Mining Closed Itemsets in Data Stream Using Formal Concept Analysis

Anamika Gupta, Vasudha Bhatnagar, and Naveen Kumar

Department of Computer Science, University of Delhi, India
{agupta,vbhatnagar,nk}@cs.du.ac.in

Abstract. Mining of frequent closed itemsets has been shown to be more efficient than mining frequent itemsets for generating non-redundant association rules. The task is challenging in data stream environment because of the unbounded nature and no-second-look characteristics.

In this paper, we propose an algorithm, CLICI, for mining all recent closed itemsets in landmark window model of online data stream. The algorithm consists of an online component, which processes the transactions arriving in the stream without candidate generation and updates the synopsis appropriately. The offline component is invoked on demand to mine all frequent closed itemsets. User can explore and experiment by specifying the support threshold dynamically.

The synopsis, CILattice, stores all recent closed itemsets in the stream. It is based on Concept Lattice - a core structure of Formal Concept Analysis (FCA). Closed itemsets stored in the form of lattice facilitate generation of non-redundant association rules and is the main motivation behind using lattice based synopsis.

Experimental evaluation using synthetic and real life datasets demonstrates the scalablility of the algorithm.

Keywords: Closed Itemsets, Data Stream, Landmark Window Model Formal Concept Analysis.

1 Introduction

A data stream is an unbounded sequence of data often coming at a high speed. The problem of mining data stream is more challenging than mining static datasets in view of following aspects[2,3,10]. First, stream is a continuous flow of data and hence data must be processed at a rate faster than its arrival. Second, each element of stream must be examined only once. Third, memory usage should be bounded even though the stream is continuously growing. Further, the results should be instantly available in real time and error in the results, if any, should be bounded. Since stream evolves with time, capturing recent information is another vital issue in data stream mining.

Mining of frequent itemsets (FI) for association rules has been studied extensively in both static datasets [6] and data stream [3]. Researchers have recently explored the idea of mining frequent closed itemsets (FCI) instead of frequent itemsets for discovering non-redundant association rules [11,12,16]. Set of FCI has been shown to be a complete, loss-less and reduced representation of set of FI [11]. Mining FCI instead of FI saves computation efforts and memory usage. Several algorithms like Closet, CHARM, Closet+, CHARM-L, FP-Close, DCI-Closed [6] have been proposed to generate FCIs in static datasets.

Mining frequent closed itemsets in data stream throws newer challenges. Since every transaction in stream is a closed itemset [13], it leads to addition of at least one entity in the synopsis. The computational challenge is an immediate consequence of the large size of the synopsis and may result into loss of data because of higher per-transaction-processing time.

Traditionally, algorithms for mining frequent closed itemsets (FCI) in data stream use either sliding window model [4,7,9,14] or landmark window model [10] to meet the dual objective of 'maintaining recency' and 'constrained memory usage'. Sliding window model constrains the size of the synopsis by fixing the number of transactions in the window, leading to nearly constant per transaction processing time. However, the model falls short of monitoring the continuous variation of data stream.

Landmark window model considers entire data starting from a particular landmark to the curent time [2]. Although this model enables monitoring of gradual changes in the data stream, capturing recent data and keeping the size of synopsis under control is a challenging task.

State of the Art. Moment [14], CFI-Stream [7], NewMoment [9] and GC-Tree [4] are some of the known algorithms for mining FCI in sliding window model of data stream. Moment, CFI-Stream, and GC-Tree algorithms store current transactions in a memory based window. On arrival of a new transaction, these algorithms make multiple scans of the window for finding support of discovered itemsets. NewMoment stores bitwise representation of all the transactions along with 1-itemsets to find support of itemsets. Moment and CFI-Stream generate all candidates while processing new transaction, leading to increased per-transaction-processing time.

To the best of our knowledge, FP-CDS [10] algorithm has been recently proposed for discovery of frequent closed itemsets in landmark window model. The algorithm works in a batch mode, dividing the landmark window into several 'basic' windows, using them as updating units. However, in this process it ignores the recency of discovered itemsets. Recency in landmark window model has been handled by estDec [2], an algorithm for mining frequent itemsets. The effect of older transactions is diminished by decaying their old occurences and later pruning the decayed ones periodically, thus maintaining only the recent data.

Formal Concept Analysis Technique. Recently, data mining researchers have exploited Formal Concept Analysis(FCA), a field of applied mathematics [5], for discovery of closed itemsets. Zaki [16], Pasquier et al. [11] and Stumme et al. [12] proved that intent of concept (section 2) represents closed itemset and all concept generating algorithms generate closed itemsets as well. FCA based algorithms generate all the concepts (closed itemsets) and have been shown to work well on small datasets [8,15].

Substantial reduction in the set of generated rules is the main motivation for exploring concept lattice in association rule mining. FCA based mining algorithms store the closed itemsets in the form of a concept lattice. Concept lattice facilitates the efficient generation of non-redundant association rules [16]. One look characteristic of the updation algorithms for concept lattice is another advantage of using lattice in discovery of closed itemsets.

High memory and computational requirement of FCA based algorithms prohibits their use in data stream environment. Each node of the concept lattice stores the extent (set of transactions) alongwith the intent (closed itemset) which contribute to high memory usage. Further, processing of a new transaction involves computation of intersection of its extent with extent of different nodes in the lattice making it computationally expensive.

Further, decaying and pruning away older data to capture recent changes in the dataset has not been addressed in these algorithms and hence makes them unsuitable for data stream.

Our Contribution. In this paper, we present an algorithm for mining recent closed itemsets in landmark window model of on-line data stream. The proposed algorithm has an online component that processes the transactions without candidate generation and stores the results in a synopsis data structure. Offline component is invoked on demand and mines the closed itemsets from the synopsis based on dynamically specified support threshold. The salient features of the proposed algorithm CLICI (Concept Lattice based Incremental Closed Itemset) are listed below:

1. The algorithm mines *all* recent closed itemsets in landmark window model of online data stream. It fades out the obsolete information of old transactions using a decay function and later prunes the decayed information, thereby ensuring the recency of closed itemsets and in turn keeping size of the synopsis under control.
2. The algorithm processes the transactions without any candidate generation.
3. The algorithm is based on sound mathematical foundation of Formal Concept Analysis and stores the closed itemsets in a lattice based synopsis, CILattice. Following advantages accrue due to use of lattice based synopsis.
   (a) Generation of non-redundant association rules is naturally facilitated.
   (b) Data is scanned only once for maintaining the synopsis.
   (c) Only closed itemsets are stored in CILattice, unlike concept lattice where set of transactions are stored additionally.
4. CILattice maintains *all* closed itemsets, thereby providing a valuable facility for experimentation with varying support threshold without any overhead.
5. Experimental evaluation using synthetic and real life datasets demonstrates the scalablility of the algorithm.

The remainder of this paper is organized as follows. Section 2 defines closed itemsets and presents the background of Formal Concept Analysis (FCA). Section 3 describes the algorithm in detail for mining closed itemsets in landmark window model. Section 4 presents the experimental results and section 5 concludes the paper.

2 Background

Given a database D of N transactions and a set I of n items in D. A transaction $t \in D$ is a set of items and is associated with a unique identifier TID. A set of one or more items belonging to I is termed as an itemset. A $k-itemset$ is an itemset of cardinality k. An itemset X is **frequent closed itemset (FCI)** if it is frequent and there exists no proper superset Y of X such that support of Y is same as that of X.

Formal Concept Analysis. Following Ganter and Wille [5], we give some definitions and a theorem used in the paper.

Definition 1. *A formal context $K = (G, M, I)$ consists of two sets G (objects) and M (attributes) and a relation I between G and M. For a set $X \subseteq G$ of objects, the set of all attributes common to the objects in X is defined as $X' = \{m \in M | gIm \text{ for all } g \in X\}$. Correspondingly, for a set Y of attributes, the set of objects common to the attributes in Y is defined as $Y' = \{g \in G | gIm \text{ for all } m \in Y\}$. A formal concept of the context (G, M, I) is a pair (X, Y) with $X \subseteq G$, $Y \subseteq M$, $X' = Y$ and $Y' = X$. X is called the extent and Y is the intent of the concept (X, Y).*

Definition 2. *If (X_1, Y_1) and (X_2, Y_2) are concepts of a context, (X_1, Y_1) is called a subconcept of (X_2, Y_2), provided that $X_1 \subseteq X_2$ (which is equivalent to $Y_2 \subseteq Y_1$). In this case, (X_2, Y_2) is a superconcept of (X_1, Y_1) and we write $(X_1, Y_1) \leq (X_2, Y_2)$. The relation $\leq$ is called the hierarchical order of the concepts. The set of all concepts of (G, M, I) ordered in this way is called the concept lattice of the context (G, M, I).*

Definition 3. *For an object $g \in G$, $g' = \{m \in M | gIm\}$ is object intent of g. Correspondingly, $m' = \{g \in G | gIm\}$ is the attribute extent of the m.*

Theorem 1. *Each Concept of a context (G, M, I) has the form (X'', X') for some subset $X \subseteq G$ and the form (Y', Y'') for some subset $Y \subseteq M$. Conversely all such pairs are Concepts. This implies every extent is the intersection of attribute extents and every intent is the intersection of object intents.*

For proof of the theorem, please refer to [5].

3 CLICI Algorithm

CLICI algorithm mines all recent closed itemsets without candidate generation in landmark window model of data stream. Use of landmark window model facilitates continuous monitoring of changes in the stream. Maintaining all recent closed itemsets allow user to experiment with varying support thresholds. The algorithm has an online component that processes the transactions in stream and stores the results in a synopsis called CILattice. The algorithm inserts the new transaction in the lattice, if not already there. Offline component is invoked on demand and mines the closed itemsets from the synopsis based on dynamically specified support threshold.

3.1 Terminology and Data Structure

Let D_N denote the current data stream with N transactions seen so far and I denote the set of n items in D_N. The incoming transactions are inserted in a data structure, CILattice, which has two components i) a lattice $\mathcal{L}$ ii) header table *Itable*. $\mathcal{L}$ is a complete lattice, with topnode $\top$ and bottom node $\perp$. A node X of $\mathcal{L}$ represents a closed itemset I_X and stores its frequency f_X along with links to its parents and children nodes. *Itable* is an array storing items and pointers to the nodes corresponding to first occurrence of that item in $\mathcal{L}$, which aids efficient traversal during search and insert procedure. The definitions and observations used later in the algorithm are given below:

Let $A(X)$, $D(X)$, $P(X)$ and $C(X)$ denote the set of ancestors, descendants, parents and children respectively of the node X in $\mathcal{L}$.

Definition 4. *A node X is ancestor of node Y iff $I_X \subset I_Y (I_X \neq I_Y)$.*

Definition 5. *A node X is descendant of node Y iff $I_X \supset I_Y (I_X \neq I_Y)$.*

Definition 6. *A node X is parent of a node Y if $X \in A(Y)$ and $\nexists$ any $Z \in A(Y) : X \in A(Z)$ and $Z \neq X$.*

Definition 7. *A node X is a child of node Y if $X \in D(Y)$ and $\nexists$ any $Z \in D(Y) : X \in D(Z)$ and $Z \neq X$.*

It is obvious from the above definitions that ancestor nodes are generalizations of the descendant nodes. Further, parents of a node are its immediate ancestors (generalizations) and children of a node are its immediate descendants (specializations).

Observation 1. *If node X has a child node Y in $\mathcal{L}$ then closed itemset of Y is minimal superset of all descendants of X. Similarly closed itemset of X is maximal subset of all ancestors of Y.*

Definition 8. *First Node F_i of an item i is a node X in the $\mathcal{L}$ where $i \in I_X$ and $\nexists$ a node Y such that $i \in I_Y$ and $Y \in A(X)$.*

Naturally there is exactly one F_i for each item $i \in I$ and Itable stores the pair (i, F_i). Fig. 1 shows a toy database and the corresponding data structure $< \mathcal{L}, ITable >$.

Example 1. Fig. 1 shows a toy database and the corresponding $< \mathcal{L}, Itable >$.

3.2 Capturing Recent Closed Itemsets

CLICI algorithm maintains all closed itemsets starting from the set landmark point. The effect of older transactions is diminished by decaying their old occurences and later pruning the decayed ones, thus maintaining only the recent closed itemsets. This feature controls the size of lattice also.

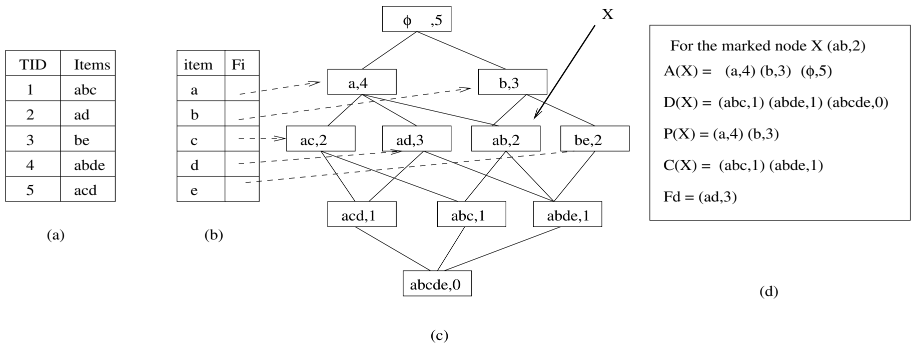

Fig. 1. (a) Toy Database (b) Itable (c) $\mathcal{L}$ (d) Set of ancestors, descendants, parents and children of node X

Chang and Lee [2] define a decay factor d which is associated with each transaction in the stream and is defined as $d = b^{-1/h}$ $(b > 1, h \geq 1, b^{-1} \leq d < 1)$, where decay-base b determines the amount of weight reduction per a decay-unit and decay-base-life h is defined by the number of decay-units that makes the current weight be b^{-1}. When a new transaction t_k arrives at time k, number of transactions in the current stream $|D|_k$ is updated as $|D|_k = |D|_{k-1} * d + 1$. Decayed frequency of all nodes corresponding to itemsets of t_k is updated as $(f_k = f * d^{k-(MRtid_{pre})} + 1)$, where f denote the decayed frequency of node and MRtid is the transaction identifier of most recent transaction that contain t_k. $MRtid_k$ is set to k. Decayed support count of nodes is calculated as $f/(|D|_k)$.

Periodically, the lattice is traversed and all nodes having support count less than threshold for pruning (S_{prn}) are removed from $\mathcal{L}$. Note here that S_{prn}, which is also user specified parameter, is distinctly different from support threshold. While support threshold deals with the frequency of the itemset, S_{prn} takes into account the effect of age of the itemset. Higher values of S_{prn} leads to considerable reduction in the size of the synopsis. Since, data characteristics of the stream change with time, guessing the correct value of S_{prn} is a difficult task. An alternative approach, though somewhat crude, is to fix the size of synopsis based on total available memory and remove decayed nodes from the lattice so as to keep the recent and repetitively occurring itemsets intact.

3.3 Processing of Transaction

Each incoming transaction t is a closed itemset [13]. In case transaction t exist already as a node in $\mathcal{L}$, then the support of the relevant nodes is updated as mentioned in section 3.2. Otherwise processing of t results into insertion of one or more nodes in $\mathcal{L}$.

If t exists as a node in $\mathcal{L}$ then we write $t \in \mathcal{L}$ for simplicity purposes.

We present below the procedures for search, insert and delete a node.

Search Procedure. This procedure checks whether incoming transaction t exists as a node in $\mathcal{L}$ or not. The bruteforce approach of searching $\mathcal{L}$ for node containing t would be to start search either from $\top$ or $\bot$. This approach is computationally expensive as the lattice has a tendency to grow in size as the stream progresses. Proposition 1 permits an optimization by searching *First Node* corresponding to any one of the items in the transaction and its descendents.

Proposition 1. *Let t be the transaction to be searched in $\mathcal{L}$. It is necessary and sufficient to search among F_i and its descendants, for an arbitrarily chosen i, $i \in t$.*

Proof. If $t = I_{F_i}$ for an arbitrarily chosen i, $i \in t$, then $t \in \mathcal{L}$.

By definition 5 and 8, $i \in I_{F_i}$ $\forall i \in t$. $D(F_i)$ is a set containing all supersets of I_{F_i}. If $t \in \mathcal{L}$ then t is a superset of I_{F_i} and hence t exists as a descendant of F_i. Thus t is necessarily a descendant of each of the items contained in t.

By definition 5 and 8, no other node except nodes belonging to $D(F_i)$ contain i. Since $i \in t$ so if $t \in \mathcal{L}$ then t is among the descendants of F_i. Hence it is sufficient to search among descendants of F_i.

Thus, if t contains an item i for which F_i does not exist, then $t \notin \mathcal{L}$. Otherwise it needs to be searched among F_i and its descendants corresponding to any one of the arbitrarily chosen item i in t.

Insert Procedure. If transaction t does not already exists as a node in $\mathcal{L}$ then it is added as a new node N in $\mathcal{L}$, with $I_N = t$. Subsequently N is linked to its child nodes as well as parents node in $\mathcal{L}$. Insert Procedure determines the minimal superset of I_N to find the child nodes and the maximal subsets of I_N to find the parents of I_N (Observation 1). Proposition 2 states that Node N can have only one child in $\mathcal{L}$.

Proposition 2. *If N is the node corresponding to transaction t that has been just added to lattice $\mathcal{L}$, then $|C(N)| = 1$.*

Proof. We prove this assertion by contradiction. Let, if possible, $|C(N)| = m > 1$. Let $C(N) = C_1, C_2, \ldots, C_m$ be the set of children of N in $\mathcal{L}$. By definition, $I_N = I_{C_1} \cap I_{C_2} \ldots \cap I_{C_m}$. According to theorem 1, intersection of extents of nodes in lattice is always an extent of a node in lattice i.e. there must exist a node X in $\mathcal{L}$ such that $I_X = I_{C_1} \cap I_{C_2} \ldots \cap I_{C_m}$. But then X and N are the same nodes. This contradicts the fact that N is a new node. Hence the assumption that $|C(N)| = m > 1$ is not correct.

Further m cannot be zero because bottom node $\bot$ of $\mathcal{L}$ contains all the items and is always a superset of transaction t. Hence $m = 1$. ■

The insert procedure traverses $\mathcal{L}$ using *ITable* for finding the child node and parent nodes of N. The use of Proposition 2 reduces the effort involved in searching children nodes of N as it can have only one child in $\mathcal{L}$. However, potential number of ancestors can be $2^{|I_N|}$. The exact number of immediate ancestors i.e. parents is unpredictable and may involve rigorous searching in the lattice $\mathcal{L}$. This process of searching immediate ancestors i.e. parents of N is speeded up using F_i of

Itable, $i \in t$. F_i provides the entry point for search in $\mathcal{L}$. However the involved task depends on the relationship of F_i and N. Following three cases arise:

Case 1: F_i is ancestor of N i.e. $I_{F_i} \subset I_N$. Child of N is among the descendants of F_i. Parent(s) of N is either the node F_i itself or some of F_i's descendant(s). If closed itemset of any of the child node X of F_i is superset of I_N then F_i is the parent of N and X is the child of N. Otherwise we search amongst the descendants of F_i till we find a node X such that I_X is superset of I_N and closed itemset of one of the parent Y of X is subset of I_N. In that case, X is the child of N and Y is the parent of N.

Case 2: F_i is descendant of N i.e. $I_{F_i} \supset I_N$. F_i is the child of N and *First Node* F_i corresponding to $i \in I_N$ is set to N. We search among the parents of F_i for finding parents of N. If closed itemset of those parents is subset of I_N, then those parents are parents of N also. If no such parent exist then top node $\top$ of $\mathcal{L}$ is the parent of N.

Case 3: F_i is neither ancestor nor descendent of N i.e. $I_{F_i} \not\subset I_N$, $I_{F_i} \not\supset I_N$ and $I_{F_i} \cap I_N \neq \phi$. In this case, common parent of F_i and N with closed itemset as $I_{F_i} \cap I_N$ may exist in $\mathcal{L}$ or it may not exist.

Case 3.1: $I_{F_i} \cap I_N$ exists as closed itemset of a node X in $\mathcal{L}$. If such a node X exists in $\mathcal{L}$ then X is the parent of N. Child of N is among the descendants of F_i. We check all the descendants of F_i till we get a node whose closed itemset is minimal superset of closed itemset of N, that node becomes the child of N.

Case 3.2: $I_{F_i} \cap I_N$ does not exist as closed itemset of a node in $\mathcal{L}$. A new node X is created with $I_X = I_{F_i} \cap I_N$ and added to $\mathcal{L}$. X becomes the parent of N and child of N is among the descendants of F_i. We check all the descendants of F_i till we get a node whose closed itemset is minimal superset of closed itemset of N, that node becomes the child of N. Next step is to find the child and parents of X. Parents of X are among ancestors of F_i. We check all the ancestors of F_i. If we get a node Y such that closed itemset of Y is superset of X and closed itemset of one of the parent Z of Y is subset of X, then Z becomes the parent of X. If we find an ancestor Y of F_i such that Y is neither ancestor nor descendent of X, then a new node U is created with $I_U = I_Y \cap I_X$. Then we repeat the process to find child and parents of node U.

The algorithm for inserting a new transaction is given in Algorithm 27.

Delete procedure. $\mathcal{L}$ is traversed from bottom and all nodes having decayed support less than S_{prn} are removed by invoking delete procedure. Bottom node is never removed from the $\mathcal{L}$ so as to maintain the whole structure of $\mathcal{L}$. We may note here that if decayed support count of a node is less than S_{prn} then decayed support count of all its subsets is also less than S_{prn} i.e. if a node has been removed from $\mathcal{L}$ then all its descendents (except bottom) have already been removed from $\mathcal{L}$.

Let N denotes the node to be deleted. Node N will have one child node i.e. bottom and can have more than one parent. Delete procedure removes link of

N from its parents and child nodes. Two cases may arise: i) child node of N (i.e. bottom node) has no parent. In that case, parents of N become parents for bottom. ii) parent nodes of N has no child. In that case, bottom become child for parents of N.

```
Input: $\mathcal{L}$ - lattice of closed itemsets, $t$ - transaction to insert
Output: $\mathcal{L}$ - updated lattice
Process:
create a new node $N$ with $I_N = t$
for all item $i \in t$ in $\mathcal{L}$ do
  if $F_i$ is ancestor of $N$ then
    find child and parents of $N$ among the descendants of $F_i$
  else
    if $F_i$ is descendant of $N$ then
      child of $N$ is $F_i$ and parents of $N$ are among the ancestors of $F_i$
    else
      if common parent $X$ of $F_i$ and $N$ exist in $\mathcal{L}$ then
        parent of $N$ is $X$ and child of $N$ is among the descendants of $F_i$
      else
        create a new node $X$ where $I_X = I_{F_i} \cap I_N$
        parent of $N$ is $X$ and child of $N$ is among descendants of $F_i$
        children of $X$ are $F_i$ and some ancestor of $N$.
        parents of $X$ are among the ancestors of $F_i$.
        if $X$ is neither ancestor nor descendent of node $Y \in \mathcal{L}$ and $I_X \cap I_Y \neq \phi$
        then
          create a new node $U$ such that $I_U = I_Y \cap I_X$
          find child and parents of node $U$
        end if
      end if
    end if
  end if
  Update frequency of $N$.
end for
```

Algorithm 1. Insert Procedure

Update ITable. If N is added as an ancestor node of F_i in $\mathcal{L}$, First Node F_i corresponding to $i \in I_N$ is set to N. If a node N is deleted from $\mathcal{L}$ then First Node F_i corresponding to $i \in I_N$ is set to bottom as all the descendents of N (except bottom) have already been removed from $\mathcal{L}$.

4 Experimental Analysis

Since there is no algorithm known for mining all recent closed itemsets in landmark window model, it is not possible to perform a comparative analysis. We evaluate our algorithm for scalability using several synthetic and real life datasets. All experiments were done on a 2GHz AMD Dual-Core PC with 3 GB main memory, running redhat linux operating system. All algorithms are implemented in C++ and compiled using g++ compiler without optimizations. In all experiments, the transactions of each dataset are examined one by one in sequence to simulate the environment of an online data stream.

Experiments on Synthetic data
We generated four different datasets T3I4D100K, T5I4D100K, T8I4D100K, T10I4D100K using IBM data generator [1]. Three numbers of each dataset denote the average transaction length (T), average maximum potential frequent itemset size (I) and the total number of transactions (D) respectively.

Fig. 2 (a) shows the effect of pruning threshold S_{prn} on per-transaction-processing time in the dataset T10I4D100K, as stream progresses. Higher values of S_{prn} lead to substantial reduction in the size of synopsis. Hence per-transaction-processing time is nearly constant. As value of S_{prn} decreases, size of the synopsis increases, leading to increase in rate of growth of per-transaction-processing time. However if size of synopsis is fixed, per transaction processing time is nearly

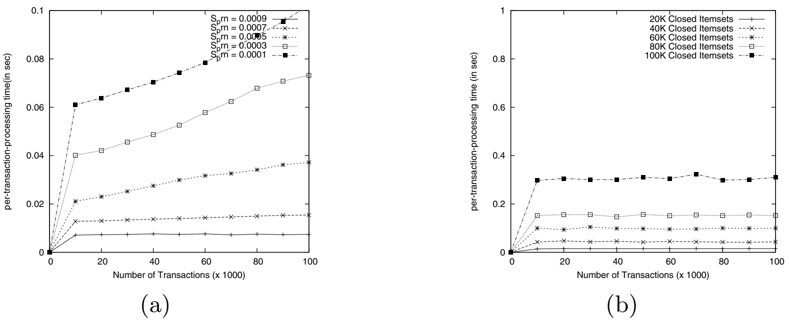

Fig. 2. Per-transaction-processing time corresponding to (a) different values of S_{prn} (b) fixed size of synopsis in T10I4D100K dataset

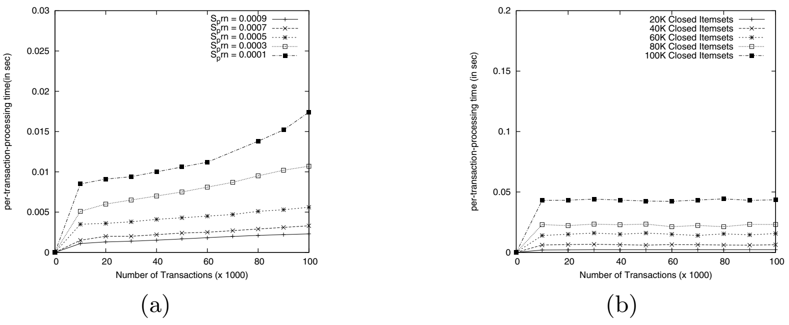

Fig. 3. Per-transaction-processing time corresponding to (a) different values of S_{prn} (b) fixed size of synopsis in BMS-Web-View-1 dataset

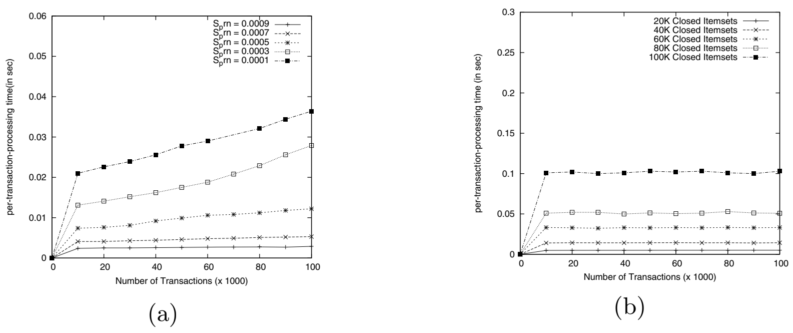

Fig. 4. Per-transaction-processing time corresponding to (a) different values of S_{prn} (b) fixed size of synopsis in BMS-POS dataset

constant. Fig 2 (b) shows the results for different sizes of synopsis. This establishes the scalability of the algorithm.

Experiments on Real Life datasets
We experimented on two real life datasets, BMS-Web-View-1 and BMS-POS [17]. BMS-Web-View-1 contains a few months of clickstream data from an e-commerce web site [17]. There are 59602 transactions with 497 items; average transaction length is 2.5 and maximum transaction length is 267. BMS-POS contains several years of point of sale data from a large electronics retailer [17]. There are 515,597 transactions with 1657 items; average transaction length is 6.5 and maximum transaction length is 164.

Fig. 3 (a) and Fig. 4 (a) shows graph depicting per-transaction-processing time when CLICI runs on BMS-Web-View-1 and BMS-POS datasets respectively. Different values of pruning threshold S_{prn} are tested and it is observed that higher values of S_{prn} leads to reduced size of the synopsis and hence reduced per-transaction-processing time. It is also observed that rate of growth of per-transaction-processing time increases with decrease in pruning threshold. Fig. 3 (b) and Fig. 4 (b) present results on different sizes of the synopsis, 20K, 40K, 60K, 80K, 100K closed itemsets. As depicted in the graphs, per-transaction-processing time remains nearly constant on a particular size of the synopsis and increases with increase in size of synopsis.

5 Conclusion and Future Work

We have proposed CLICI algorithm to mine all recent closed itemsets in landmark window model of data stream. The algorithm is based on Formal Concept Analysis, a well established discipline in applied mathematics. The proposed algorithm maintains a lattice of recent closed itemsets in the stream and delivers frequent closed itemsets to the user on demand, based on the dynamically specified support threshold. Use of lattice as a synopsis facilitates efficient formulation of non-redundant association rules as shown in earlier works. Since the synopsis is independent of the support threshold, user is encouraged to explore and experiment.

References

1. Agarwal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: 20th International Conference on Very Large Databases, pp. 487–499 (1994)
2. Chang, J., Lee, W.: Finding Recent Frequent Itemsets Adaptively over Online Data stream. In: 9th ACM SIGKDD, pp. 487–492. ACM Press, New York (2003)
3. Cheng, J., Ke, Y., Ng, W.: A Survey on Algorithms for Mining Frequent Itemsets over Data stream. KAIS Journal 16(1), 1–27 (2008)
4. Chen, J., Li, S.: GC-Tree: A Fast Online Algorithm for Mining Frequent Closed Itemsets. In: Proceeding of PAKDD Workshop of HPDMA, pp. 457–468 (2007)
5. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Heidelberg (1999)
6. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent Pattern Mining: Current Status and Future Directions. Journal of Data Mining and Knowledge Discovery 15, 55–86 (2007)
7. Jiang, N., Gruenwald, L.: CFI-Stream: Mining Closed Frequent Itemsets in Data stream. In: 12th ACM SIGKDD, Poster Paper, pp. 592–597. ACM Press, New York (2006)
8. Kuznetsov, S.O., Obiedkov, S.A.: Comparing Performance of Algorithms for Generating Concept Lattices. JETAI 14, 189–216 (2002)
9. Li, H., Ho, C., Lee, S.: Incremental Updates of Closed Frequent Itemsets Over Continuous Data stream. Expert Systems with Applications 36, 2451–2458 (2009)
10. Liu, X., Guan, J., Hu, P.: Mining Frequent Closed Itemsets from a landmark window over online data stream. Journal of Computers and Mathematics with Applications 57(6), 927–936 (2009)
11. Pasquier, N., et al.: Efficient Mining of Association Rules using Closed Itemset Lattices. Journal of Information Systems 24(1), 25–46 (1999)
12. Stumme, G., et al.: Computing Iceberg Concept Lattices with Titanic. Journal on Knowledge and Data Engineering 42(2), 189–222 (2002)
13. Valtchev, P., Missaoui, R., Godin, R.: A framework for incremental generation of closed itemsets. Discrete Applied Mathematics 156(6), 924–949 (2008)
14. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Catch the Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. Journal of Knowledge and Information Systems 10, 265–294 (2006)
15. Yahia, S.B., Hamrouni, T., Nguifo, E.M.: Frequent Closed Itemset Based Algorithms: A thorough structural and analytical survey. ACM SIGKDD Explorations Newsletter 8, 93–104 (2006)
16. Zaki, M.J.: Generating Non-Redundant Association Rules. In: 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 34–43. ACM Press, New York (2000)
17. Zheng, Z., Kohavi, R., Mason, L.: Real World Performance of Association Rule Algorithms. In: Proceedings of the 2001 International Conference Knowledge Discovery and Data Mining, SIGKDD 2001 (2001)