The Journal of Systems and Software 112 (2016) 110–121

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Efficient discovery of periodic-frequent patterns in very large databases

R. Uday Kiran [a,b,*], Masaru Kitsuregawa [a,c], P. Krishna Reddy [d]

[a] The University of Tokyo, Japan
[b] National Institute of Information and Communications Technology, Japan
[c] National Institute of Informatics, Japan
[d] International Institute of Information Technology-Hyderabad, India

ARTICLE INFO

Article history:
Received 20 February 2015
Revised 12 August 2015
Accepted 26 October 2015
Available online 2 November 2015

Keywords:
Data mining
Knowledge discovery
Frequent patterns

ABSTRACT

Periodic-frequent patterns (or itemsets) are an important class of regularities that exist in a transactional database. Finding these patterns involves discovering all frequent patterns that satisfy the user-specified maximum periodicity constraint. This constraint controls the maximum inter-arrival time of a pattern in a database. The time complexity to measure *periodicity* of a pattern is $O(n)$, where n represents the number of timestamps at which the corresponding pattern has appeared in a database. As n usually represents a high value in voluminous databases, determining the *periodicity* of every candidate pattern in the itemset lattice makes the periodic-frequent pattern mining a computationally expensive process. This paper introduces a novel approach to address this problem. Our approach determines the periodic interestingness of a pattern by adopting greedy search. The basic idea of our approach is to discover all periodic-frequent patterns by eliminating aperiodic patterns based on suboptimal solutions. The best and worst case time complexities of our approach to determine the periodic interestingness of a frequent pattern are $O(1)$ and $O(n)$, respectively. We introduce two pruning techniques and propose a pattern-growth algorithm to find these patterns efficiently. Experimental results show that our algorithm is runtime efficient and highly scalable as well.

1. Introduction

Frequent pattern (or itemset) mining is an important knowledge discovery technique. It typically involves finding all patterns that are occurring frequently in a transactional database. Frequent patterns play a key role in discovering associations (Agrawal et al., 1993), correlations (Brin et al., 1997; Omiecinski, 2003), episodes (Mannila, 1997), multi-dimensional patterns (Lent et al., 1997), diverse patterns (Srivastava et al., 2011), emerging patterns (Dong and Li, 2009), and so on. The popular adoption and successful industrial application of frequent patterns has been hindered by a major obstacle: "*frequent pattern mining often generates too many patterns, and majority of them may be found insignificant depending on application or user requirements.*" When confronted with this problem in real-world applications, researchers have tried to reduce the desired set by finding user interest-based frequent patterns such as maximal frequent patterns (Gouda and Zaki, 2001), demand driven patterns Wang et al. (2005), utility patterns (Yao et al., 2004), constraint-based patterns (Pei et al., 2004), diverse-frequent patterns (Swamy et al., 2014), top-k patterns (Han et al., 2002) and periodic-frequent patterns (Tanbeer et al., 2009). This paper focuses on efficient discovery of periodic-frequent patterns.

An important criterion to assess the interestingness of a frequent pattern is its temporal occurrences within a database. That is, whether a frequent pattern is occurring periodically, irregularly, or mostly at specific time intervals in a database. The class of frequent patterns that are occurring periodically within a database are known as **periodic-frequent patterns**. These patterns are ubiquitous and play a key role in many applications such as finding co-occurring genes in biological datasets (Zhang et al., 2007), improving performance of recommender systems (Stormer, 2007), intrusion detection in computer networks (Ma and Hellerstein, 2001) and discovering events in Twitter (Kiran et al., 2015). A classic application to illustrate the usefulness of these patterns is market-basket analysis. It analyzes how regularly the set of items are being purchased by the customers. An example of a periodic-frequent pattern is as follows:

$$\{Bed, Pillow\} \quad [support = 10\%, \quad periodicity = 1\ hour].$$

The above pattern says that 10% of customers have purchased the items '*Bed*' and '*Pillow*' at least once in every hour. The basic model of periodic-frequent patterns is as follows (Tanbeer et al., 2009):

* Corresponding author at: The University of Tokyo, Japan. Tel.: +810354526254.
E-mail addresses: uday.rage@gmail.com, uday_rage@tkl.iis.u-tokyo.ac.jp (R.U. Kiran), kitsure@tkl.iis.u-tokyo.ac.jp (M. Kitsuregawa), pkreddy@iiit.ac.in (P.K. Reddy).
URL: http://researchweb.iiit.ac.in/~uday_rage/index.html (R.U. Kiran), http://www.tkl.iis.u-tokyo.ac.jp/Kilab/Members/memo/kitsure_e.html (M. Kitsuregawa), http://faculty.iiit.ac.in/~pkreddy/index.html (P.K. Reddy)

http://dx.doi.org/10.1016/j.jss.2015.10.035
0164-1212/

Table 1
Transactional database.

ts	Items	*ts*	Items
1	*ab*	6	*def*
2	*acdi*	7	*abi*
3	*cefj*	8	*cde*
4	*abfgh*	9	*abef*
5	*bcd*	10	*acg*

Let $I = \{i_1, i_2, \ldots, i_n\}$, $1 \leq n$, be the set of items. Let $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β number of items is called a $\boldsymbol{\beta}$**-pattern**. A **transaction**, $tr_k = (ts_k, Y)$, $1 \leq k$, is a tuple, where $ts_k \in \mathbb{R}$ represents the timestamp of Y pattern. For a transaction $tr_k = (ts_k, Y)$, such that $X \subseteq Y$, it is said that X occurs in tr_k and such timestamp is denoted as ts^X_k. A **transactional database** *TDB* over I is a set of transactions, $TDB = \{tr_1, \ldots, tr_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. Let $TS^X = \{ts^X_j, \ldots, ts^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where X has occurred in *TDB*. In this paper, we call this list of timestamps of X as **ts-list** of X. The number of transactions containing X in *TDB* is defined as the **support** of X and denoted as $Sup(X)$. That is, $Sup(X) = |TS^X|$. Let ts^X_q and ts^X_r, $j \leq q < r \leq k$, be the two consecutive timestamps in TS^X. The time difference (or an inter-arrival time) between ts^X_r and ts^X_q can be defined as a ***period*** of X, say p^X_a. That is, $p^X_a = ts^X_r - ts^X_q$. Let $P^X = \{p^X_1, p^X_2, \ldots, p^X_r\}$ be the set of periods for pattern X. The ***periodicity*** of X, denoted as $Per(X) = maximum(p^X_1, p^X_2, \ldots, p^X_r)$. The pattern X is a **frequent pattern** if $Sup(X) \geq minSup$, where *minSup* refers to the user-specified *minimum support* constraint. The frequent pattern X is said to be **periodic-frequent** if $Per(X) \leq maxPer$, where *maxPer* refers to the user-specified *maximum periodicity* constraint. The problem definition of periodic-frequent pattern mining involves discovers all those patterns in *TDB* that satisfy the user-specified *minSup* and *maxPer* constraints. Please note that both *support* and *periodicity* of a pattern can be described in percentage of $|TDB|$.

Example 1. Consider the transactional database shown in Table 1. This database contains 10 transactions. Therefore, $|TDB| = 10$. Each transaction in this database is uniquely identifiable with a timestamp (*ts*). The set of all items in *TDB*, i.e., $I = \{a, b, c, d, e, f, g, h, i, j\}$. The set of items '*a*' and '*b*', i.e., {*a, b*} is a pattern. For brevity, we represent this pattern as '*ab*'. This pattern contains two items. Therefore, it is a 2-pattern. The pattern '*ab*' appears at the timestamps of 1, 4, 7 and 9. Therefore, the list of timestamps containing '*ab*' (or *ts*-list of '*ab*') is {1, 4, 7, 9}. In other words, $TS^{ab} = \{1, 4, 7, 9\}$. The *support* of '*ab*' is the size of TS^{ab}. Therefore, $Sup(ab) = |TS^{ab}| = 4$. The periods for this pattern are: $p^{ab}_1 = 1$ $(=1 - ts_{initial})$, $p^{ab}_2 = 3$ $(=4 - 1)$, $p^{ab}_3 = 3$ $(=7 - 4)$, $p^{ab}_4 = 2$ $(=9 - 7)$ and $p^{ab}_5 = 1$ $(=ts_{final} - 9)$, where $ts_{initial} = 0$ represents the timestamp of initial transaction and $ts_{final} = |TDB| = 10$ represents the timestamp of final transaction in the database. The *periodicity* of *ab*, i.e., $Per(ab) = maximum(1, 3, 3, 2, 1) = 3$. If the user-defined $minSup = 3$, then '*ab*' is a frequent pattern because $Sup(ab) \geq minSup$. If the user-defined $maxPer = 3$, then the frequent pattern '*ab*' is said to be a periodic-frequent pattern because $Per(ab) \leq maxPer$. The complete set of periodic-frequent patterns discovered from Table 1 are shown in Table 2.

The space of items in a transactional database gives rise to an itemset lattice. Fig. 1 shows the itemset lattice for the items '*a*', '*b*' and '*c*.' This lattice is a conceptualization of search space while finding the user interest-based patterns. Tanbeer et al. (2009) have introduced a pattern-growth algorithm, called Periodic-Frequent Pattern-growth (PFP-growth), to mine the periodic-frequent patterns. This algorithm generates periodic-frequent patterns by applying depth-first search in the itemset lattice. From a singleton periodic-frequent pattern *i*, successively larger periodic-frequent patterns are discovered by adding one item at a time.

Table 2
Periodic-frequent patterns discovered from Table 1.

P	*Sup*	*Per*	*P*	*Sup*	*Per*
a	6	3	*f*	4	3
b	5	3	*ab*	4	3
c	5	3	*cd*	3	3
d	4	3	*ef*	3	3
e	4	3			

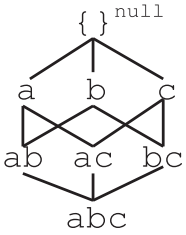

Fig. 1. Itemset lattice generated for the items '*a*, ' '*b*' and '*c*'.

The measure, *periodicity*, plays a key role in periodic-frequent pattern mining. This measure ensures that the anti-monotonic property (see Property 1) of frequent patterns still holds for periodic-frequent patterns. Measuring the *periodicity* of a pattern requires a complete scan on its ts-list. As a result, the time complexity of finding *periodicity* of a pattern is $O(n)$, where n represents the number of timestamps at which the corresponding pattern has appeared in a database. **As** n **typically represents a very large number in voluminous databases, measuring the** *periodicity* **of every candidate pattern in the huge itemset lattice makes the periodic-frequent pattern mining a computationally expensive process or impracticable in real-world applications.**

Example 2. Let '*xy*' be a frequent pattern in a very large transactional database appearing randomly at the timestamps, say 5, 9, 20, 23, 27, 50 and so on. The existing state-of-the-art algorithms measure the *periodicity* of this pattern by performing a complete search on its huge list of timestamps. In the next step, they determine '*xy*' as either periodic or aperiodic by comparing its *periodicity* against the user-specified *maxPrd*. In other words, current approaches determine the periodic interestingness of this pattern by performing a complete search on its huge list of timestamps. This approach of determining the periodic interestingness of every candidate pattern in the itemset lattice makes the periodic-frequent pattern mining a computationally expensive process.

Property 1 (Anti-monotonic property of periodic-frequent patterns)**.** *For a pattern X, if* $Sup(X) \geq minSup$ *and* $Per(X) \leq maxPer$*, then* $\forall Y \subset X$ *and* $Y \neq \emptyset$, $Sup(Y) \geq minSup$ *and* $Per(X) \leq maxPer$.

Reducing the computational cost of periodic-frequent pattern mining is a non-trivial and challenging task. The main reason is that we cannot sacrifice any information pertaining to periodic-frequent patterns in order to reduce the computational cost.

With this motivation, we propose an approach to reduce the computational cost of finding the periodic-frequent patterns. Our approach determines the periodic interestingness of a pattern by adopting greedy search on its ts-list. The usage of greedy search achieves two important tasks. First, reduces the need of complete search on the ts-lists of aperiodically occurring patterns by identifying them based on sub-optimal solution. Second, finds global optimal solution (i.e., *periodicity*) for every periodic-frequent pattern. As a result, our approach reduces the computational cost without missing any