An Efficient Algorithm for Maintaining Frequent Closed Itemsets over Data Stream

Show-Jane Yen, Yue-Shi Lee, Cheng-Wei Wu, and Chin-Lin Lin

Department of Computer Science and Information Engineering, Ming Chuan University,
5 De Ming Rd., Gwei Shan District, Taoyuan County 333, Taiwan
{sjyen,leeys}@mail.mcu.edu.tw

Abstract. Data mining refers to the process of revealing unknown and potentially useful information from a large database. Frequent itemsets mining is one of the foundational problems in data mining, which is to discover the set of products that purchased frequently together by customers from a transaction database. However, there may be a large number of patterns generated from database, and many of them are redundant. Frequent closed itemset is a well-known condensed representation of frequent itemset, and it provides complete information of frequent itemsets. Extensive studies have been proposed for mining frequent closed itemsets from transaction database, but most of them do not take streaming data into consideration. In this paper, we propose an efficient algorithm for maintaining frequent closed itemsets over data streams. Whenever a transaction is added to database, our approach incrementally updates the information of closed itemsets and outputs updated frequent closed itemsets based on user-specified thresholds. The experimental results show that our approach outperforms previous studies.

Keywords: Data Mining, Data Stream, Frequent Closed Itemsets, Frequent Itemsets.

1 Introduction

Data mining [1, 3] refers to the process of revealing unknown and potentially useful information from a large database. *Mining frequent itemsets* from a transaction database is a fundamental task to several data mining applications. The problem of discovering frequent itemsets within database is stated as follows: Given a large database D which containing N transactions, where each transaction $t \in D$. The finite set of items $I = \{a_1,a_2...,a_M\}$, each transaction t is a list of distinct items $\{i_1,i_2,...,i_m\}$, $i_j \in I$, $1 \le j \le m$. A *k-itemset* is a set of k distinct items $\{i_1,i_2,...,i_k\}$, $i_j \in I$, $1 \le j \le k$, where k is *the length of itemset.* Given an *itemset* X, $SC(X)$ is the *support count of* X and defined as the number of transactions in D that include X. The *support of* X is defined as $SC(X)/N$. An itemset is said to be *frequent* if its support is no less than a user-specified threshold *min_sup (count)* which is called *minimum support (count)*. Otherwise the itemset is said to be *infrequent*. Mining all frequent itemsets from D is equivalent to the discovery of all itemsets having a support no less than *min_sup (count)*. Let Table 1 be the database, and set *min_sup* = 40%. The *minimum support*

B.-C. Chien et al. (Eds.): IEA/AIE 2009, LNAI 5579, pp. 767–776, 2009.

count is 40% * 5 = 2.Then, the frequent itemsets are {A}:4, {B}:3, {C}:4, {D}:2, {AB}:3, {AC}:3, {BC}:2, {CD}:2 and {ABC}:2. The number beside each frequent itemset is its support count.

Table 1. A Transaction Database

TID	Transaction
t_1	C,D
t_2	A,B
t_3	A,B,C
t_4	A,B,C
t_5	A,C,D

Many studies [1, 3] for mining frequent itemsets mainly focused on traditional database. They do not take streaming data into consideration. A data stream is an order sequence of transactions that arrives in a timely order. Mining frequent itemsets over data streams is a fundamental problem in several applications such as network traffic analysis, on-line transaction analysis and other important tasks. However, mining in data streams have posed new challenges. First, data come continuously, unbounded and usually with high speed. To keep all information about itemsets from an entire stream is hard. Second, data distribution in streams usually changes with time. The status of itemsets may be changed (from frequent into infrequent or from infrequent into frequent). Third, most analysts are interested in the most recent patterns. They may expect that the information of current frequent itemsets can be output in real time based on user specified-threshold. Some approaches [2, 4] are proposed for above problems. But they often generate a huge set of frequent itemsets and degrade the mining performance.

One of the solutions to this problem is to mine only the frequent closed itemsets [3]. Mining closed itemset is equivalent to mine only those itemsets having no proper supersets with the same support. Moreover, a closed itemset is said to be frequent if its support is no less than *min_sup*. For the above example, the frequent closed itemsets are {A}:4, {C}:4, {AB}:3, {AC}:3 and {ABC}:2. In this case, {B} is non-closed since its support count is same as its superset {AB}. It implicitly indicates that {B} will not appear in a transaction without {A}. Therefore, non-closed itemsets are regarded as redundant.

In general, the number of frequent closed itemsets is much smaller than the number of frequent itemsets. Therefore, mining only the closed itemsets reduces the mining time and memory space. Besides, complete set of frequent closed itemsets can uniquely determine all frequent itemsets and their exactly support without any information loss [6]. Moreover, frequent closed itemset has been proven more meaningful for analysis [5]. Some approaches [2, 4, 6, 7, 8] are proposed for traditional database, but they do not take streaming data into consideration.

Chi et al. [2] propose an algorithm called *Moment* for mining frequent closed itemsets over data streams. It uses a *CET Tree (Closed Enumerate Tree)* to maintain the main information of itemsets. Each node in CET Tree represents an itemset with different node type. Some nodes in CET Tree are not closed so that there are still

some redundant nodes in CET Tree. *Moment* must maintain huge CET nodes for a frequent closed itemset. Chi et al. indicated that the ratio of CET nodes for a closed itemsets is about 20:1. If there are a large number of frequent closed itemsets, it will consume a lot of memory space. When a new transaction arrives, the node is inserted and updated according to its node type. The exploration of frequent itemsets and node type checking are time consuming.

CFI-Stream is another algorithm for this problem [4]. Only the closed itemsets are maintained in a lexicographical ordered tree which is called *DIU Tree (DIrect Update Tree)*. Each node consists of a closed itemset and its support count. When a new transaction X arrives, CFI-Stream will generate all the subsets of X, and check if each subset Y is closed or not after the transaction arrives. To check whether an itemset Y is closed or not, CFI-Stream may need to search all supersets of Y from *DIU Tree*. It takes a lot of time to generate all the subsets of a new transaction and search their supersets from DIU Tree.

In this paper, we propose an efficient algorithm, called *CloStream*, for maintaining frequent closed itemsets in a data stream. A *Closed Table* and a *Cid List* are used to maintain the information of closed itemsets. All closed itemsets and their support counts are records in *Closed Table*. Each closed itemset has a unique closed itemset identifier, call *cid*. The cids of super closed itemsets for each item are maintained in the Cid List. As a transaction arrives to the database, it incrementally updates closed itemsets and their supports based on previous mining results. Unlike CFI-Stream, CloStream doesn't need to generate subsets of the transaction, and doesn't need to search supersets for each subset. Via the simple intersection of the transaction and certain closed itemsets once. The updated closed itemsets can obtained without multiple scans of whole search spaces. Hence, our approach has better performance than previous approaches.

2 Preliminaries

In this section, we formally define the closed itemsets [6] and describe some properties.

Definition 1 (Closure Operator)
Let T be the subsets of all that transactions in D, $T \subseteq D$, and Y be the subsets of all items appear in D, $Y \subseteq I$. The concept of closed itemset is based on the following two functions f and g:

$$f(T) = \{i \in I \mid \forall t \in T, i \in t\} \quad (1)$$

Function f takes a set of transactions T as an input and returns an itemset included in all transactions belonging to T.

$$g(Y) = \{t \in D \mid \forall i \in Y, i \in t\} \quad (2)$$

Function g takes an itemset Y as an input and returns a set of transactions including Y. A function $C = f \circ g$ is composted by f and g, and is called closure operator [6].

Definition 2 (Closed Itemset)
An itemset X is called a closed itemset if and only if

$$C(X) = f_{\circ}\ g(X) = f(g(X)) = X \tag{3}$$

$C(X)$ is called *the closure of X*. Definition 2 shows that an itemset X is called *closed* if and only if X equals to its *closure* $C(X)$. Otherwise, X is *non-closed*.

For above example, $g(\{B\}) = \{t_2, t_3, t_4\}$ since these transactions are including {B}. Let $T = \{t_2, t_3, t_4\}$, $f(T) = \{AB\}$ since {AB} belongs to each transaction in T. {B} is non-closed since $C(\{B\}) = \{AB\}$. From above discussions, we have the following definitions and properties.

Definition 3 (Frequent Closed Itemset)
An itemset X is called a *frequent closed itemset* if and only if $X = C(X)$ and its support is no less than *min_sup*.

Property 1. If $Y = C(X)$, then $SC(X) = SC(Y)$ [6].

Property 2. If $Y = SC(X)$, then Y is the smallest closed itemset containing X, $X \subseteq Y$. Moreover, $SC(Y) > SC(Z)$, Z is any superset of X, $X \subset Z$ and $Z \neq Y$.

Reason. Since $X \subset Z$, $SC(X) \geq SC(Z)$. If $SC(X) = SC(Z)$, then each transaction containing X is containing Z, the closure of X is Z instead of Y. It is a contradiction with $Y = C(X)$, $Z \neq Y$. Otherwise, if $SC(X) > SC(Z)$, then $SC(Y) > SC(Z)$ since $SC(X) = SC(Y)$ (Property 1).

Property 3. If $SC(X) > SC(Y)$, Y is any superset of X, $Y \supset X$, then $X = C(X)$ [6].

3 CloStream Algorithm

In this section, we present our proposed CloStream algorithm and its structure. Our approach uses two in-memory data structures which are called *Closed Table* and Cid List respectively. An additional hash table is used to put those itemsets need to be updated as transaction arrives.

Closed Table is used to maintain the information of closed itemsets. Each record of Closed Table represents the information of a closed itemset. It consists of three fields: *Cid*, *CI* and *SC*. Each closed itemset was assigned a unique closed identifier, called *cid*. Cid field is utilized to identify closed itemsets. Given a cid, CloStream gets corresponding closed itemsets in CI field. The support counts are stores in SC field. Initially, the value of first record in Closed Table is set to 0. It will be used in our approach. Table 2 illustrates a Closed Table after five transactions (in Table 1) arrives.

Cid List is used to maintain the items and their *cidsets*. It consists of two fields: *Item* field and *cidset* field. The *cidset of an item X* is denoted as *cidset(X)*, it is a set which contains all cids of X's super closed itemsets. The update of Cid List is stated

as follows: As CloStream finds a new closed itemset Y and assigns c as its cid. Then, c will be added into the cidsets of items that are contained by Y. For example, if {AB} is closed, whose cid is assigned as 2, then 2 will be added into cidset(A) and cidset(B) respectively. Table 3 illustrates a Cid List. It maintains the items and their superset cids (in Table 2).

A hash table, called $Temp_A$, is used to put those itemsets need to be updated as transaction arrives. It takes *TI* field as key, and *Closure_Id* field as value. Itemsets S need to be updated are put in TI field. The closure id of S are stored in Closure_Id field. $Temp_A$ was shown in Table 4. In Table 4, the second record is ({B}, 2). The corresponding closed itemset is {AB} in Closed Table (in Table 2). The support count of {B} is same as its closure {AB}.

Table 2. Closed Table after adding t_1, t_2 ,t_3, t_4, t_5

Cid	CI	SC
0	{0}	0
1	{CD}	2
2	{AB}	3
3	{ABC}	2
4	{C}	4
5	{ACD}	1
6	{A}	4
7	{AC}	3

Table 3. Cid List after adding t_1, t_2 ,t_3, t_4, t_5

Item	Cidset
A	{2, 3, 5, 6, 7}
B	{2, 3}
C	{1, 3, 4, 5, 7}
D	{1, 5}

Table 4. The status of $Temp_A$ as t_6 arrives

TI	Closure_Id
{C}	4
{B}	2
{BC}	3

3.1 Maintenance Rules

In this subsection, we discuss the maintenance rules for CloStream. In the following, we assume that t_A is a newly arrived transaction. D_B is the database before adding t_A. $D_U = D_B \cup \{t_A\}$ is an updated database after adding t_A. $SC_{DB}(X)$ and $SC_{DU}(X)$ represent the support counts of itemset X within D_B and D_U respectively. $C_{DB}(X)$ and $C_{DU}(X)$ represent the closure of X within D_B and D_U respectively. The set of closed itemsets in D_B and D_U are denoted as C_{DB} and C_{DU} respectively. The maintenance rules are based on the following lemmas.

Lemma 1. As t_A arrives to D_B, then itemset $t_A \in C_{DU}$.

Lemma 2. As t_A arrives to D_B, if an itemset Y is not a subset of t_A, then the status of Y will not be changed, i.e., $SC_{DB}(Y) = SC_{DU}(Y)$ and $C_{DB}(Y) = C_{DU}(Y)$.

Lemma 3. Suppose that an itemset $S = t_A \cap X$, $S \in C_{DB}$. If $S \neq \varnothing$, then S is a closed itemset in D_U.

3.2 Adding a Transaction

As a transaction with k items, $t_A = \{i_1, i_2, \ldots, i_k\}$, $i_j \in I$, $1 \leq j \leq k$, is added to the database D_B. CloStream consists of two phases. In first phase, CloStream finds all

itemsets need to be updated with their closures in D_B, and puts them into Temp_A. In second phase, CloStream updates their support counts according to Property 2, and updates Closed Table and Cid List.

Phase 1. As t_A arrives, according to Lemma 1, t_A is a closed in D_U. CloStream puts t_A into Temp_A and sets its Closure_Id as 0. Since the closure of t_A is unknown at the beginning. The values of Closure_Id will be updated in the middle of mining process. Then, CloStream intersect t_A with associated closed itemsets to get itemsets need to be updated. The set of cids of associated closed itemsets is defined as $SET(\{t_A\}) = cidset(i_1) \cup ... \cup cidset(i_k)$. CloStream finds itemsets need to be updated by intersection of t_A and closed itemsets whose cids are in the $SET(\{t_A\})$. According to Lemma 3, the results of intersection are closed itemsets in D_U. This process can be performed by Cid List and Closed Table. Suppose that S is the intersection result of t_A and a closed itemset C which cid $i \in SET(\{t_A\})$. If S is not in Temp_A, then put (S, i) into Temp_A. Otherwise, if S is already in Temp_A with its current Closure_Id t, compare $SC_{DB}(C)$ with a closed itemset Q which cid is t in Closed Table. If $SC_{DB}(C)$ is greater than $SC_{DB}(Q)$, then CloStream replaces (S, t) which is in Temp_A with (S, i). The reason is that the closure of S has a support greater than any its superset's support (Property 2, 3). The intersections of t_A with C stops till all cids in $SET(\{t_A\})$ are processed. The purpose of phase 1 is to find itemsets need to be updated and find their closure before t_A arrives.

Table 5. Closed Table after adding t_6

Cid	CItemset	SC
0	0	0
1	{CD}	2
2	{AB}	3
3	{ABC}	2
4	{C}	5
5	{ACD}	1
6	{A}	4
7	{AC}	3
8	{B}	4
9	{BC}	3

Table 6. Cid List after adding t_6

Item	Cidset
A	{2, 3, 5, 6, 7}
B	{2, 3, 8, 9}
C	{1, 3, 4, 5, 7, 9}
D	{1, 5}

Phase 2. CloStream gets itemsets X with their Closure_Id c from Temp_A, and checks that whether X is equal to closed itemset whose cid is c in Closed Table. If X is already in Closed Table with cid c, then X is originally a closed in D_B. In this case, directly increase $SC_{DU}(X)$ by 1. Otherwise, X is a new closed itemset after t_A arrives. In this case, $SC_{DU}(X)$ is equal to the support count of its closure increased by 1. At the same time, CloStream assigns X a new cid n, puts X into Closed Table, and update Cid List. The phase 2 is completed till all records in Temp_A are processed. Finally, a new set of closed itemsets after t_A arrives can be obtained in new Closed Table. All frequent closed itemsets can be output by scanning Closed Table once.

```
01 Procedure CloStream (t_A, CT, CL)
02   Temp_A ← (t_A, 0)
03   SET({t_A}) = cidset(i_1) ∪ ... ∪ cidset(i_k)
04   for each cid i ∈ SET({t_A}) do
05         S ← NULL
06         S ← t_A •CT[i].CI
07         if (S ∈ Temp_A ) then
08               if (CT[i].SC > CT[t].SC) then
09                    replace (S, i) with (S, t) in Temp_A
10         else
11              Temp_A← Temp_A ∪ (S, i)
12   end for
13   for each (X, c) ∈ Temp_A do
14        if (X == CT[c].CI ) then CT[c].SC++
15        else
16             j ← j+1
17             CT ← CT ∪ ( j, X, CT[c].SC+1)
18             for each i_i ∈t_A do cidset(i_i) ← cidset(i_i) ∪ j
19        end if-else
20   end for
21 end Procedure CloStream
```

3.3 A Running Example for CloStream

Let Table 1 be a running example. Before t_1 arrives, $D_B = \varnothing$. The first record of Closed Table is set to (0, 0, 0). Each cidsets in Cid List is set to $\varnothing$. As t_1 = {CD} arrives, $D_U = D_B \cup t_1$. CloStream puts {CD} into $Temp_A$ and sets its Closure_Id to 0. Then, CloStream unions *cidset*(C) and *cidset*(D) to get *SET*({CD}), i.e., *SET*({CD}) = *cidset*(C) ∪ *cidset*(D) = $\varnothing$. Since *SET*({CD}) is empty, t_1 does not need to intersect with any closed itemsets. Therefore phase 1 was completed, CloStream goes to phase 2. In phase 2, CloStream updates itemsets within $Temp_A$ by their Closure_Id. Only ({CD}, 0) in $Temp_A$. CloStream finds a closed itemset whose cid is 0 from Closed Table, *CT*[0] ={0}. Because {0} is not equal to {CD}. {CD} is a new closed itemset after t_1 arrives. Then, CloStream assigns {CD} a new cid which is 1. Then, CloStream determines SC_{DU}(CD), which equals to $SC_{DB}(CT[0])$ increased by 1. Therefore, SC_{DU}(CD) is 1. Finally, CloStream updates Closed Table and Cid List respectively. CloStream inserts (1, {CD}, 1) into Closed Table and inserts 1 into Cid List. Deal with t_2, t_3, t_4 and t_5 in same manners. After transactions in Table 1 are inserted, updated Closed Table and Cid List are shown as Table 3 and Table 4 respectively.

As t_6 = {BC} arrives, CloStream puts {BC} into $Temp_A$, and sets its Closure_Id to 0. Then, *SET*({BC}) = {2, 3} ∪ {1, 3, 4, 5, 7} = {1, 2, 3, 4, 5, 7} (in Table 4). CloStream intersects t_6 with those closed itemsets whose cids belongs to *SET*({BC}). The first is *CT*[1] = {CD} and {BC}∩{CD} = {C}. Put {C} into $Temp_A$, and set its Closure_Id as 1. Deal with next itemset *CT*[2] = {AB}, i.e., {BC}∩{AB} = {B}. Put ({B}, 2) into $Temp_A$. Deal with *CT*[3] = {ABC}, i.e., {BC}∩{ABC}={BC}. However, {BC} is already in $Temp_A$, its current Closure_Id is 0. In this case, CloStream compares $SC_{DB}(CT[0])$ with $SC_{DB}(CT[3])$. As a consequence, $SC_{DB}(CT[3])$

is greater than $SC_{DB}(CT[0])$. According to property 4, CloStream replaces ({BC}, 0) with ({BC}, 3). Deal with rest closed itemsets in same steps. The result of $Temp_A$ was shown in Figure 4.

In phase 2, CloStream updates the support counts of itemsets which are in $Temp_A$. The first record in $Temp_A$ is ({C}, 4). CloStream finds corresponding closed itemset from Closed Table, which is $CT[4]$ = {C}, and increases SC_{DB}({C}) by 1. Then, deal with next record in $Temp_A$, which is ({B}, 2). However, {B} is not in original Closed Table. CloStream assigns 8 as its cid. Then, set SC_{DU}({B}) equals to SC_{DB}({AB}) increased by 1. CloStream deals with last record ({BC},3) in same manner. Finally, updated Closed Table and Cid List are shown as Table 6 and Table 7 respectively. A new set of closed itemsets after t_6 arrives are maintained in the updated Closed Table. All frequent closed itemsets can be found by scanning Closed Table once.

4 Experimental Results

In this section, we compare the performance of CloStream against CFI-Stream [4]. These two algorithms are coded in Java language. All experiments are evaluated on a 1.83 GHz Intel Core 2 Duo Processor with 2 Gigabyte memory, and running on Windows Vista. For performance evaluation, we generate the synthetic dataset by IBM data generator [1], publicly available from IBM Almaden.

Table 7. Parameters of IBM data generator

D	The total number of transactions.
T	The average transaction size.
I	The average maximal potential frequent itemset size.
N	Number of distinct items.

Table 7 describes the parameters of synthetic datasets. Two synthetic datasets T5I2D20K and T5I4D20K are generated with fixed N = 2K (K=1000) and T = 5. In order to simulate the environment of a data stream, transactions are inserted to database one by one in our experiments.

Figure 1 shows the running time and memory requirement respectively when performing addition operation. We take 10K transactions as an original database, and evaluate each execution time and memory usage for adding 1K transactions to updated database respectively.i The addition performance of CloStream outperforms than CFI-Stream. The reason is that CFI-Stream needs to check the status of each itemset in new transaction, and needs to find its supersets from DIU Tree. Assume that a transaction with m items arrives to database, and CFI-Stream needs to search DIU Tree 2^m-1 times. Different from CFI-Stream, CloStream finds associated closed itemsets by a Cid List, and simply intersect them with new transaction to get closed itemsets need to be updated. Hence, CloStream achieves better performance than CFI-Stream.

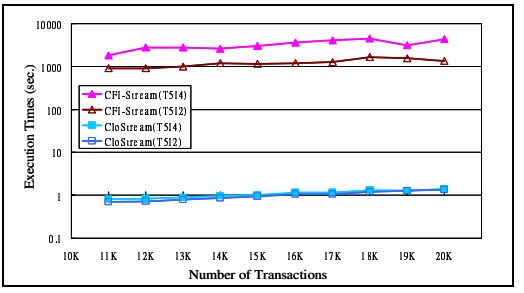

Fig. 1. Execution times after adding transactions

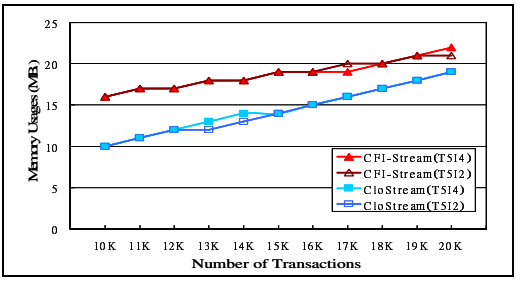

Fig. 2. Memory usages after adding transactions

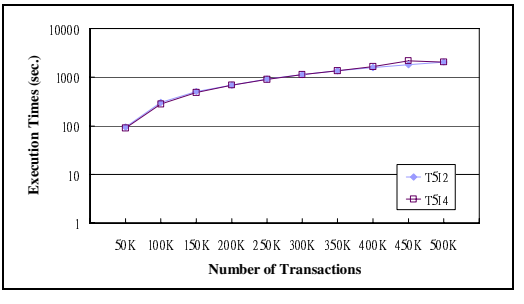

Fig. 3. Scalability on transaction size

Figure 2 indicates that the memory usage of CloStream is lower than CFI-Stream. The reason is that a hash table is attached to improve the execution time of CFI-Stream, and causes higher memory consumption. If CFI-Stream does not attach a hash table, it turns very slow and can not be competed with CloStream.

Figure 3 shows scalability of CloStream. It takes 0K transactions as an original database, and evaluate each execution time for adding 50K new transactions to updated database. Figure 3 only shows the scalability of CloStream, since CFI-Stream is very slow and can not be competed with CloStream. When dataset become larger, CFI-Stream turns to very slow. As shown in Figure 3, CloStream has a good scalability with larger transaction size. Its execution time is increased stably with larger datasets.

5 Conclusions

In this paper, we propose an efficient algorithm, called *CloStream*, for maintaining frequent closed itemsets in data stream. When a transaction arrives to the database, CloStream incrementally updates the closed itemsets without scanning original database. Unlike CFI-Stream [4], CloStream does not need to generate all subsets of the new transaction does not need to search supersets for each subset to determine whether the subset is a closed or not. Extensive experiments are performed to evaluate the efficiency of CloStream and CFI-Stream. In the experiments, CloStream outperforms CFI-Stream over 1000 times, and has good scalability with larger database.

References

1. Agrawal, R., Srikant, R.: Fast Algorithm for Mining Association Rules. In: Proceedings of International Conference on Very Large Data Bases, Santiago, Chile, pp. 487–499 (1994)
2. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In: Proceedings of 2004 IEEE International Conference on Data Mining, Brighton, pp. 59–66 (2004)
3. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, USA, pp. 1–12 (2000)
4. Jiang, N., Gruenwald, L.: CFI-Stream: Mining Closed Frequent Itemsets in Data Streams. In: Proceedings of 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, pp. 592–597 (2006)
5. Ji, L., Tan, K.-L., Tung, A.: Compressed Hierarchical Mining of Frequent Closed Patterns from Dense Data Sets. IEEE Trans. on Knowledge and Data Engineering 19(9), 1175–1187 (2007)
6. Pasquier, N., Bastide, T., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel, pp. 398–416 (1999)
7. Pei, J., Han, J., Mao, R.: Closet: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: Proceedings of the ACM SIGKDD International Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, USA, pp. 21–31 (2000)
8. Wang, J., Han, J., Pei, J.: Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, pp. 236–245 (2003)