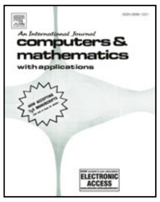
Computers and Mathematics with Applications 57 (2009) 927–936

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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



Mining frequent closed itemsets from a landmark window over online data streams[☆]

Xuejun Liu[a,c,*], Jihong Guan[b], Ping Hu[c]

[a] Department of Computer Science & Engineering, Fudan University, Shanghai 200433, China
[b] Department of Computer Science & Technology, Tongji University, Shanghai 200092, China
[c] College of Information Science & Engineering, Nanjing University of Technology, Nanjing 210009, China

ARTICLE INFO

Keywords:
Data streams
Frequent closed items
Landmark window
Data mining

ABSTRACT

The frequent closed itemsets determine exactly the complete set of frequent itemsets and are usually much smaller than the later. However, mining frequent closed itemsets from a landmark window over data streams is a challenging problem. To solve the problem, this paper presents a novel algorithm (called FP-CDS) that can capture all frequent closed itemsets and a new storage structure (called FP-CDS tree) that can be dynamically adjusted to reflect the evolution of itemsets' frequencies over time. A landmark window is divided into several basic windows and these basic windows are used as updating units. Potential frequent closed itemsets in each basic window are mined and stored in FP-CDS tree based on some proposed strategies. Extensive experiments are conducted to validate the proposed method.

1. Introduction

In recent years, database and knowledge discovery communities have focused on a new data model, in which data arrive in the form of continuous data streams. Although frequent pattern mining has been widely studied, it is challenging to extend it to data streams. Compared to mining from a static transaction dataset, mining data stream has far greater complexity to manage and far more information to track. First, it is unrealistic to keep the entire stream in main memory or even in secondary storage, since data stream come continuously. Second, traditional methods of mining on static datasets by multiple scans are infeasible since the streaming data is passed only once. Third, mining data streams requires fast, real-time processing in order to keep up with the high data arrival rate. In addition, Infrequent items can become frequent later on and hence cannot be ignored. The storage structure needs to be dynamically adjusted to reflect the evolution of itemset frequencies over time.

The number of frequent patterns in data streams is often very large, and it is difficult to understand and use these patterns. A frequent pattern of length n implies that $2^n - 1$ subsets of the frequent pattern are required to examine one by one. When n is large, it is practically unfeasible to mine the set of all frequent patterns. Pasquier et al. proposed the concept of frequent closed itemsets [1]. Frequent closed itemsets are lossless in the sense that they uniquely determine the set of all frequent itemsets and their exact frequency. At the same time frequent closed itemsets can themselves be orders of magnitude smaller than all frequent itemsets, especially on dense databases. Subsequently, many frequent closed pattern mining algorithms have been developed [1–3]. However, those algorithms are not suitable for data stream mining.

☆ This work is supported by National Natural Science Foundation of China under grants no. 60496327, no. 60573183 and no. 60773123, Jihong Guan is also supported the Program for New Century Excellent Talents in University of China (NCET-06-0376).

* Corresponding author at: Department of Computer Science & Engineering, Fudan University, Shanghai 200433, China.
E-mail addresses: xuejliu@fudan.edu.cn (X. Liu), jhguan@mail.tongji.edu.cn (J. Guan), huping@njut.edu.cn (P. Hu).

0898-1221/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2008.10.060

In this paper, we present a new algorithm (called FP-CDS) for frequent closed itemsets mining over data streams from landmark windows. A new storage structure (called FP-CDS tree) is adopted and it can be dynamically adjusted to reflect the evolution of itemsets' frequencies over time. The current closed frequent itemsets can be output in real time based on any user's specified thresholds. We then conduct simulation experiments using synthetic datasets to evaluate the performance of our proposed algorithm. The experimental results indicate that our proposed method is efficient.

The rest of paper is organized as follows. We discuss the related work in Section 2 and give the problem definition in Section 3. In Section 4, our algorithms are discussed in detail followed by analysis. We conducted extensive experimental studies, and report our experimental results in Section 5. We conclude our work in Section 6.

2. Related work

Many approaches have been recently proposed to address frequent itemsets mining over data streams. Giannella [4] et al. propose a FP-tree-based algorithm, called FP-streams, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique. Manku [5] et al. propose two single-pass algorithms, sticky-sampling and lossy counting, to mine frequent itemsets over landmark windows. Karp [6] and Charikar [7] focus on single frequent pattern mining. Chang [8] et al. study the discovery of recent frequent patterns and develop a BTS-based algorithm for mining frequent itemsets in sliding windows model. Teng [9] et al. study temporal sequential frequent pattern mining problem over sliding windows, and put forward a regression-based algorithm called FTP-DS. Graham [10] and Tatsuya [11] study hierarchical frequent pattern mining of data streams and frequent pattern mining of semi-structured data streams, respectively. Graham also proposes a group test method to find frequent patterns [12]. Jin [13] puts forward a hash-based approach to discover frequent patterns. The experiments show that its efficiency is better than the group test method in [12]. The methods in [12,13] are more suitable to single pattern or fixed pattern. Zhang Xin [14] et al. improve the FP-stream algorithm and put forward FPIL-STREAM algorithm. The time efficiency is priority over FP-stream. In [15,16], the authors propose algorithms to mine closed frequent itemsets over a data stream sliding window. In [17], Mao et al. propose an algorithm to find maximal frequent itemsets from data streams. Lin et al. [18] propose an incremental mining algorithm to find the set of frequent itemsets in a time-sensitive sliding window. Jin and Agrawal [19] present an algorithm, called StreamMining, for in-core frequent itemset mining over data streams. Li et al. [20] propose an efficient one-pass algorithm, called NewMoment, to maintain the set of closed frequent itemsets in data streams with a transaction-sensitive sliding window.

In [4–14,18,19], those algorithms focus on frequent itemsets, instead of frequent closed itemsets, with one scan over entire data stream. In [15,16,20], the author study that discovery of frequent closed itemsets over sliding window. However, those researches do not involve the mining of frequent closed itemsets over a data stream landmark window. The mining of frequent closed itemsets based on a landmark window has not yet been seen in related research reports.

3. Problem definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items, $T = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where transaction t_i $(i = 1, 2, \ldots, n)$ is a subset of I, i.e. $t_i \subseteq I$. Transactions can be represent by unique transaction identifier, called TID, or the itemsets which transactions contain.

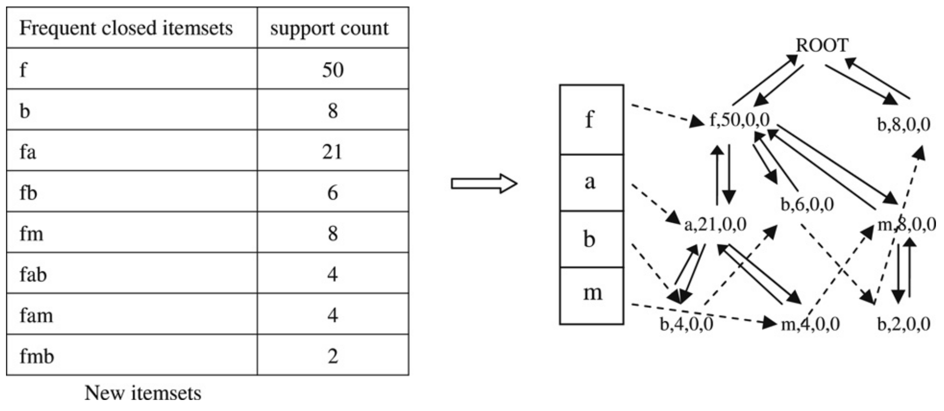
Definition 1. Data stream DS is an ordered sequence of transactions that arrive in a timely order, i.e. $DS = \{t_1, t_2, \ldots, t_i, \ldots, t_j, \ldots\}$. Suppose itemset X is a subset of I, the number of transactions containing itemset X, denoted as $f_{DS}(X)$, is called the support count of itemset X. The support of itemset X, denoted as $X.sup$, is defined as $f_{DS}(X)/|DS|$, where $|DS|$ is the number of transactions in data streams DS. An itemset with k items is called a k-itemset.

Definition 2. The basic window w consists of some transactions arriving continuously, i.e. $w = \{t_i, t_{i+1}, \ldots, t_j\}$, where t_i is original transaction, t_j is "*latest*" transaction. A landmark window W consists of many neighboring basic windows, i.e. $W = \{w_1, w_2, \ldots, w_i, \ldots, w_j, \ldots\}$, where w_1 is original basic window, and the current length of W continually changes along with new transactions arriving.

Definition 3. Let $|N|$ be the number of transactions in window DSW (basic window or landmark window), and the support count of itemset X be denoted as $f_{DSW}(X)$. Given minimum support threshold S and maximum support error threshold ε, if $f_{DSW}(X) > (S - \varepsilon)|N|$, X is called frequent itemset. If $f_{DSW}(X) > \varepsilon|N|$, X is called subfrequent itemset. If $f_{DSW}(X) \leq \varepsilon|N|$, X is called infrequent itemset.

Definition 4. A context is a triple (O, I, R). O and I are finite sets of objects and items, respectively. $R \subseteq O \times I$ is a binary relation between objects and items. Each couple $(o, i) \in R$ denotes the fact that object $o \in O$ is related to the item $i \in I$.

Definition 5. Suppose O is a context, according to $O' \subset O$ and $I' \subset I$, we define that $f(O') = \{i \in I | \forall o \in O', (o, i) \in R\}$, and $g(I') = \{o \in O' | \forall i \in I', (o, i) \in R\}$, then $f(O')$ is called common itemsets set of transaction $O' \subset O$, $g(I')$ is called support transactions set of itemset $I' \subset I$.



Frequent closed itemsets	support count
f	50
b	8
fa	21
fb	6
fm	8
fab	4
fam	4
fmb	2

New itemsets

Fig. 1. Structure of the FP-CDS tree.

Definition 6. Operator $h = f \cdot g, h(I) = f(g(I))$, itemset $C \subset I$ is closed, Only when $h(C) = C$. If C is frequent itemset, then C is called frequent closed itemset. And if C is subfrequent itemset, then C is called subfrequent closed itemset.

Definition 7 (*FP-CDS Tree*). A FP-CDS tree is a tree structure defined below.

(1) It consists of one root node labeled as 'null', a set of item-prefix subtree as children of the root, and a subfrequent-item-header table. The item-prefix subtree consists of subfrequent closed itemsets.

(2) Except root node, each node in the item-prefix subtree consists of seven fields: *(data, f, fnew, dsgrow, par, leftchild, rightchild)*, where *data* is item-name of the node, f denotes the number of the itemset represented by the portion of the path reaching this node, *fnew* is a count-value for avoiding the repeated insertion of itemset when re-constructing a FP-DS tree, *dsgrow* is a sign for avoiding frequent itemsets output repeatedly, *par* is a pointer pointing to parent node, *Leftchild* is a pointer which points to the first child node, and *rightchild* is the one pointing to brother node;

(3) The header table is similar to that of FP-tree.

Fig. 1 shows the structure of the FP-CDS tree.

Definition 8. A path is formed from the node e of the FP-CDS tree to root node. All nodes in this path compose an itemset and the itemset is called the pattern of e (pattern e in short). The set of all patterns e is called pattern set of e.

Theorem 1. *If the infrequent itemsets in the current batch are ignored, even they becomes frequent later on, their support error at most cannot exceed ε. According to Definition 3, deleting those infrequent itemsets does not affect the correct output of frequent itemsets.*

Proof. Suppose data streams DS consists of k batches. $|N_i|$ denotes the length of the ith batch, namely the number of transactions which the batch contains. $|N|$ denotes the length of data streams DS. Given maximum support error ε, if itemset X in batch N_i is infrequent itemset, then

$$f_{N_i}(X) \le \varepsilon|N_i|, \quad 0 \le i < k.$$

Suppose support of itemset X in data streams DS is S and itemset X is infrequent in m ($m < k$) batches. If we ignore $f_{N_i}(X)$ in m batches, the approximate count of itemset X in DS is denoted as $f'_N(X)$, then

$$f'_N(X) = f_N(X) - \sum f_{N_i}(X) = S|N| - \sum f_{N_i}(X).$$

Furthermore,

$$\sum f_{N_i}(X) \le \varepsilon \sum |N_i| = \varepsilon * m * |N_i| < \varepsilon|N|.$$

Therefore,

$$f'_N(X) = S|N| - \sum f_{N_i}(X) > S|N| - \varepsilon|N| = (S-\varepsilon)|N|,$$

namely

$$f'_N(X) > (S-\varepsilon)|N|.$$

This shows that only to store subfrequent itemsets can guarantee the correct outputs of frequent itemsets which satisfy Definition 3. □

4. Mining frequent closed itemsets from a landmark window

The FP-CDS algorithm is used to mine frequent closed itemsets from a landmark window. The basic FP-CDS algorithm is described as follows:

(1) The incoming stream is conceptually divided into some batches, where each batch consists of $k * \lceil \frac{1}{\varepsilon} \rceil$ transactions. These batches are denoted as $N_1, N_2, \ldots$. (2) A set of global subfrequent 1-itemset, called f_list, is stored. In each batch, the algorithm scans all new transactions from the current batch and update f_list; (3) A new FP-CDS tree is constructed, and the original FP-CDS tree only contains a root node. (4) If the current batch $i > 1$, the subfrequent itemsets from the FP-CDS tree built in batch N_{i-1} are reordered according to the order of f_list. The items which are not included in f_list are ignored, and the rest are inserted into the new PF-CDS tree built in batch N_i. At one time the old one built in batch N_{i-1} is deleted. (5) The new transactions from the batch N_i are scanned and potential frequent closed itemsets are mined (support as ε) by existing frequent closed pattern algorithms. The itemsets are sorted in the order of f_list, and the items which are not included in f_list are ignored, and the rest are inserted into the FP-CDS tree. (6) Given a support S, the frequent closed itemsets based on the FP-CDS tree are obtained.

4.1. Global subfrequent 1-itemset

The data streams are divided into batches of fixed size. Let N_i $(i = 1, 2, \ldots)$ denote the ith batch. The number of transactions in batch N_i is $|N_i| = \lceil \frac{1}{\varepsilon} \rceil$ or $k * \lceil \frac{1}{\varepsilon} \rceil (k = 1, 2, \ldots)$. In order to facilitate description of the algorithm, let $|N_i| = \lceil \frac{1}{\varepsilon} \rceil$. The f_list consists of global subfrequent 1-itemsets. Each 1-itemset in f_list consists of three fields: *data*, f and *del*, where *data* is the item-name of 1-itemset, f denote the number of the 1-itemset, and *del* is a conditional-variable to delete the 1-itemset. When the transactions from batch N_i are scanned, if any 1-itemset $e_j \in f\_list$, let $e_j.f = e_j.f + 1$, $e_j.del = e_j.del + 1$. Otherwise, e_j is inserted into f_list, and let $e_j.f = 1$, $e_j.del = 1$. After scanning batch N_i, all 1-itemsets are sorted in their frequency descending order, and the value of *del* of each 1-itemset minus 1(that is $e_j.del = e_j.del - 1$). If $del = 0$, the 1-itemset is ignored. The detailed algorithm is described as follows.

```
Algorithm 1. Scands_ DB algorithm.
Input: data of batch $N_i$; the f_list of batch $N_{i-1}$ (when $i > 1$).
Output: the f_list of batch $N_i$.
(1) for every new data stream element $e_i$, if $e_i$ inf_list then
(2)   $e_i.f = e_i.f + 1$, $e_i.del = e_i.del + 1$;
(3) else insert $e_i$, let $e_i.f = 1$, $e_i.del = 1$;
(4) sort f_list in frequency descending order of 1-itemsets;
(5) for each $e_i \in f\_list${
(6)   $e_i.del = e_i.del - 1$;
(7)   if $e_i.del = 0$ then delete $e_i$.}
```

Theorem 2. *According to the above algorithm, the* 1-*itemsets deleted from f_list must be infrequent itemsets, and f_list is a set of subfrequent 1-itemsets.*

Proof. For any 1-itemset $a \in f\_list$, it follows that $a.del = a.del - 1$ in every batch containing a. Suppose the current batch is N_i, and $|N|$ denotes the number of all transactions so far. If there are k batches containing a, then

$$k \le i, |N| = i * |N_i|.$$

According to Algorithm 1, we have

$$a.f \le a.del + k.$$

Hence

$$a.f \le a.del + i.$$

Since the condition of ignoring a in batch N_i is $a.del = 0$, then

$$a.f \le i.$$

It is easy to see that

$$\varepsilon|N| = \varepsilon * i * |N_i| \ge \varepsilon * i * (1/\varepsilon) = i.$$

Hence,

$$a.f \le \varepsilon|N|,$$

i.e. a is an infrequent itemset, and f_list is a set of subfrequent 1-itemsets. If a 1-itemset is an infrequent itemset, its supersets must be infrequent itemsets too. □

Theorem 3. *The maximum storage space is $O(L/\varepsilon)$ for f_list, where L denotes the average length of the transactions.*

In order to prove Theorem 3, we need the Theorem 4.

Theorem 4. *Suppose a is a 1-itemset, $I(DS, \varepsilon) = \{a | f_{DS}(a) > \varepsilon * |DS|\}$, then we have $|I(DS, \varepsilon)| \leq L * \lceil 1/\varepsilon \rceil$.*

Proof. We use counter-proof. According to the definition, we obtain

$$\sum_{a \in I(DS,\varepsilon)} f_{DS}(a) \leq L * |DS|.$$

Since the supports of all items in $I(DS, \varepsilon)$ are all greater than $\varepsilon * |DS|$, then

$$\sum_{a \in I(DS,\varepsilon)} f_{DS}(a) \geq \sum_{a \in I(DS,\varepsilon)} \varepsilon * |DS| = |I(DS, \varepsilon)| * \varepsilon * |DS|.$$

Therefore,

$$|I(DS, \varepsilon)| * \varepsilon * |DS| \leq L * |DS|.$$

If $|I(DS, \varepsilon)| > L * \lceil 1/\varepsilon \rceil$, we obtain

$$|I(DS, \varepsilon)| * \varepsilon * |DS| > L * \lceil 1/\varepsilon \rceil * \varepsilon * |DS| \geq L * 1/\varepsilon * \varepsilon * |DS| = L * |DS|.$$

Hence,

$$|I(DS, \varepsilon)| * \varepsilon * |DS| > L * |DS|.$$

This is in contradiction with the conclusion we have got $|I(DS, \varepsilon)| * \varepsilon * |DS| \leq L * |DS|$, so the theorem is proved. □

According to Theorem 4, the number of items needed to store in *f_list* does not exceed $L * \lceil 1/\varepsilon \rceil$. When support error is ε, we need space $O(L/\varepsilon)$ at most to store all the 1-itemsets with the above approach. So Theorem 3 is proved.

The maximum space cost of *f_list* is given by Theorem 3. The maximum storage space of *f_list* has nothing to do with the number of the transactions and the number of items. It is only related to ε and the average length of transactions. Therefore, even ε is very small, it does not need a huge storage space to store *f_list*.

4.2. The construction of FP-CDS tree

FP-CDS tree is used to store global subfrequent closed itemsets. To ensure that the tree structure is compact and informative, only subfrequent length-1 items will have nodes in the tree, and the tree nodes are arranged in such a way that more frequently occurring nodes will have better chances of node sharing than less frequently occurring ones. In each batch, a new FP-CDS tree is constructed, simultaneously the FP-CDS tree of previous batch are removed. When the FP-CDS tree of the ith batch is constructed, the global subfrequent closed itemsets consist of the new subfrequent closed itemsets (called α-itemsets) from the current batch and the subfrequent closed itemsets (called β-itemsets) from the FP-CDS tree of the i—1th batch. The construction of the FP-CDS tree is similar to that of the FP-tree. Firstly, the root of a tree and a header table are created, and the root is labeled with "null". Then all subfrequent itemsets are inserted into the tree. We obtain the first branch of the tree when the first subfrequent itemset is inserted. The itemset is sorted in order of f_list and only those items in f_list are selected. For the second subfrequent itemset, if it shares a common prefix with the existing branch, its count *f* is combined with the count of each node along the prefix. Otherwise, the itemset is spread into a new branch. The above procedure iterates until all subfrequent itemsets are inserted. To facilitate tree traversal, an item header table is built in which each item points to its first occurrence in the tree.

The combination of the above counts has three cases. (1) α-itemsets. Suppose γ is a item not only in α-itemset, but also in a branch of the FP-CDS tree. If $\gamma_\alpha.f > \gamma_T.f$, let $\gamma_T.f = \gamma_\alpha.f$, where $\gamma_\alpha.f$ and $\gamma_T.f$ denote the count of γ in α and in the FP-CDS tree, respectively. (2) β-itemsets. Suppose η is a item not only in β-itemset, but also in a branch of the FP-CDS tree. If $\eta_\beta.f > \eta_T.fnew$, let $\eta_T.f = \eta_T.f + \eta_\beta.f - \eta_T.fnew$, and $\eta_T.fnew = \eta_\beta.f$, where $\eta_\beta.f$ and $\eta_T.f$ denote the count of η in β and in the FP-CDS tree, respectively, $\eta_T.fnew$ denotes the value of *fnew* of the node and its initial value is 0; (3) θ is defined as the intersection of α-itemset and β-itemset. Suppose λ is a item not only in θ-itemset, but also in a branch of the FP-CDS tree. If $\lambda_\theta.f > \lambda_T.f$, let $\lambda_T.f = \lambda_\theta.f$, where $\lambda_\theta.f$ is defined as the sum of the count of λ in α-itemset and in β-itemset, $\lambda_T.f$ denotes the count of λ in the FP-CDS tree. If $\lambda_\theta.fnew > \lambda_T.fnew$, let $\lambda_T.fnew = \lambda_\theta.fnew$, where $\lambda_\theta.fnew$ is defined as the count of λ in β-itemset, $\lambda_T.fnew$ denotes the value of *fnew* of the node.

Before discussing the proposed algorithm, we use a example to illustrate the construction of the FP-CDS tree.

Example 1. Suppose that α-itemsets is {{f, 4}, {b, 3}, {(f, p), 3}, {(c, f, m, a), 3}, {(c, m), 4}}, β-itemsets is {{(c, f, a, e), 2}, {c, 5}, {a, 3}, {f, 3}}, and f_list = {c, f, a, m, b, p, e}, where {(f, p), 3} indicates itemsets (fp) appeared thrice. A FP-CDS tree is constructed as Fig. 2.

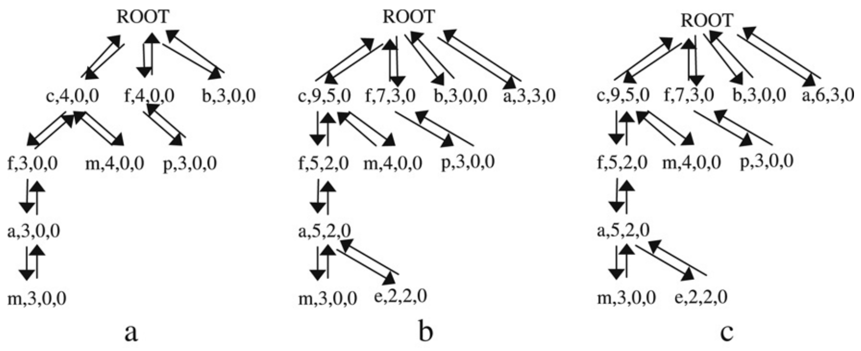


Fig. 2. The construction of FP-CDS tree.

The construction of the FP-CDS tree is described as follows. But, in the following steps, the header table of the FP-CDS tree is omitted for concise presentation.

(1) Every itemset from α-itemsets is sorted in the order of f_list. Those items which don't exist in f_list are ignored, and the rest are inserted into the FP-CDS tree. For example, the FP-CDS tree with respect to two itemsets, {(c, f, m, a), 3} and {(c, m), 4}, constructed by FP-CDS algorithm is described as follows. Itemsets {(c, f, m, a), 3} and {(c, m), 4} are sorted in the order of f_list. We obtain itemsets {(c, f, a, m), 3} and {(c, m), 4}. Next, the items c, f, a, m of itemset {(c, f, a, m), 3} are inserted into the FP-CDS tree. So we obtain a branch of the tree, i.e. {(c, 3, 0, 0), (f, 3, 0, 0), (a, 3, 0, 0), (m, 3, 0, 0)}. When itemset {(c, m), 4} are inserted into the FP-CDS tree, it will share a prefix c with itemset {(c, f, m, a), 3} already in the FP-CDS tree. And the count of c is updated as 4. We also obtain the branch {(c, 4, 0, 0), (m, 4, 0, 0), 4}. After processing α-itemsets, the result is shown in Fig. 2(a).

(2) Every itemset of β-itemsets are sorted in the order of f_list. Those items which don't exist in f_list are ignored, and the rest are inserted into the FP-CDS tree. We consider three itemsets {(c, f, a, e), 2}, {c, 5} and {a, 3}. when itemset {(c, f, a, e), 2} is inserted, it shares common prefix (c, f, a) with the branch {(c, 4, 0, 0), (f, 3, 0, 0), (a, 3, 0, 0), (m, 3, 0, 0)} of the tree. So the count of each node in the prefix is updated. We obtain the branch {(c, 6, 2, 0), (f, 5, 2, 0), (a, 5, 2, 0), (e, 2, 2, 0)}. When itemset {c, 5} is inserted, the node (c, 6, 2, 0) is updated. We obtain updated node (c, 9, 5, 0). When itemset {a, 3} is inserted, the branch {(a, 3, 3, 0)} is obtained. After processing β-itemsets, the FP-CDS tree generated so far is shown in Fig. 2(b).

(3) θ is defined as the intersection of α-itemsets and β-itemset. Every itemset θ is also inserted into FP-CDS tree. The intersection of itemsets {(c, m), 4} and {c, 5} is {c, 9, 5}. When itemset {c, 9, 5} is inserted, the FP-CDS tree does not change. The intersection of itemsets {(c, f, m, a), 3} and {a, 3} is {a, 6, 3}. After inserting itemset {a, 6, 3}, the branch {(a, 3, 3, 0)} is updated. Fig. 2(c) plots the final FP-CDS tree.

```
**Algorithm 2.** Construct the FP-CDS tree of batch $N_i$.
Input: data of batch $N_i$; the FP-CDS tree of batch $N_{i-1}$ (when $i > 1$); support error $\varepsilon$;
    the f_list of batch $N_i$.
Output: the FP-CDS tree of batch $N_i$
(1) Construct the original FP-CDS tree of batch $N_i$ which consists of a header table and a root node.
    The root node is labeled with 'null';
(2) With $\varepsilon$ as the support, call the existing algorithm of mining frequent closed patterns to generate the
new subfrequent closed itemsets from this batch, sort every itemset in the order of f_list, ignore those
items which don't exist in f_list, then insert the rest into the FP-CDS tree of batch $N_i$, and at the same time
obtain set of the itemsets named CFCInew; /*$\alpha$-itemsets*/
(3) if ($i > 1$)
(4) Call function insertconstruct ; /* the subfrequent itemsets from the FP-CDS tree of batch $N_{i-1}$ are
inserted into the FP-CDS tree of batch $N_i$.*/
(5) Delete the FP-CDS tree of batch $N_{i-1}$;
```

Insertconstruct function is described as follows.

```
(1) Obtain the frequent closed itemsets $\beta$-itemsets according to the FP-CDS tree of batch $N_{i-1}$;
(2) for every itemset $\beta_i \in \beta$-itemsets {
(3) if ($\beta_i.f > 1$) {
(4)   sorts every items in the order of f_list, ignore the items which are not included in f_list,
      insert the rest into FP-CDS tree;
(5)      for every itemset $\alpha_j \in$ CFCInew {
(6)          Let $b_{ji} = \alpha_j \cap \beta_i$, insert $b_{ji}$ into the FP-CDS tree; }}} /*Intersection $\theta$*/
```

4.3. *The generation of frequent closed itemsets*

The mining algorithm of frequent itemsets is similar to *insertconstruct* function. The former handles the f of items and the latter handles the *reval* of items. Following are detailed description.

```
Algorithm 3. CDSgrowth.
Input: the FD-CDS tree of the batch $N_i$, support $S$ (Suppose the corresponding support count is SF).
Output: frequent closed itemsets of batch $N_i$.
(1) Take item $e_i$ in the inverted order of header table;
(2) for each $e_i${
(3)   Obtain the pattern set of $e_i$ from the FP-CDS tree, whose patterns are pattern $e_{i1}$, pattern $e_{i2}$,
      pattern $e_{i3}$, respectively ...
(4)   for each $e_{ij}${
(5)     for every $e_i$ in $e_{ij}$, if($e_i.f = e_i.par.f$)$e_i.par.dsgrow = 1$;
(6)   if($e_{ij}.f > SF$ and $e_i.dsgrow == 0$)
(7)     Output pattern $e_{ij}$ and its count; }}
```

Support the support count is 4, according to Fig. 2(c), we obtain frequent closed itemsets {(c, f, a), 5}, {c, 9}, {(c, m), 4}, {f, 7} and {a, 6}.

Theorem 5. *Frequent closed itemsets based on FP-CDS algorithm are complete.*

Proof. Suppose there is a landmark window DSW, we consider three cases.

(1) Suppose X is an infrequent closed itemset in batch N_i. According to Theorem 1, if $f_{N_i}(X)$ is ignored, the correct output of frequent closed itemsets is not affected.

(2) Suppose X is a subfrequent closed itemset in batch N_i, $let W = DSW - N_i$, then

$$f_{DSW}(X) = f_W(X) + f_{N_i}(X), \qquad f_{DSW}(P) = f_W(P) + f_{N_i}(P).$$

If X is not a frequent closed itemset, it must satisfy $\exists P \supset X$ and $f_{DSW}(X) \le f_{DSW}(P)$.
If not $\exists P \supset X$, obviously, X must be the candidate itemset of frequent closed itemsets.
If $\exists P \supset X$, then

$$f_W(X) \ge f_W(P), \qquad f_{N_i}(X) > f_{N_i}(P).$$

So

$$f_{DSW}(X) > f_{DSW}(P).$$

X might become frequent closed itemset later on, so X is the candidate itemset of frequent closed itemsets.

(3) Suppose X is a subfrequent closed itemset in batch N_i, Y is a subfrequent closed itemset in batch N_j ($j \ne i$) and $X \ne Y$. Let

$$Z = X \bigcap Y,$$

then

$$f_{dsw}(Z) > f_{dsw}(X), \qquad f_{dsw}(Z) > f_{dsw}(Y). \quad \square$$

According to definition of frequent closed itemsets, Z is the candidate itemset of frequent closed itemset.

As a result, only the intersection of subfrequent closed itemsets in the different batches and subfrequent closed itemsets in every batch might become frequent closed itemsets in the future. In FP-CDS algorithm, we have fully considered those situations. Therefore, frequent closed itemsets based on FP-CDS algorithm are complete.

When the length of batch is the integer times of $\lceil \frac{1}{\varepsilon} \rceil$ (suppose n times), after scanning batch N_i once, let every 1-itemset in f_list follow $e_j.del = e_j.del - n$. when $e_j.del = 0$, e_j is ignored. Other parts are the same as described above.

One pass scan of data stream on a landmark window can not get the exact solution. According to Theorem 1, if we deliver all itemsets whose approximate frequency is larger than $(S - \varepsilon)|N|$, we will not miss any frequent itemset. However, we may return some itemsets whose frequency is between $(S - \varepsilon)|N|$ and $S|N|$. This is reasonable when ε is small.

5. Experimental evaluation

Our FP-CDS algorithm is written in C++. The data streams adopted in this paper is Customer shopping datasets generated by IBM synthetic data generator. The experiments adopt six datasets: $T7I4D3000K$, $T5I4D3000K$, $T3I4D1000K$, $T5I4D1000K$, $T7I4D1000K$ and $T10I4D1000K$, where T denotes the average length of transactions of datasets, I denotes average length of frequent itemsets, D denotes the total number of transactions. There are $1K$ different items in datasets. The default values for all other parameters of the data generator are used. Suppose support is S, let maximum support error $\varepsilon = 0.1S$. We adopt the improved FP-growth algorithms to mine new subfrequent closed itemsets of batch N_i.

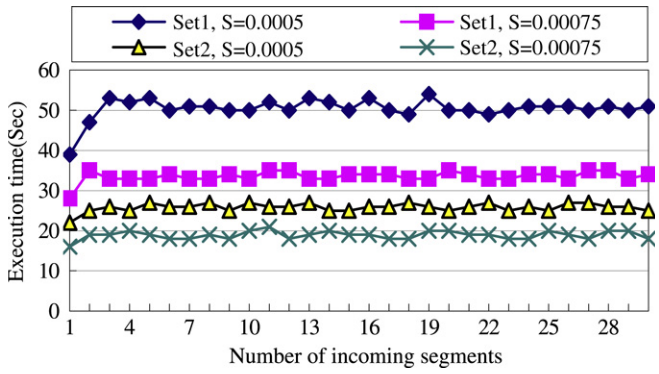


Fig. 3. Running time of each batch.

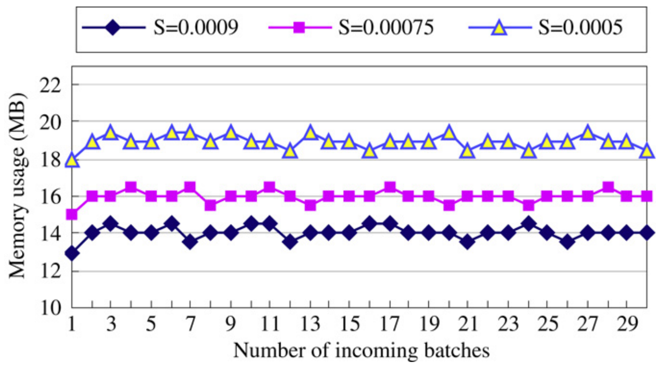


Fig. 4. Maximum memory usage of each batch.

5.1. Performance analysis

Our experiments are performed on a workstation using a 1 GHz Intel Celeron processor, 128 MB of memory. The operating system in use is Redhat9.0. The experiments adopt datasets: $T7I4D3000K$(abbreviated as Set1) and $T5I4D3000K$ (abbreviated as Set2). Support S separately is fixed at 0.0005 and 0.00075, and corresponding ε, respectively, is 0.00005 and 0.000075. Data streams are divided into batches, and every batch contains 50,000 transactions. Experiments mainly study time and space efficiency of the algorithm. In Fig. 3, execution time refers to running time of each batch. The maximum memory usage in the Fig. 4 refers to the maximum memory space which algorithm needs when dealing with every data batch (including temporary memory).

Seen from Fig. 3, as minimum support decreases and average length of transactions increases, running time for every batch gradually grows. The main reason is: in these two kinds of situations, more subfrequent itemsets will be generated and a lot of time will be cost to generate and store these subfrequent itemsets. Moreover, the size of FP-CDS tree also increases, so that we reconstruct and traverse FP-CDS tree also need more time. These have all affected the time efficiency of the algorithm.

From Fig. 4 we can see that, with the decrease of support, the maximum memory requirement of every batch increases gradually. However, when the support is given, stability is gained quickly. The maximum memory requirement of every batch tends to stabilize or grow very slowly. That is to say, it has nothing to do with the size of data sets. Consequently, we can apply this algorithm to long data streams.

5.2. Performance comparison

In [15,16,20], the algorithms focus on mining closed frequent itemsets over a data stream sliding window. But those algorithms can not be used in a landmark window. FP-Stream [4] is a classical frequent patterns mining algorithm over data streams and also is a more popular algorithm at present. FP-Stream can used to mine frequent itemsets from a landmark window. Zhang Xin [14] et al. improve FP-Stream algorithm and propose the FPIL-STREAM algorithm which time efficiency is better than FP-Stream. So we choose FP-Stream and FPIL-STREAM as the contrast to FP-CDS algorithm. The experimental environment and datasets adopted by our experiments are same as the literature [4]: Experimental environment is DELL PC, CPU is Pentium III-933 MHz, Memory is 384 MB, Operating system is Windows 2000. Experimental dataset, respectively, is $T3I4D1000K$ (abbreviated as Set3), $T5I4D1000K$ (abbreviated as Set4), $T7I4D1000K$ (abbreviated as Set5) and $T10I4D1000K$

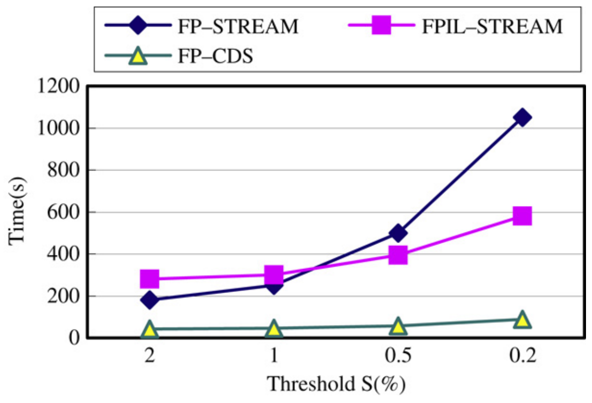


Fig. 5. Running time on different supports.

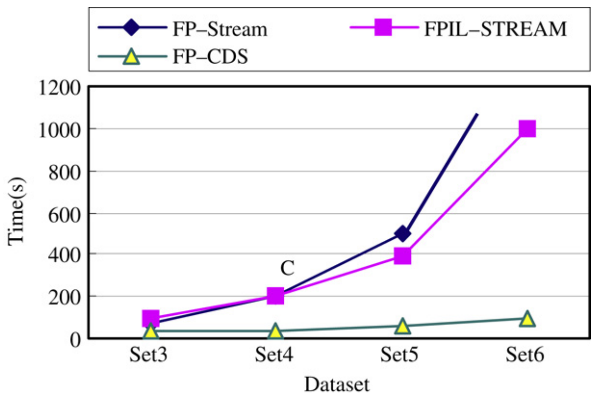


Fig. 6. Running time on different datasets.

(abbreviated as Set6). The incoming data streams are divided into batches, and length of every batch is 100 K. Let support error $\varepsilon = 0.1S$.

We first compare the running time of three algorithms over the same dataset and with different supports. Fig. 5 shows the result of three algorithms on the datasets Set5 (1000 K transaction) and the different supports. The results show that time efficiency of FP-CDS greatly surpasses that of FP-Stream and FPIL-STREAM. Moreover, improvement of algorithm efficiency is more obvious when support is small.

We then compare the running time among three different datasets. Fig. 6 gives results on support $S = 0.005$ and the different datasets. As can be Seen from this figure, compared with FP-Stream and FPIL-STREAM, FP-CDS is especially suitable for frequent itemsets of longer sizes. The response time of FP-CDS is faster than that of FP-Stream and the FPIL-STREAM when average length of transactions is long. When average length of transactions becomes short, because the length of frequent patterns also becomes short, the decrease of the running time of FP-CDS is inconspicuous.

6. Conclusion

Frequent closed itemsets uniquely determine the exact frequency of all itemsets, moreover they can be orders of magnitude smaller than the set of all frequent itemsets and they are often easier to understand and apply in practice. However, mining frequent closed itemsets from a landmark window over data streams is a challenging problem. At present related research reports are not be found. Landmark window is one of most classical windows under data stream environments. FP-CDS can effectively mine the frequent closed itemsets online from a landmark window, and moreover it does not have the pattern detention and can guarantee the error of support not to exceed ε. The FP-CDS tree is adopted to dynamically record the change of the frequent closed itemsets in a landmark window. FP-CDS tree is an effective storage structure of frequent closed itemsets and can be incremental updated. Compared with FP-Stream and FPIL-STREAM, FP-DS has a better time efficiency.

References

[1] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, Discovering frequent closed itemsets for association rules, in: C. Beeri, et al. (Eds.), Proc. of the 7th Int'l. Conf. on Database Theory, Springer-Verlag, Jerusalem, 1999, pp. 398–416.

[2] M.J. Zaki, C.J. Hsiao, CHARM: An efficient algorithm for closed itemset mining, in: R. Grossman, et al. (Eds.), Proc. of the 2nd SIAM Int'l. Conf. on Data Mining, SIAM, Arlington, 2002, pp. 12–28.
[3] J.Q. Liu, X.Y. Sun, Y.T. Zhuang, Y.H. Pan, Mining frequent closed patterns by adaptive pruning, Journal of Software 15 (1) (2004) 94–102.
[4] C. Giannella, J. Han, J. Pei, X. Yan, P.S. Yu, Mining frequent patterns in data streams at multiple time granularities, in: H. Kargupta, A. Joshi, K. Sivakumar, Y. Yesha (Eds.), Next Generation Data Mining, AAAI/MIT, 2003.
[5] G.S. Manku, R. Motwani, Approximate frequency counts over streaming data, in: Proc. of the 28th Int. Conference on Very Large Data Bases, VLDB 2002, August 2002.
[6] R.M. Karp, C.H. Papadimitriou, S. Shenker, A simple algorithm for finding frequent elements in streams and bags, ACM Transactions on Database Systems (2003).
[7] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: Proceedings of 29th International Colloquium on Automata, Languages and Programming, 2002.
[8] J.H. Chang, W.S. Lee, Finding recent frequent itemsets adaptively over online data streams, in: The 9th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining, KDD 03, Washington, DC, August, 2003.
[9] W.G. Teng, M.S. Chen, P.S. Yu, A regression-based temporal pattern mining scheme for data streams, in: Proceedings of the International Conference on Very Large Data Bases, Berlin, Germany, Sept. 2003.
[10] G. Cormode, F. Korn, S. Muthukrishnan, D. Srivastava, Finding hierarchical heavy hitters in data streams, in: The International Conference on Very Large Data Bases, VLDB 2003.
[11] T. Asai, H. Arimura, K. Abe, S. Kawasoe, S. Arikawa, Online algorithms for mining semi-structured data stream, in: The IEEE International Conference Data Mining, ICDM 2002.
[12] G. Cormode, S. Muthukrishnan, What's hot and what's not: Tracking most frequent items dynamically, in: The ACM Symposium on Principles of Database Systems, PODS, 2003.
[13] C. Jin, W. Qian, C. Sha, J.X. Yu, A. Zhou, Dynamically maintaining frequent items over a data stream, in: The Conference on Information and Knowledge Management, CIKM, 2003.
[14] X. Zhang, X.G. Li, D.L. Wang, G. Yu, A high-speed heuristic algorithm for mining frequent patterns in data stream, Journal of Software 16 (12) (2005) 2099–2105.
[15] Y. Chi, H. Wang, P.S. Yu, R.R. Muntz, Moment: Maintaining closed frequent itemsets over a stream sliding window, in: Int'l Conference on Data Mining, November 2004.
[16] N. Jiang, L. Gruenwald, CFI-Stream: Mining closed frequent itemsets in data streams, in: ACM International Conference on Knowledge and Data Discovery, KDD, August 2006.
[17] G. Mao, X. Wu, X. Zhu, G. Chen, C. Liu, Mining maximal frequent itemsets from data streams, Journal of Information Science 33 (3) (2007) 251–262.
[18] C. Lin, D. Chiu, Y. Wu, A. Chen, Mining frequent itemsets from data streams with a time-sensitive sliding window, in: Proceedings of 2005 SIAM International Conference on Data Mining, Newport Beach, CA, USA.
[19] R. Jin, G. Agrawal, An algorithm for in-core frequent itemset mining on streaming data, in: Proceedings of the 5th IEEE International Conference on Data Mining, Houston, TX, USA, 2005.
[20] H.F. Li, C.C. Ho, F.F. Kuo, S.Y. Lee, A new algorithm for maintaining closed frequent itemsets in data streams by incremental updates, in: Proceedings of IEEE International Workshop on Mining Evolving and Streaming Data, IWMESD-2006, to be held in conjunction with ICDM-2006, Hong Kong, 2006.