So, the smaller the event ordinal number, the more frequently it happens. The intervals between successive events in the generated data are the same.

5.2. Experiment performance

The algorithms described for periodic events detection are denoted as BL (Base line Approach), ATL (ATL-based approach) and DBP (Divider-based pruning approach).

Fig. 1 illustrates that the behavior of the algorithms depends on the time-series frequency. Time-series frequency (F_S) is a value that indicates how many events occur in a time unit or a patterns window. The other parameters of the data: data for the time span in a month, 1200 different events, the window size for patterns from 10 minutes to an hour, the periods from 3 hours to a week, the support of the generated patterns ranges from 60 to 100%. The algorithm works with P_min= 3 hour, P_max= 1 week, Sup_min= 70%.

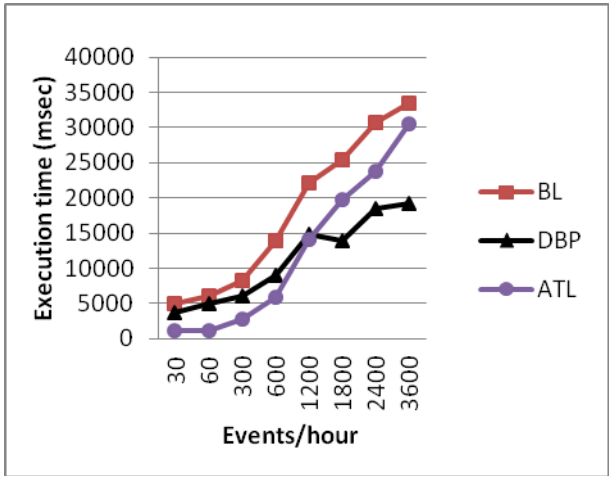

Fig. 1

Figure 1 shows a significant efficiency gain by DBP and ATL over base line approach BL. These algorithms allow to improve the execution time by 1,6-5,4 times. ATL is more efficient for a smaller number of entries an hour while it is opposite with DBP. We assume that the algorithms performance really depends on the ratio of different events number and time-series frequency, and not only on the frequency of series. It is confirmed by the second experiment (Fig. 2), where the size of events set E is changed for a fixed frequency (20 events per minute).

For our data if the rate $|E|/F_S < 0,7$ then the algorithm ATL is more efficient, and it is otherwise if $|E|/F_S > 0,7$ then DBP.

Fig. 3 gives execution time against range of periods. The minimum period is set at three hour and the maximum period value varies from 24 to 168 hours. All algorithms are ecexuted longer with increasing the range. But DBP execution time grows slower by 1,3-1,5 times as compared to the other two.

The scalability of the algotithms with respect to the analyzed data size is displayed in Fig. 4. We compare the performance of each approach for data from 8 to 96 Mb, which corresponds to the data period from 1 to 12 months if the time-series has 5 events per minute on average and 900 different events. The graphs show that three algorithms have scalability close to linear. However, ATL execution time increase grows about 1,6 times more slowly than BL and 2 times more slowly than DBP for such data.

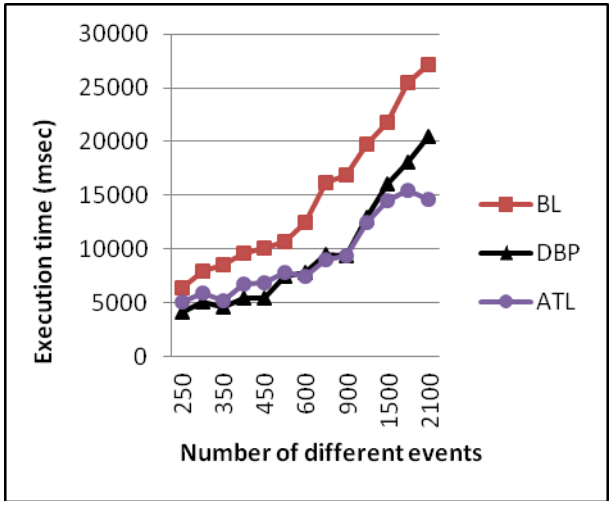

Fig. 2

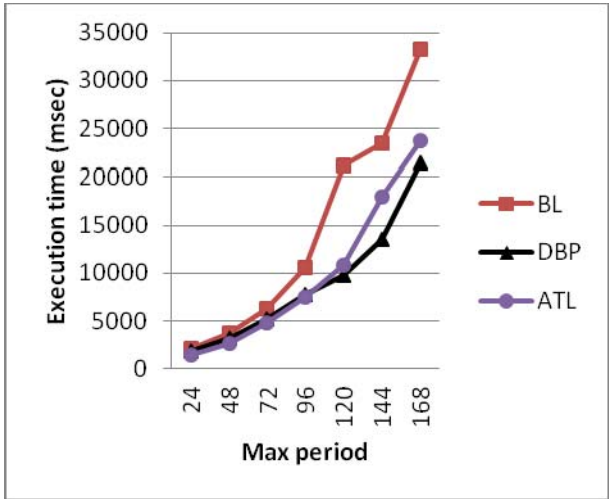

Fig. 3

Obviously, the generation of k+1-sets depends largely on the previous step, i.e. how many k-sets were found (the greater the number of k-sets, the more time is required for k+1-sets detection) while the generation time of the 1-sets relies on the input data size mostly. Therefore the stage of the periodic event extracting may take a significant part of periodic set mining algorithm and the improvement of this step can accelerate the algorithm performance as a whole. Fig. 5 shows the performance of the algorithm against the number of 1-sets. For step of 1-sets detection we use the BL algorithm.

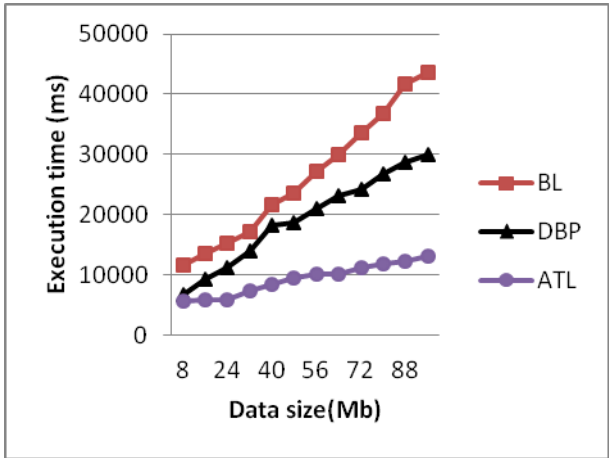

Fig. 4

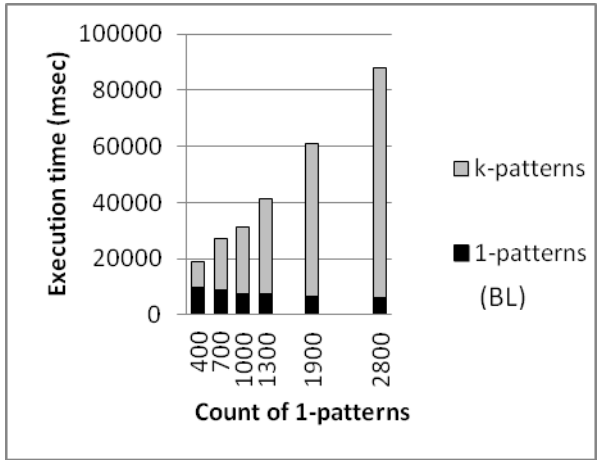

Fig. 5

Unlike the BL and ATL methods, using the introduced DBP approach in order to mine the periodic events allows to reduce the time of the k-patterns discovery as well. It is achieved due to the removal of multiple periods. For the same data in previous experiment the k-sets generation after the first step with DBP is 2-12% faster than after BL.

6. Conclusion

In this work the problem of periodic 1-patterns finding was considered. We represent the approach to periodic sets generation relying on the methods of frequent pattern mining. The new algotithm DPA for the periodic item mining was introduced as well as its evaluation was determined. We considered in detail the other existing approaches and compared them with the proposed one. The series of experiments shows that the proposed algorithms DPA can give up to a hundreds percent increase in performance of 1-patterns mining over the base line approach used in most of the previous studies. The our algorithm is the more advantageous then ATL in some cases also. The experimental comparison of the existing methods with different data and parameters is described in the paper.

In the future work it is interesting to analyze the memory management and explore the algorithms performance on the real and big data. Although the BL requires more time, it is no need to store the additional structures as DBP or ATL. Other directions for the future work are the solution of useful periodic patterns detection problem and developing the parallel extensions of the algorithms.

References

[1] RakeshAgrawal and RamakrishnanSrikant. Fast algorithms for mining association rules in large databases. In VLDB, pages 487-499, 1994.

[2] Juan M. Ale and Gustavo Rossi. Discovering association rules in temporal databases. In Encyclopedia of Database Technologies and Applications, pages 195-200. 2005.

[3] Huiping Cao, David W. Cheung, and Nikos Mamoulis. Discovering partial periodic patterns in discrete data sequences. In PAKDD, pages 653-658, 2004.

[4] Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. Periodicity detection in time series databases. IEEE Trans. Knowl. Data Eng., 17(7):875-887, 2005.

[5] FlorisGeerts, Bart Goethals, and Jan Van den Bussche. A tight upper bound on the number of candidate patterns. In ICDM, pages 155-162, 2001.

[6] GöstaGrahne and Jianfei Zhu. E-ciently using prex-trees in mining frequent itemsets. In FIMI, 2003.

[7] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov.,15(1):55-86, 2007.

[8] Jiawei Han, Guozhu Dong, and Yiwen Yin. E-cient mining of partial periodic patterns in time series database. In ICDE, pages 106-115, 1999.

[9] Jiawei Han and MichelineKamber. Data Mining: Concepts and Techniques, second edition. Morgan Kaufmann, 2000.

[10] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In SIGMOD Conference, pages 1-12, 2000.

[11] Wan-Jui Lee, Jung-Yi Jiang, and Shie-Jue Lee. Mining fuzzy periodic association rules. Data Knowl. Eng., 65(3):442_462, 2008.

[12] Yingjiu Li, PengNing, Xiaoyang Sean Wang, and SushilJajodia. Discovering calendar-based temporal association rules. In TIME, pages 111-118, 2001.

[13] BanuÖzden, Sridhar Ramaswamy, and Abraham Silberschatz. Cyclic association rules. In ICDE, pages 412-421, 1998.

[14] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. In Proceedings of the 1995 ACMSIGMOD international conference on Management of data, SIGMOD '95, pages 175-186, New York, NY, USA, 1995. ACM.

[15] Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe. An effcient algorithm for mining association rules in large databases. In VLDB, pages 432-444, 1995.

[16] Chang Sheng, Wynne Hsu, and Mong-Li Lee. Mining dense periodic patterns in time series data. In ICDE, page 115, 2006.

[17] Keshri Verma and Om Prakash Vyas. E-cient calendar based temporal association rule. SIGMOD Record, 34(3):63-70, 2005.

[18] Mohammed Javeed Zaki. Scalable algorithms for association mining. IEEETrans. Knowl. DataEng., 12(3):372-390, 2000.