British Journal of Mathematics & Computer Science

British Journal of Mathematics & Computer Science
18(3): 1-19, 2016, Article no.BJMCS.27888
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Mining Incrementally Closed Itemsets with a New Intermediate Structure

Thanh-Trung Nguyen[1*]

[1] *Department of Computer Science, University of Information Technology, Vietnam National University, HCM City, Vietnam.*

Author's contribution

The sole author designed, analyzed and interpreted and prepared the manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/27888
Editor(s):
(1) Doina Bein, Applied Research Laboratory, The Pennsylvania State University, USA.
Reviewers:
(1) Sherin Zafar, Jamia Hamdard University, India.
(2) Stephen Akandwanaho, University of KwaZulu-Natal,University, South Africa.
Complete Peer review History: http://www.sciencedomain.org/review-history/15927

Received: 23rd June 2016
Accepted: 8th August 2016
Published: 25th August 2016

Original Research Article

Abstract

The problem of closed frequent itemset discovery is a fundamental issue of data mining, having applications in numerous domains. Until now, the general technic for incremental mining is using an intermediate structure in order to update the structure whenever there is a variation in the data. As for incremental mining closed itemsets, the intermediate structure used is a concept lattice. The concept lattice promotes the efficiency of the search process, but it is costly to adjust the lattice when there is an addition or removal, as well as it is difficult in developing parallelization strategy. This article proposes incremental algorithms to search all closed itemsets with a new intermediate structure which is a linear list. To the best of our knowledge, this is the first algorithm for incremental mining closed itemsets using a linear list as an intermediate structure proposed so far. When comparing experimental results between using intermediate structure concept lattice and linear list initially show that the greater number of transactions and the number of closed itemsets obtained in the mining process, the more efficient the use of linear list promotes.

Keywords: Closed itemsets; concept lattice; data mining; incremental mining; mining methods and algorithms; new intermediate structure.

**Corresponding author: E-mail: nguyen_thanh_trung_key@yahoo.com.vn;*

1 Introduction

Frequent sets are sets of items, subsequences or substructures appearing in a dataset with the frequency which is greater or equal a user-defined threshold.

The formal definition of frequent set is as follows: Given $I = \{i_1, i_2, . . . , i_m\}$ is the set of distinct items, $O = \{o_1, o_2, . . . , o_n\}$ is the set of transactions on the transactional database. A mining context is a triple $D = (O, I, R)$, for $R \subseteq O \times I$ is a binary relation of transactions and items. Each $(o, i) \in R$ represents transaction $o \in O$ containing item $i \in I$.

A set-k α, consisting of k elements from I, is frequent if α appears in transactional database not less than $\theta|O|$ times, for θ is a *minimal support threshold* defined by users (then called *minsup*), and $|O|$ is the total number of transactions. The number of occurrences of α is called the support of α (*support*(α)).

We call L set of frequent item sets. Set M containing *maximal frequent item sets* in D is defined as follows: $M = \{C \in L \mid \nexists C' \in L, C \subset C'\}$

For $B \subseteq O$ and $C \subseteq I$:

$f(B): 2^O \to 2^I$

$f(B) = \{i \in I \mid \forall o \in B, (o, i) \in R\}$

$g(C): 2^I \to 2^O$

$g(C) = \{o \in O \mid \forall i \in C, (o, i) \in R\}$

$C \subseteq I$ is closed set if and only if $h(C) = C$, for $h = f \text{ o } g$

Closed set C is called frequent if the support of C in D is greater or equal *minsup*. Set FC containing *frequent closed item sets* in D is defined: $FC = \{C \subseteq I \mid C = h(C) \wedge support(C) \geq minsup\}$

Set MC containing *maximal frequent closed item sets* in D is defined: $MC = \{C \in FC \mid \nexists C' \in FC, C \subset C'\}$

Incremental mining is the process that the only updated data should be exploited in order to discover frequent sets. The main purpose of incremental mining is that because data add continueously to the initial transaction, hence the size of database becomes larger and mining the entire database will take more time for calculating, so it is better if the only updated data is mined. Thence, it supports the ability of execution faster than not incremental methods.

Formal definition of incremental mining is as follows:

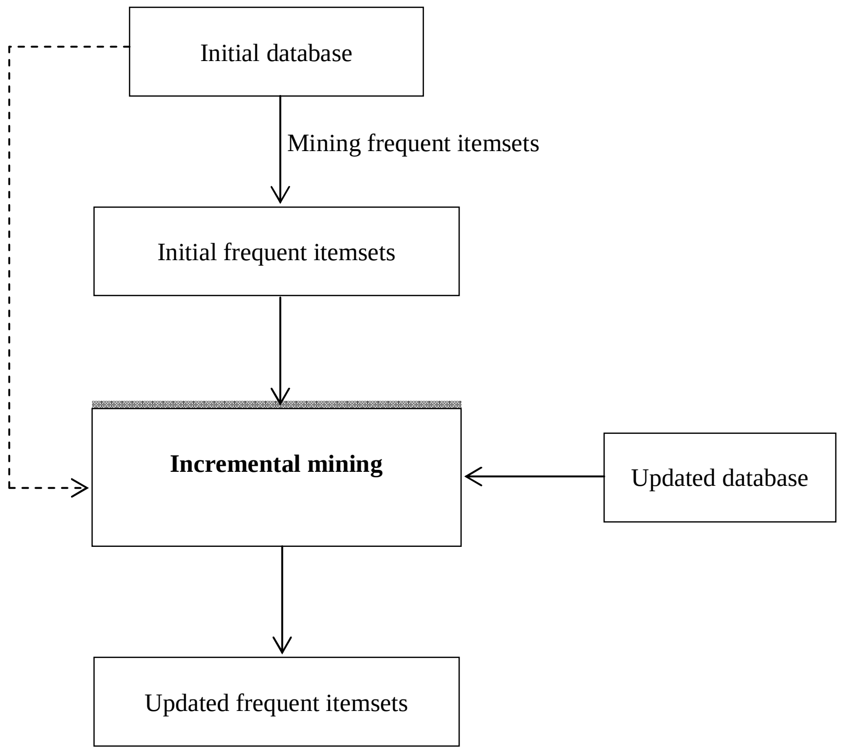
D: data mining context
A: algorithm for mining frequent item sets,
L: set of frequent sets
D, A (*minsup*) $\to L$
$\{t\}$: updated data

For not incremental algorithm:

$D^+ = D \cup \{t\}$
$D^- = D \setminus \{t\}$
$D^+, A\ (minsup) \to L^+$
$D^-, A\ (minsup) \to L^-$

For incremental algorithm (A^*):

$\{t\}, A^*, L \to L^+$
L: result of the previous period (frequent set)



The article proposes incremental algorithms to find all closed sets with a new intermediate structure.

According to [22], set of all frequent closed sets is sufficient to determine a reduced set of association rules. Then, it helps to solve another important problem: limiting the number of generated rules without data loss. So, frequent closed sets might directly create reduced set of association rules without the need of determining all frequent sets, hence reducing the cost of calculating algorithm. In addition, because of thousands of hiding association rules, simplifying the number of generated rules without data loss plays an important role with obtained results.

In 2003, the workshop Frequent Item set Mining Implementation on implementing algorithms of mining frequent sets was reported by Goethals and Zaki [8]. Mining closed sets provides a valuable and important alternation for the problem of mining frequent sets because it inherits the same strength of analysis but creating a set of much smaller result.

2 Overview

First, the article is going to provide an overview of panoramic perspective on incremental mining. Almost incremental mining algorithms are divided into two main categories: Apriori-based algorithms and tree-based algorithms.

Second, the article is going to focus on the problem of incremental mining (frequent) closed itemsets.

2.1 Incremental mining

2.1.1 Apriori-based algorithms

The algorithm FUP (Fast Update) [3] is the first algorithm proposing incremental mining association rules. It solves the issue of database with new added transactions, but cannot solve the case of deleting transactions.

Cheung et al. [4] proposed the algorithm FUP_2 which is the extension of the algorithm FUP. FUP updates association rules in a database when new transactions are added into the database. Meanwhile, FUP_2 updates the existing association rules when transactions added to and deleted from database. FUP_2 is similar to FUP in the case of adding new transactions, and is an additional algorithm for FUP in the case of deleting transactions.

The algorithms DELI and ULI face the issue of determing when updating current model. In order to decide when to update, [17] proposed the algorithm DELI (Difference Estimations for Large Itemsets), which applies the method of getting statistical samples to determine when the current model becomes obsolete. ULI (Update Large Itemsets) was proposed by [28]. ULI attempts to decrease I/O requirement to update the set of frequent itemsets by maintaining previous frequent itemsets and negative borders [17] and their supports.

In [1], the algorithm UWEP (Update With Early Pruning) was proposed, in which using updating technique with early-pruning. The advantage of the algorithm UWEP excels the FUP-based algorithms in that it prunes supersets of an initial frequent itemset in D as soon as it becomes infrequent in the updated D', instead of waiting until the iteration k^{th}.

The concept of negative borders [29] was used in [28] to improve the effect of FUP-based algorithms in incremental mining. Let L be a set of frequent itemsets, negative borders $B_d–(L)$ of L consists of minimal frequent itemsets $X \subseteq R$ but not in L, for R is the set of all items. In other words, negative borders consists of all sets which generated candidates with insufficient support.

Both algorithms MAAP (Maintaining Association rules with Apriori Property) [40] and PELICAN [34] are similar to the algorithm FUP_2, yet their main objective is to maintain the maximal frequent itemsets when the database is updated. These algorithms do not consider non-maximal frequent itemsets, so they do not need to calculate the supports of non-maximal frequent itemsets. The difference between the two algorithms is that MAAP calculates maximal frequent itemsets by relying on Apriori while PELICAN bases on the vertical data format and decomposing lattice.

Lee et al. [16] proposed the approach SWF (Sliding-Window Filtering). SWF divides database into many partitions, and applies a filtering threshold on each partition to create candidate sets. [35] described the algorithm ZigZag, using *tidlist* (list of transaction id) and calculating maximal frequent itemsets in the updated database to avoid generating many unnecessary candidates.

2.1.2 Tree-based algorithms

In [6], DB-tree and PotFp-tree were proposed for incremental mining. The algorithm DB-tree (*Database tree*) stores all items in a FP-tree instead of only 1-element frequent itemsets in database. Additionally, building a DB-tree is exactly as the same way as FP-tree. Hence, DB-tree might be seen as a FP-tree with the minimal threshold = 0. Another algorithm proposed in [6] is PotFp-tree (*Potential Frequent Pattern tree*), which only stores a few potential frequent items beside1-element frequent itemsets. A tolerance parameter t is used to decide whether an item is frequent potentially or not.

The algorithm AFPIM (*Adjusting FP-tree for Incremental Mining*) [13] updates FP-tree built previously by only scanning the increment of database. This increment database contains new transactions affecting the frequence of items. When items are ordered by descending frequency based on the initial dataset, AFPIM re-arranges items in the tree according to new value of frequency based on the increment dataset, using bubble-sort sorting method by recursively swapping adjacent items.

The tree-based algorithms EFPIM (*Extending FP-tree for Incremental Mining*) [19] and FUFP-tree (*Fast Updated Frequent Pattern tree*) [11] as well as AFPIM, conduct incremental mining by using a compressed data structure, mainly adjusting the structure FP-tree. These approaches still require two times of scanning database for the initial part (in order to build the FP-tree structure) and the increment part (in order to update the tree structure).

CATS-tree (*Compressed and Arranged Transaction Sequence tree*) [5] and DB-tree are the same because both store all items without caring whether they are frequent or not. This feature allows CATS-tree to avoid re-scanning database when updates occur. However, the way of building CATS-tree is different from FP-tree and DB-tree. In more details, FP-tree is built based on the order of global supports of all frequent items while CATS-tree is built based on the order of local supports of items in their path.

In [18], a tree structure called CanTree (*Canonical Tree*) proposed in order to obtain content of transactional database and arrange tree nodes in a canonical order.

CP-tree (*Compact Pattern tree*) was proposed in [27]. This algorithm also builds prefix tree by conducting a unique scanning on database. [36] proposed modified CP-tree, constructing a tree for entire database with items arranged on the same order as their occurrence on transactions.

Lin et al. [20] proposed PreLarge-tree for incremental mining association rules based on concept of pre-large itemsets. A pre-large itemset is not actually large, but maybe large with a high probability in the future. A pre-large itemset is a itemset having frequency greater than lower support threshold defined by users and less than upper support threshold defined by users.

SPO-tree (*Single Pass Ordered tree*) [14] orders items of a transaction by descending frequency. It re-constructs periodically the tree based on a parameter called *Edit Distance*. The tree is re-organized once Edit Distance of items in the order exceeding the pre-defined threshold.

The algorithm BIT (*Batch Incremental Tree*) [30] was proposed for batch processing incrementally increasing database in order to construct a canonical ordered tree (CanTree). The algorithm BIT merges two FP-trees of two small adjacent periods to obtain a FP-tree which equivalent to FP-tree obtained when entire database is processed at the same time from the beginning of the first period to the end of the second period. In [31], the authors proposed applying the same principle used in the algorithm BIT to build the equivalent FP-tree but with the algorithm FP-Growth. That is they uses batch incremental mining to build FP-tree by applying algorithm FP-Growth, and named BIT_FPGrowth.

Incremental mining based on the intermediate structure of FP-tree shows a weakness in the implementation process because the FP-tree structure depends on the global property of supports of items in the database. Therefore, when the data is updated, it will create influence on the FP-tree structure, specifically in situations: new items are frequent, or old items become less frequent than new items. It is particularly serious with the situation of old items become unfrequent.

To solve this problem, there are solutions such as updating periodically the tree structures or basing on the indicators to determine the time for updating. Especially, there is the solution of using the canonical order to avoid having to depend on the global order of supports of items. However, there are still problems as described in detail in each of the study above.

A new research direction is to use an intermediate structure of concept lattice. The next section presents an overview of mining incrementally closed sets with the intermediate structure of concept lattice, and the techniques of not incremental mining closed sets.

2.2 Incremental mining (frequent) closed itemsets

The approaches for mining (frequent) closed itemsets are now divided into two groups: incremental mining and not incremental mining.

Mining frequent closed itemsets was first proposed by [22], with an algorithm based-on-Apriori, called A-Close.

In series, the algorithms for mining closed itemsets include CLOSET [23], CHARM [39], CLOSET+ [38], FPClose [9] and AFOPT [21].

The main challenge in mining frequent closed itemsets is to check whether an itemset is closed or not. There are two strategies to approach this problem: (1) keep track of TID list of an itemset and index the itemset by hashing its TID values. This method is used by CHARM whose task of maintaining a TID list called a diffset, and (2) maintain itemsets discovered in a tree similar to FP-tree. This method is exploited by CLOSET +, AFOPT and FPClose.

The methods described above are the mining approaches which is not incremental. In the recent period, researchers are focusing on the tendency to use concept lattices to serve the purpose of incremental mining. The concept lattice, widely used in mathematics [7], is the hierarchical structure between concepts. Each concept consists of three components: a set of objects, a set of attributes and a relation between these sets. Correspondingly, each concept can include a closed set, a transaction set and the relationship between these two sets [26]. Methods for maintaining a concept lattice can be divided into 2 groups: (1) *direct-update*, new transactions are added separately to the lattice and (2) *merge-lattices*, constructing the lattice from new transactions added and merging this with the original lattice.

The algorithms proposed in [12,33,10,25,24,37,15] belong to the direct-update group.

The methods of [32,2] belong to the merge-lattices group.

Until now, the general technic for incremental mining is using an intermediate structure in order to update the structure whenever there is a variation in the data. As for incremental mining closed itemsets, the intermediate structure used is a concept lattice. The concept lattice promotes the efficiency of the search process, but it is costly to adjust the lattice when there is an addition or removal, as well as it is difficult in developing parallelization strategy. This is evident when the studies of incremental mining closed itemsets by merging lattices have not been significantly developed since the 2007.

This article proposes incremental algorithms to search all closed itemsets with a new intermediate structure which is a linear list. Experimental comparing results between using intermediate structure concept lattice and linear list initially show: The greater number of transactions as well as the number of closed itemsets is obtained in the mining process, the more efficient the use of linear list promotes.

3 Proposal Work

3.1 Constructing the intermediate structure

Let $B = \{0, 1\}$, B^m is the space of m-tuple bit chains, whose elements are $s = s_1 s_2 \ldots s_m$, $s_i \in B$, $i = 1, \ldots, m$.

Definition 1: Given two bit-chains with the same length: $a = a_1a_2\ldots a_m$, $b = b_1b_2\ldots b_m$. a is said to *cover b* or *b is covered by a* – denoted $a \hookrightarrow b$ – if $pos(b) \subseteq pos(a)$ for $pos(s) = \{i \mid s_i = 1\}$. To be negative, the operator ! is used, particularly $a\ !\hookrightarrow b$.

Definition 2:

+ Let u be a bit-chain, k is a natural number, we call $[u; k]$ *a sample*.

+ Let S be set of m-tuple bit-chains (bit-chain with the length of m bits), u is a m-tuple bit-chain. If there are at least k bit-chains in S covering u, we say: u is a *form* of S with the frequency of k; and $[u; k]$ is a sample of S – denoted $[u; k]_{\rightarrow S}$.

Example 1: $S = \{1110, 0111, 0110, 0010, 0101\}$ and $u = 0110$. We say u is a form with the frequency of 2 in S, hence $[0110; 2]_{\rightarrow S}$.

+ A sample $[u; k]$ of S called *maximal sample* – denoted $[u; k]_{\max\to S}$ – if and only if it does not exist k' that $[u; k']_{\to S}$ and $k' > k$. For the above example, $[0110; 3]_{\max\to S}$

Definition 3 (operations and binary relation):

+ Two m-tuple bit-chains a and b are called equal– denoted $a = b$ – if and only if $a_i = b_i$ $\forall i \in \{1, \dots, m\}$, vice versa $a \neq b$.

+ Given two samples $[u_1; p_1]$ and $[u_2; p_2]$. $[u_1; p_1]$ is said to *be contained* in $[u_2; p_2]$ – denoted $[u_1; p_1] \subseteq [u_2; p_2]$ – if and only if $u_1 = u_2$ and $p_1 \leq p_2$, vice versa $[u_1; p_1] \not\subset [u_2; p_2]$.

+ Given two m-tuple bit-chains a and b. A m-tuple bit-chain z is called *minimal sequence* of a and b – denoted $z = a \wedge b$ – if and only if $z_k = \min(a_k, b_k)$ $\forall k \in \{1, \dots, m\}$.

+ *Minimal sample* of two samples $[u_1; p_1]$ and $[u_2; p_2]$ is a sample $[u'; p']$ – denoted $[u'; p'] = [u_1; p_1] \circ [u_2; p_2]$ – for $u' = u_1 \wedge u_2$ and $p' = p_1 + p_2$.

Definition 4: P is a re*presentative set* of S when $P = \{[u; p]_{\max\to S} \mid \nexists [v; q]_{\max\to S} \neq [u; p] : (v \hookrightarrow u \text{ and } q \geq p)\}$. Each of elements of P is called *a representative sample* of S.

The rationale for constructing the set P:

Representative set P is the set of closed sets of S (according to the definition from [22]). Once *minsup* is established, we can obtain closed frequent sets of S.

Theoretical bases for constructing set P are as follows:

The definition of closed set from [22]:

A context of mining dataset is a triple $D = (O, I, R)$. O and I are sets of finite transactions and items. $R \subseteq O \times I$ is a binary relation of transactions and items. Each pair $(o, i) \in R$ shows that transaction $o \in O$ containing item $i \in I$,

For $B \subseteq O$ and $C \subseteq I$:

$f(B)$: $2^O \to 2^I$

$f(B) = \{i \in I \mid \forall o \in B, (o, i) \in R\}$

$g(C)$: $2^I \to 2^O$

$g(C) = \{o \in O \mid \forall i \in C, (o, i) \in R\}$

$C \subseteq I$ is closed set if and only if $h(C) = C$, for $h = f \circ g$

With C a closed set, we have two following affirmations:

* In case of adding new items to C, becoming C^+ ($C \subset C^+$) and if C^+ is a closed set, $g(C^+)$ has to have the strictly smaller number of elements than $g(C)$ ($g(C^+) \subset g(C)$). Indeed:

- If $g(C^+) = g(C)$, for $f(g(C^+)) = C^+$ hence $f(g(C)) = C^+$, conflicting with the definition of that C is a closed set.
- If $g(C^+) \supset g(C)$, it conflicts with the definition of $g(C) = \{o \in O \mid \forall i \in C, (o, i) \in R\}$ (find **all** $o \in O$ so that each o contains all $i \in C$, so why there are $o' \in g(C^+)$ containing all $i \in C$).

* In case of withdrawing items out of C, becoming C^- ($C^- \subset C$) and if C^- is a closed set, $g(C^-)$ has to have the strictly greater number of elements than $g(C)$ ($g(C^-) \supset g(C)$). Indeed:

- If $g(C^-) = g(C)$, $f(g(C^-)) = f(g(C)) = C^-$, conflicting with the definition of that C is a closed set.
- If $g(C^-) \subset g(C)$, it conflicts with the definition of $g(C^-) = \{o \in O \mid \forall i \in C^-, (o, i) \in R\}$ (find **all** $o \in O$ so that each o contains all $i \in C^-$, so why there are $o' \in g(C)$ containing all $i \in C^-$).

Remark 1: *Without loss of generality, an arbitrary closed set of D either differents from (does not cover or is not covered by) other closed sets or if strictly contained in a closed set α, its frequency has to be greater than the frequency of α.*

Basing on this basis to construct the set $P = \{[u; p]_{\max\to S} \mid \nexists[v; q]_{\max\to S} \neq [u; p] : (v \hookrightarrow u \text{ and } q \geq p)\}$.

In more details, the set P will have:

Non-existing cases	**Existing cases**
$v \hookrightarrow u$ and $q_v \geq p_u$	$v\ !\hookrightarrow u$ or $q_v < p_u$
• $v = u$ and $q_v = p_u$ (not actual) • $v = u$ and $q_v > p_u$ (not actual) • ***v* strictly covers *u* and $q_v = p_u$** • ***v* strictly covers *u* and $q_v > p_u$**	• $v\ !\hookrightarrow u$ and $q_v \geq p_u$ **- $v \neq u$ ($u\ !\hookrightarrow v$) and $q_v \geq p_u$** **- *u* strictly covers *v* and $q_v > p_u$** (not =) • $v\ !\hookrightarrow u$ and $q_v < p_u$ **- $v \neq u$ ($u\ !\hookrightarrow v$) and $q_v < p_u$** - *u* strictly covers *v* and $q_v < p_u$ (discard, because of returning the non-existing cases, left column) • $v \hookrightarrow u$ and $q_v < p_u$ **- *v* strictly covers *u* and $q_v < p_u$** - $v = u$ and $q_v < p_u$ (not actual)

The property of the set P:

We are able to show a set of transactions as the set S of bit-chains. For a bit-chain in S, the i^{th} bit is established as 1 when the i^{th} item is purchased and vice versa.

When an arbitrary minimal support threshold *minsup* is established, the representative set P will give all *closed frequent sets* and *maximal frequent sets* of S.

Indeed:

Firstly, we repeat the definitions of closed frequent set and maximal frequent set.

A set α is a *closed frequent set* on dataset D if α is frequent on D and there is not any strict superset β of α and *support*(α) = *support*(β) on D.

A set α is a *maximal frequent set* on D if α is frequent, and there is not any strict superset β for $\alpha \subset \beta$ and β is frequent on D.

Therefore, a maximal frequent set is a special closed frequent set. More specific, according to the definition, a maximal frequent set is definitely a closed frequent set at a support equaling its frequency and it is the closed frequent set which is not able to be contained by any other closed frequent sets.

With the definition $P = \{[u; p]_{\max\to S} \mid \nexists[v; q]_{\max\to S} \neq [u; p] : (v \hookrightarrow u \text{ and } q \geq p)\}$, considering an arbitrary element $[u; p] \in$ P, once $p \geq minsup$ (that is, u shows a frequent set α with support p). Then, according to the definition of set P, there is not any bit-chain v (showing a set β) which is able to cover u with the frequency of $q = p$. That means, at the support value p of the frequent set α, it does not exist a strictly larger set β having the same support of p. Since, $[u; p]$ is a closed frequent set.

Hence, the set P contains all closed frequent sets of S, inferring P also contains all maximal frequent sets of S.

We may easily calculate closed frequents set based on P.

So, the major issue is concentrated on re-constructing the representative set P whenever S is modified (adding, deleting elements).

3.2 The incremental algorithm for adding a new transaction

Let S be a set of n m-tuple bit-chains with the representative set P. In this part, we will consider the algorithm for rebuilding set P when a new bit-chain is added to S.

The algorithm NewRepresentative:

```
Input: $P$ is the representative of $S$, $z$ is a bit-chain added to $S$
Output: The new representative set $P$ of $S \cup \{z\}$
    For each $x \in P$:
      • Using the operation o in order to find the smaller closed set of $x$ and its frequency is
        greater than $x$ 1 unit. (relying on Remark 1)
      • Considering $z$ as a new closed set of $P$. (Now, $P$ has two group of element: previous
        elements and new elements created by the operation o)
      • Verifying to discard invalid elements of $P$ (in order to ensure the property of $P$):
            -  Discarding previous elements are contained by new elements
            -  Discarding new elements contain mutually
    Output the set $P$
```

3.3 The incremental algorithm for deleting a transaction

Definition 5:

Let S be a set of bit-chains and P be the representative set of S. P is obtained by applying the algorithm *NewRepresentative* to S. Let $[p; k] \in P$, and $s_1, s_2, \dots, s_r \in S$ be r ($r \leq k$) bit-chains taking part in forming p, denoted p_crd: $s_1, s_2, \dots, s_r$, vice versa, denoted: p_crd: $!s_1, !s_2, \dots, !s_r$.

***Example 2*:** In *Example 1*, we have bit-chains 1110 and 0111 are 2 of 3 bit-chains participating in forming [0110; 3]. Let s_1 = 1110, s_2 = 0111 and p = 0110, we have: p_crd = s_1, s_2. Let s_3 = 0101 not participating in forming [0110; 4], so: p_crd = $!s_3$.

The following is the algorithm to find the new representative set of S when a bit-chain is deleted from S.

The algorithm NewRepresentative_Delete:

```
Input: $P$ is the representative set of $S$, $z$ is the deleted transaction
Output: the new representative set $P$ of $S \setminus \{z\}$
    For each $x \in P$
        If z $\hookleftarrow$ the form of $x$ (i.e. $x$.form_crd = $z$)
            Decreasing the frequency of $x$ 1 unit
            If the frequency of $x = 0$ then removing $x$ from $P$
            Verifying to discard $x$ if $x$ is contained by an element in $P$ (in order to ensure the
            property of $P$)
        End if
    End for
    Output the set $P$
```

4 Experiment for Verifying Results

The algorithm proposed by [12] has been one of the first approaches to build the lattice of closed sets. This algorithm builds a lattice containing all closed sets of the original dataset and allows incremental mining by the direct-update method when a transaction is added. The obtained results (including the quantity, meaning and purpose) of this algorithm are completely accurate as the results of the algorithm *NewRepresentative*. Therefore, the comparative experiment is conducted with the algorithm of [12].

*** Infrastructure:** one computer with the configuration as follows:

- CPU: Intel(R) Core(TM) i3-2100 (4 CPUs), ~3.1GHz
- RAM: 8192MB
- Operation Systems: Windows 7 Ultimate 64-bit (6.1, Build 7601) Service Pack 1
- Programming language: C#.NET

*** Experimental datasets:** got from http://fimi.ua.ac.be/data/

*** *T10I4D100K*:**

Number of transactions on the database:	100,000
Maximal number of items on each transaction:	29
Maximal items on the dataset:	1,000

Table 1. Comparative figures between 2 algorithms for *T10I4D100K*

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	**Total time (second)**	**No. closed sets**	**The time for adding 1 transaction (second)**	**Total time (second)**	**No. closed sets**
1,000	0.003	1.65	7,311	0.06	19.26	7,311
2,000	0.02	9.66	18,213	0.64	147.79	18,213
3,000	0.04	35.58	31,022	0.81	536.50	31,022
4,000	0.06	85.91	44,528	0.47	1,288.87	44,528
5,000	0.19	166.26	59,279	2.31	2,628.95	59,279
6,000	0.11	271.86	74,006	1.40	4,622.61	74,006
7,000	0.20	418.83	89,830	3.99	7,546.51	89,830
8,000	0.25	604.89	105,544	6.24	11,362.58	105,544
9,000	0.09	815.71	121,166	1.47	16,307.64	121,166
10,000	0.33	1,089.89	139,491	7.71	23,455.05	139,491
11,000	0.52	1,415.70	158,138			
12,000	0.33	1,806.82	176,766			
13,000	0.30	2,253.67	195,557			
14,000	0.69	2,769.95	215,066			
15,000	0.75	3,362.96	235,747			
16,000	2.09	4,021.19	255,604			
17,000	0.81	4,667.43	275,385			
18,000	1.26	5,417.64	295,607			
19,000	0.45	6,202.65	315,910			
20,000	0.25	7,083.47	336,109			
21,000	0.20	8,043.19	355,853			
22,000	2.32	9,088.06	376,549			
23,000	2.29	10,251.07	396,845			
24,000	1.20	11,507.05	417,756			
25,000	0.92	12,926.93	440,785			
26,000	0.55	14,404.83	461,495			

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	Total time (second)	No. closed sets	The time for adding 1 transaction (second)	Total time (second)	No. closed sets
27,000	0.87	15,897.83	483,930			
28,000	0.19	17,470.77	506,176			
29,000	1.30	19,169.08	529,346			
30,000	2.31	20,925.02	552,617			
31,000	0.66	22,815.77	573,595			
32,000	0.36	24,446.15	595,934			
33,000	2.68	26,050.77	619,655			
34,000	0.67	27,727.44	643,210			
35,000	2.64	29,567.60	667,597			
36,000	2.47	31,484.93	691,155			
37,000	1.20	33,543.99	713,641			
38,000	0.98	35,563.43	736,338			
39,000	2.73	37,631.26	760,625			
40,000	1.53	39,772.44	783,805			

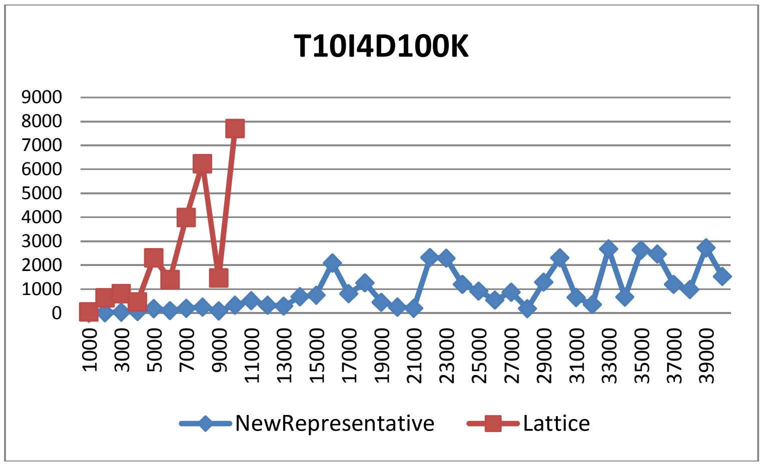


Fig. 1. Comparative chart between 2 algorithms for *T10I4D100K*

With the experimental dataset *T10I4D100K*, the algorithm of [12] (*Lattice* for short) has the phenomenon of memory overflow when the number of transactions is about 10,000. Meanwhile, the algorithm *NewRepresentative* overflows when the number of transactions is about 40,000.

The following part is a comparison chart of the two algorithms on implementing time at each landmark of the number of transactions. The vertical column represents the time in millisecond, and the horizontal bar represents landmarks of the number of transactions.

With the following experimental dataset (*retail*, *mushroom*, *connect*, *pumsb_star*, *pumsb*), results of comparison tables and graphs are presented exactly as the presentation of the dataset *T10I4D100K*.

*** *retail*:**

Number of transactions on database:	88,162
Maximal number of items on each transaction:	76
Maximal items on dataset:	16,469

Table 2. Comparative figures between 2 algorithms for *retail*

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	**Total time (second)**	**No. closed sets**	**The time for adding 1 transaction (second)**	**Total time (second)**	**No. closed sets**
1,000	0.001	1.27	4,410	0.006	1.01	4,410
2,000	0.01	7.72	11,907	0.03	50.50	11,907
3,000	0.01	31.96	25,848	0.22	333.47	25,848
4,000	0.02	76.16	40,364	1.18	1,162.11	40,364
5,000	0.07	136.55	51,877	2.58	2,254.10	51,877
6,000	0.02	238.31	69,172	0.12	4,349.16	69,172
7,000	0.08	377.10	86,722	2.62	7,804.74	86,722
8,000	0.08	513.15	97,736	1.79	10,393.21	97,736
9,000	0.26	684.84	112,927	7.22	14,312.75	112,927
10,000	0.17	929.07	132,472	0.54	21,685.91	132,472
11,000	0.19	1,176.73	145,805	2.01	28,084.02	145,805
12,000	0.37	1,433.72	157,027	9.00	33,623.61	157,027
13,000	0.12	1,765.60	171,748	0.83	42,301.62	171,748
14,000	0.13	2,250.50	193,093	10.05	69,322.26	193,093
15,000	0.23	2,651.00	206,187	5.18	84,387.09	206,187
16,000	0.42	3,064.94	218,447			
17,000	0.17	3,557.91	238,302			
18,000	1.64	4,182.06	267,375			
19,000	0.51	4,652.33	282,995			
20,000	0.09	5,175.34	298,866			
21,000	1.07	5,902.65	326,976			
22,000	1.12	6,728.45	355,411			
23,000	0.34	7,289.17	369,846			
24,000	0.58	8,003.51	389,871			
25,000	0.19	8,975.41	418,421			
26,000	0.29	9,797.55	432,478			
27,000	0.63	10,656.31	447,936			
28,000	0.27	11,886.16	476,058			
29,000	0.64	12,984.84	495,801			
30,000	0.62	13,906.40	510,335			
31,000	0.55	14,918.17	530,243			
32,000	0.59	16,277.79	562,342			
33,000	0.41	17,294.59	580,406			
34,000	0.99	18,362.52	597,965			
35,000	0.45	19,516.55	620,735			
36,000	0.98	21,312.59	657,313			
37,000	0.28	22,867.05	677,236			
38,000	0.36	24,427.39	697,467			
39,000	2.22	26,054.38	726,168			
40,000	0.47	27,745.88	753,244			
41,000	0.72	28,954.68	766,170			
42,000	0.81	30,362.63	783,859			
43,000	4.44	32,254.40	811,294			
44,000	0.24	34,402.17	841,938			
45,000	0.78	36,108.94	858,093			
46,000	11.28	37,986.32	879,523			
47,000	1.81	40,052.93	908,826			
48,000	0.53	42,130.18	940,774			
49,000	0.47	43,798.43	957,926			
50,000	0.67	45,403.28	974,458			

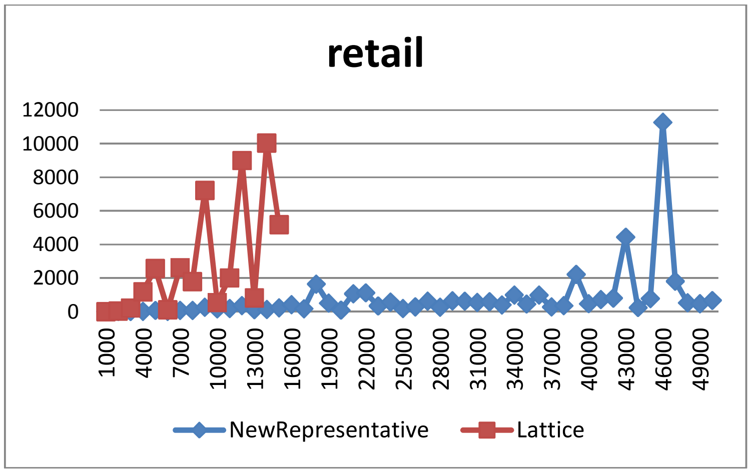


Fig. 2. Comparative chart between 2 algorithms for *retail*

*** *mushroom*:**

Number of transactions on database:	8,124
Maximal number of items on each transaction:	23
Maximal items on dataset:	119

Table 3. Comparative figures between 2 algorithms for *mushroom*

No. transactions	**NewRepresentative**			**Lattice**		
	The time for adding 1 transaction (second)	**Total time (second)**	**No. closed sets**	**The time for adding 1 transaction (second)**	**Total time (second)**	**No. closed sets**
1,000	0.58	364.04	32,513	154.97	47,893.60	32,513
2,000	2.40	2,036.38	58,982			
3,000	3.26	5,019.46	80,901			
4,000	1.89	8,290.29	104,104			
5,000	4.03	11,954.85	136,401			
6,000	3.99	19,552.01	156,573			
7,000	8.33	30,003.14	214,950			
8,000	11.60	41,696.58	237,874			
8,124	12.10	43,140.48	238,709			

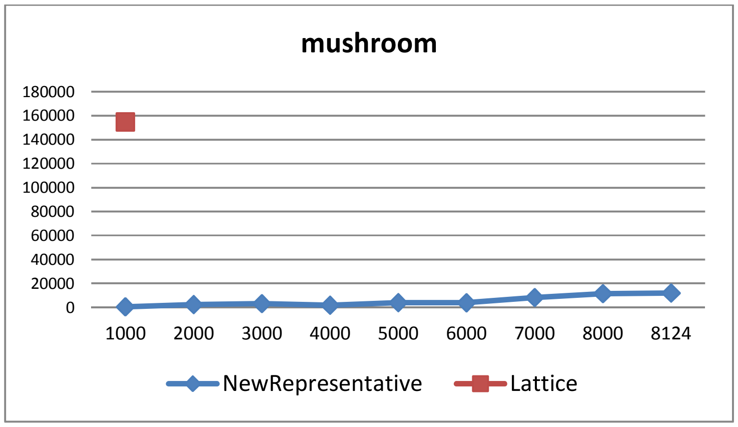


Fig. 3. Comparative chart between 2 algorithms for *mushroom*

*** *connect*:**

Number of transaction on database:	67,557
Maximal number of items on each transaction:	43
Maximal items on dataset:	129

Table 4. Comparative figures between 2 algorithms for *connect*

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	Total time (second)	No. closed sets	The time for adding 1 transaction (second)	Total time (second)	No. closed sets
100	0.54	14.44	13,406	22.39	533.41	13,406
200	10.81	485.31	63,360	1,217.30	43,131.84	63,360
300	62.52	3,738.96	149,393			
400	62.71	12,867.77	232,526			
500	370.23	35,284.15	445,676			

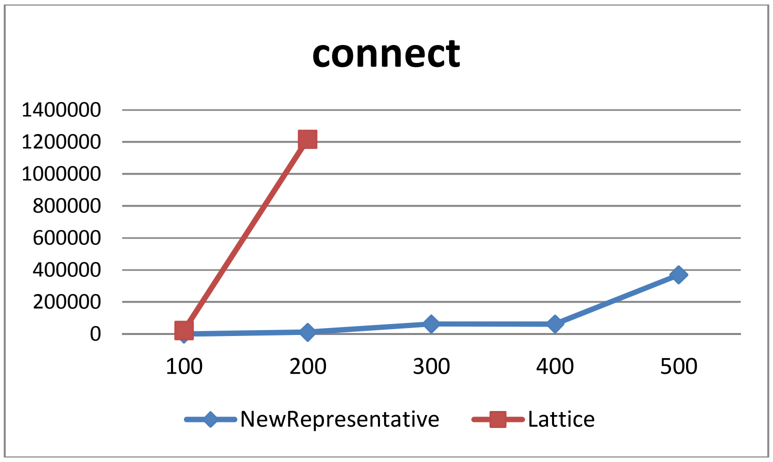


Fig. 4. Comparative chart between 2 algorithms for *connect*

*** *pumsb_star*:**

Number of transaction on database:	49,046
Maximal number of items on each transaction:	63
Maximal items on dataset:	7,116

Table 5. Comparative figures between 2 algorithms for *pumsb_star*

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	Total time (second)	No. closed sets	The time for adding 1 transaction (second)	Total time (second)	No. closed sets
10	0	0.008	189	0.002	0.16	189
20	0.01	0.15	1,301	0.06	0.35	1,301
30	0.02	0.37	4,465	1.22	5.15	4,465
40	0.07	0.86	8,974	5.12	35.58	8,974
50	0.31	2.57	17,425	15.79	170.86	17,425
60	0.35	6.31	25,433	70.20	500.91	25,433

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	**Total time (second)**	**No. closed sets**	**The time for adding 1 transaction (second)**	**Total time (second)**	**No. closed sets**
70	2.74	17.63	38,305	100.68	1,605.28	38,305
80	0.68	44.46	48,875	58.84	3,055.21	48,875
90	4.02	78.72	62,212	510.07	5,631.91	62,212
100	10.72	139.11	84,907	395.72	11,918.65	84,907
110	8.97	249.70	101,179			
120	36.22	408.70	130,144			
130	26.41	583.24	145,829			
140	4.59	768.99	168,617			
150	5.96	1,278.00	205,421			
160	141.07	1,856.49	225,118			
170	156.66	2,742.11	246,286			
180	76.11	3,403.82	277,220			
190	227.07	4,408.26	309,147			
200	160.48	5,423.01	354,489			

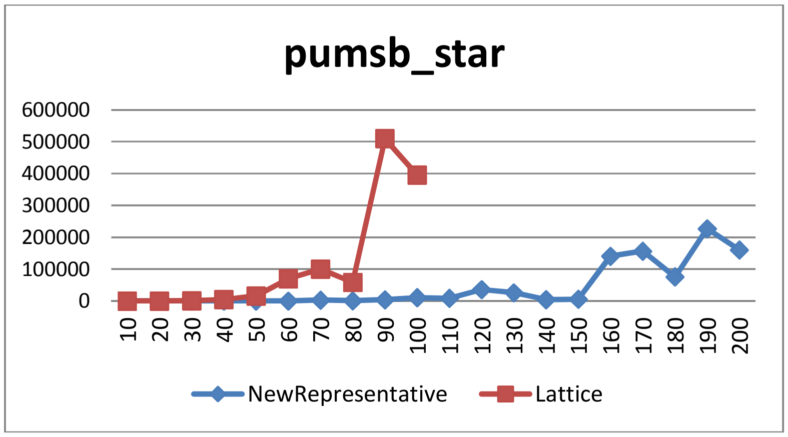


Fig. 5. Comparative chart between 2 algorithms for *pumsb_star*

*** *pumsb*:**

Number of transaction on database:	49,046
Maximal number of items on each transaction:	74
Maximal number of items on dataset:	7,116

Table 6. Comparative figures between 2 algorithms for *pumsb*

No. transactions	NewRepresentative			Lattice		
	The time for adding 1 transaction (second)	**Total time (second)**	**No. closed sets**	**The time for adding 1 transaction (second)**	**Total time (second)**	**No. closed sets**
10	0.006	0.02	240	0.004	0.13	240
20	0.05	0.21	1,873	0.14	0.66	1,873
30	0.28	1.44	10,108	14.37	37.32	10,108
40	2.33	8.44	26,544	86.66	495.26	26,544
50	4.17	102.08	99,575	1,725.10	6,261.31	99,575
60	342.14	977.71	252,113			
70	1,237.60	7,575.90	406,906			
80	840.42	18,507.95	864,979			

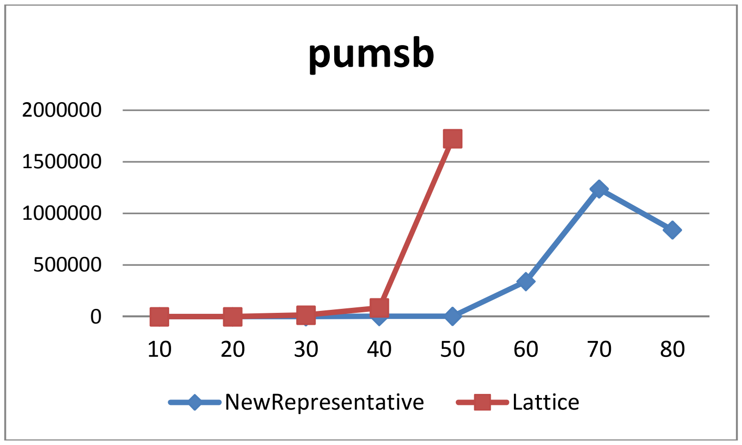


Fig. 6. Comparative chart between 2 algorithms for *pumsb*

5 Conclusion

This article proposes incremental algorithms to search all closed itemsets with a new intermediate structure which is a linear list. To the best of our knowledge, this is the first algorithm for incremental mining closed itemsets using a linear list as an intermediate structure proposed so far. Experimental comparing results between using intermediate structure concept lattice and linear list initially show: The greater number of transactions as well as the number of closed itemsets is obtained in the mining process, the more efficient the use of linear list promotes.

In the near future, we will research on parallelization algorithms to reduce implementing time and improve the performance.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Ayan NF, Tansel AU, Arkun ME. An efficient algorithm to update large itemsets with early pruning. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA. 1999;287-291.

[2] Ceglar A, Roddick JF. Incremental association mining using a closed-set lattice. Journal of Research and Practice in Information Technology. 2007;39(1):35-45.

[3] Cheung D, Han J, Ng V, Wong CY. Large databases: An incremental updating technique. Proceedings of the 12th International Conference on Data Engineering. 1996;106-114.

[4] Cheung DW, Lee SD, Kao B. A general incremental technique for updating discovered association rules. Proceedings of the Fifth International Conference On Database Systems for Advanced Applications, Melbourne, Australia. 1997;185-194.

[5] Cheung W, Zaïane OR. Incremental mining of frequent patterns without candidate generation or support constraint. Proceedings of the 7th International Database Engineering and Application Symposium. 2003;111-116.

[6] Ezeife CI, Su Y. Mining incremental association rules with generalized FP-tree. Proceedings of the 15th Canadian Conference on Artificial Intelligence; 2002.

[7] Ganter B, Wille R. Formal concept analysis: Mathematical foundations. Springer; 1999.

[8] Goethals B, Zaki M. An introduction to workshop on frequent itemset mining implementations. Proceeding of the ICDM'03 International Workshop on Frequent Itemset Mining Implementations (FIMI'03), Melbourne, FL. 2003;1-13.

[9] Grahne G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets. Proceeding of the ICDM'03 International Workshop on Frequent Itemset Mining Implementations (FIMI'03), Melbourne, FL. 2003;123-132.

[10] Gupta A, Bhatnagar V, Kumar N. Mining closed itemsets in data stream using formal concept analysis. Data Warehousing and Knowledge Discovery, LNCS. 2010;6263:285-296.

[11] Hong TP, Lin CW, Wu YL. Incrementally fast updated frequent pattern trees. Expert Systems with Applications. 2008;34(4):2424-2435.

[12] Hu K, Lu Y, Shi C. Incremental discovering association rules: A concept lattice approach. Proceeding of PAKDD99, Beijing. 1999;109-113.

[13] Koh JL, Shieh SF. An efficient approach for maintaining association rules based on adjusting FP-tree structures. Database Systems for Advanced Applications, Lecture Notes in Computer Science. 2004; 2973:417-424.

[14] Koh YS, Dobbie G. SPO-tree: Efficient single pass ordered incremental pattern mining. Springer, Berlin, Heidelberg, LNCS. 2011;6862:265-276.
DOI: 10.1007/978-3-642-23544-3-20

[15] La PT, Le B, Vo B. Incrementally building frequent closed itemset lattice. Expert Systems with Applications. 2014;41(6):2703-2712.

[16] Lee CH, Lin CR, Chen MS. Sliding-window filtering: An efficient algorithm for incremental mining. Proc. 10th Intl Conf. on Information and Knowledge Management, Atlanta, GA. 2001;263-270.

[17] Lee S, Cheung D. Maintenance of discovered association rules: When to update? Research Issues on Data Mining and Knowledge Discovery; 1997.

[18] Leung CKS, Khan QI, Hoque T. CanTree: A tree structure for efficient incremental mining of frequent patterns. Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05); 2005.

[19] Li X, Deng X, Tang S. A fast algorithm for maintenance of association rules in incremental databases. ADMA. 2006;56-63.

[20] Lin CW, Hong TP, Lu WH. Using the structure of prelarge trees to incrementally mine frequent itemset. New Gener Comput. 2010;28(1):5-20.
DOI: 10.1007/s00354-008-0072-6

[21] Liu G, Lu H, Lou W, Yu JX. On computing, storing and querying frequent patterns. Proceeding of the 2003 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03), Washington, DC. 2003;607-612.

[22] Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. Proceeding of the 7th International Conference on Database Theory (ICDT'99), Jerusalem, Israel. 1999;398-416.

[23] Pei J, Han J, Mao R. CLOSET: An efficient algorithm for mining frequent closed itemsets. Proceeding of the 2000 ACM-SIGMOD International Workshop Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX. 2000;11-20.

[24] Rouane-Hacene M, Huchard M, Napoli A, Valtchev P. Relational concept analysis: Mining concept lattices from multi-relational data. Annals of Mathematics and Artificial Intelligence. 2013;67(1):81-108.

[25] Szathmary L, Valtchev P, Napoli A, Godin R, Boc A, Makarenkov V. Fast mining of iceberg lattices: A modular approach using generators. In CLA, CEUR Workshop Proceedings. 2011;959:191-206.

[26] Szathmary L, Valtchev P, Napoli A, Godin R, Boc A, Makarenkov V. A fast compound algorithm for mining generators, closed itemsets and computing links between equivalence classes. Annals of Mathematics and Artificial Intelligence; 2013.

[27] Tanbeer SK, Ahmed CF, Jeong BS. Efficient single-pass frequent pattern mining using a prefix-tree. Elsevier International Journal Information Science. 2008;259-283.
DOI: 10.1016/j.ins.2008.10.027

[28] Thomas S, Bodagala S, Alsabti K, Ranka S. An efficient algorithm for the incremental updation of association rules in large databases. Proceeding of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA. 1997;263-266.

[29] Toivonen H. Sampling large databases for association rules. Proceedings of the 22th International Conference on Very Large Data Bases. 1996;134-145.

[30] Totad SG, Geeta RB, Prasad Reddy PVGD. Batch processing for incremental FP-tree construction. Int J Comput Appl IJCA. 2010;5(5):28-32.

[31] Totad SG, Geeta RB, Prasad Reddy PVGD. Batch incremental processing for FP-tree construction using FP-Growth algorithm. Knowledge Information Systems; 2012.

[32] Valtchev P, Missaoui R. Building concept (Galois) lattices from parts: Generalizing the incremental methods. In Proceedings of the 9th international conference on conceptual structures. California, USA: Springer. 2001;290-303.

[33] Valtchev P, Missaoui R, Godin R. A framework for incremental generation of closed itemsets. Discrete Applied Mathematics. 2008;156(6):924-949.

[34] Veloso A, Possas B, Jr. WM, de Carvalho MB. Knowledge management in association rule mining. Workshop on Integrating Data Mining and Knowledge Management (in conjuction with ICDM2001); 2001.

[35] Veloso AA, Jr. WM, de Carvalho MB, Pôssas B, Parthasarathy S, Zaki MJ. Mining frequent itemsets in evolving databases. Proc. 2nd SIAM Intl. Conf. on Data Mining, Arlington, VA; 2002.

[36] Vishnu Priya R, Vadivel A, Thakur RS. Frequent pattern mining using modified CP-tree for knowledge discovery. Springer, Berlin, Heidelberg. LNCS. 2010;6440:254–261.
DOI: 10.1007/978-3-642-17316-5-24

[37] Vo B, Hong TP, Le B. A lattice-based approach for mining most generalization association rules. Knowledge-Based Systems. 2013;45:20-30.

[38] Wang J, Han J, Pei J. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. Proceeding of the 2003 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’03), Washington, DC. 2003;236-245.

[39] Zaki MJ, Hsiao CJ. CHARM: An efficient algorithm for closed itemset mining. Proceeding of the 2002SIAM International Conference on Data Mining (SDM’02), Arlington, VA. 2002;457-473.

[40] Zhou Z, Ezeife CI. A low-scan incremental association rule maintenance method. Proceedings of the 14th Canadian Conference on Artificial Intelligence; 2001.