An Efficient Algorithm to Mine Online Data Streams

Nan Jiang
School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
nan_jiang@ou.edu

Le Gruenwald
School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
ggruenwald@ou.edu

ABSTRACT

Mining frequent closed itemsets provides complete and condensed information for non-redundant association rules generation. Extensive studies have been done on mining frequent closed itemsets, but they are mainly intended for traditional transaction databases and thus do not take data stream characteristics into consideration. In this paper, we propose a novel approach for mining closed frequent itemsets over data streams. It computes and maintains closed itemsets online and incrementally and can output the current closed frequent itemsets in real time based on users' specified thresholds. Experimental results show that our proposed method is both time and space efficient, has good scalability as the number of transactions processed increases and adapts very rapidly to the change in data streams.

1. INTRODUCTION

Frequent closed itemsets concept was first introduced by Pasquier et al in 1999 [11]. An itemset is frequent if its support is above or equal to a user-defined support threshold. An itemset is closed if none of its proper supersets has the same support as it has (the mathematical definition of a closed itemset is given in Section 2). A closed frequent itemset is an itemset which is both frequent and closed. Usually, the total number of closed frequent itemsets is much smaller than that of frequent itemsets. Furthermore, all frequent itemsets can be derived from closed frequent itemsets because a frequent itemset must be a subset of one (or more) closed frequent itemset, and its support is equal to the maximal support of those closed itemsets that contain the frequent itemset. Therefore mining frequent closed itemsets provides complete and condensed information for frequent pattern analysis. More importantly, association rules extracted from closed sets have been proved to be more meaningful for analysts because all redundancies are discarded [17].

There exist works on closed itemsets mining [9, 12-13, 18], but they are mainly for traditional databases, where multiple scans are needed and whenever new transactions arrive, additional scans must be performed on the updated transaction database; therefore, they are not suitable for data stream mining. A data stream is an ordered sequence transactions that arrives in a timely order. Different from data in traditional static databases, data streams have the following characteristics. First, they are continuous, unbounded, and usually come with high speed. Second, the volume of data streams is large and usually with an open end. Third, the data distribution in streams usually changes with time [6].

As the number of applications on mining data streams grows rapidly, such as web transactions, telephone records, and network flows, much research on how to get frequent patterns in a data stream environment has been conducted. In [2, 7, 10, 16], the authors propose algorithms to find frequent itemsets over the entire history of data streams. In [3, 5, 8], the authors use different sliding window models to find recently frequent itemsets in data streams. These algorithms focus on mining frequent itemsets, instead of closed frequent itemsets, with one scan over entire data streams.

In [4], Chi et al propose the Moment algorithm to mine closed frequent itemsets over a data stream sliding window. The algorithm maintains a dynamically selected set of itemsets which includes four types of nodes: infrequent gateway node, unpromising gateway node, intermediate node, and closed node. For each node, the itemset itself, node type, support and sum of the ids of the transactions in which the itemset occurs (tid_sum) are stored. These selected itemsets form a boundary between closed frequent itemsets and the rest of the itemsets. When a new transaction arrives, it checks the closed frequent itemsets stored in a hash table with its support and tid_sum information to decide its node type according to the node properties and incrementally updates the associated nodes' information. Moment judges the closed itemsets indirectly through node property check and excludes them from the other three types of boundary nodes stored in the data structure. It stores much more information other than the current closed frequent itemsets, which consumes much memory, especially when the support threshold is low. Furthermore, the exploration and node type check are time consuming.

In this study, we propose an algorithm, called CFI-Stream, to directly compute closed itemsets online and incrementally without the help of any support information. Nothing other than closed itemsets is maintained in our derived data structure. When a new transaction arrives, it performs the closure check on the fly; only associated closed itemsets and their support information is incrementally updated. This achieves both time and space efficiency, especially when a dataset contains highly correlated transactions. The current closed frequent itemsets can be output in real time based on any user's specified thresholds. We then conduct simulation experiments using real data sets to evaluate the performance of our proposed algorithm.

The rest of this paper is organized as follows. Section 2 formally defines the concept of closed itemsets and describes the notations to be used throughout the paper. Section 3 presents our proposed CFI-Stream algorithm. The performance evaluation is depicted in Section 4. Finally, Section 5 concludes the paper.

2. PRELIMINARY CONCEPTS

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n elements, called items. A subset $X \subseteq I$ is called an itemset. A k-subset is called a k-itemset. Each transaction t is a set of items in I. Given a set of transactions T, the support of an itemset X is the percentage of transactions that contain X.

Let T and X be subsets of all the transactions and items appearing in a data stream D, respectively. The concept of closed itemset is based on the two following functions, f and g: $f(T) = \{i \in I \mid \forall\ t \in T, i \in t\}$ and $g(X) = \{t \in D \mid \forall\ i \in X, i \in t\}$. Function f returns the set of itemsets included in all transactions belonging to T, while function g returns the set of transactions containing a given itemset X.

Definition 1 An itemset X is said to be closed if and only if $C(X) = f(g(X)) = f \bullet g(X) = X$ where the composite function $C = f \bullet g$ is called a Galois operator or a closure operator [14].

From the above discussion, we can see that a closed itemset X is an itemset whose closure C(X) is equal to itself ($C(X) = X$). The closure checking is to check the closure of an itemset X to see whether or not it is equal to itself, i.e., whether or not it is a closed itemset.

3. THE CFI-STREAM ALGORITHM

In this section, we present our proposed CFI-Stream algorithm and in-memory data structure, called DIrect Update (DIU) tree, to perform the closure checking online over a data stream sliding window. We first give an overview of CFI-Stream. Then, we discuss the conditions that we need to check for closed itemsets and how we check for them when performing addition and deletion operations on the DIU tree. Based on this, we develop an online algorithm to discover and incrementally update closed itemsets.

3.1 Algorithm Overview

When a transaction arrives or leaves the current data stream sliding window, the algorithm checks each itemset in the transaction on the fly and updates the associated closed itemsets' supports. Current closed itemsets are maintained and updated in real time in the DIU tree. The closed frequent itemsets can be output at any time at users' specified thresholds by browsing the DIU tree.

We use a lexicographical ordered DIU tree to maintain the current closed itemsets. Each node in the DIU tree represents a closed itemset. There are k levels in the DIU tree, and each level i stores the closed i-itemsets, where k is the maximum length of the current closed itemsets. Each node in the DIU tree stores a closed itemset, its current support information, and the links to its immediate parent and children nodes. Figure 1 illustrates the DIU tree after the first four transactions arrive. The support of each node is labeled in the upper right corner of the node itself. The figure shows that currently there are 4 closed itemsets C, AB, CD, and ABC in the DIU tree, and their associated supports are 3, 3, 1, and 2.

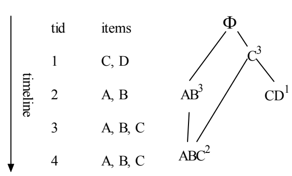

Figure 1. The lexicographical ordered direct update tree

Different from previous closure checking techniques which require multiple scans over data [2, 7, 10, 16], our proposed method performs the closure checking on the fly with only one scan over data streams. It updates only the supports of the associated closed itemsets in the DIU tree online, which reduces the computation time and provides real time updated results. Our algorithm is an incremental algorithm where we check for closed itemsets and update their associated supports based on the previous mining results. This is more efficient as compared with mining approaches that rescan and regenerate all closed itemsets when a new transaction arrives.

Compared with other data stream mining techniques [4, 8, 10], we store only the information of current closed itemsets in the DIU tree, which is a compact and complete representation of all itemsets and their support information. The current closed frequent itemsets can be output in real time based on users' specified thresholds by browsing the DIU tree. Also, our algorithm handles the concept-drifting problem [15] in data streams by storing all current closed itemsets in the DIU tree from which all itemsets and their support information can be incrementally updated. We discuss the update of the DIU tree and the closure checking procedure for addition and deletion operations in Sections 3.2 and 3.3.

3.2 Add a Transaction to the DIU Tree

In this subsection, we discuss the update and maintenance of the DIU tree when a new transaction arrives and its closure check.

3.2.1 Conditions to Check for Closed Itemsets

First, we identify and prove the following conditions in which we need to check whether an itemset is closed or not when a new transaction t arrives in the current sliding window. Condition 1: when the newly arrived transaction t is in the original transaction set, if the largest itemsets X it contains is not currently in the DIU tree, we need to check for all X's subsets Y, which are in the original transaction set to see whether they are closed or not. Condition 2: when the newly arrived transaction t is not in the original transaction set, for each its subset Y, if Y is in the original transaction set, we need to check whether it is closed or not. Below we prove why we only need to check for closed itemsets in the above two conditions. We will use the Lemma 1 and Corollary 1 in our following proofs. The proof of Lemma 1 is given in [9]. We use Lemma 1 to derive Corollary 1.

Lemma 1 Given an itemset X and an item $i \in I$, $g(X) \subseteq g(i) \Leftrightarrow i \in C(X)$.

Corollary 1 Assume $C_T(X)$ is X's closure within transaction set T. If $C_T(X) = X$ and $Y \subset X$ and $C_T(Y) \supset Y$, given an item i, where $i \in C_T(Y)$, $i \notin Y$, then we have $i \in X$ and $C_T(Y) \subseteq X$.

Proof: Because $Y \subset X$, we have $g_T(X) \subseteq g_T(Y)$. If $i \in C_T(Y)$, from Lemma 1, we have $g_T(Y) \subseteq g_T(i)$. Therefore, we have $g_T(X) \subseteq g_T(i)$. Again from Lemma 1, we have $i \in C_T(X)$. So if $i \notin X$, we have $C_T(X) \neq X$, which is a contradiction with the given condition. Therefore, we have $i \in X$. Because $i \in C_T(Y)$, we have $C_T(Y) \subseteq X$. □

When a new transaction t in the data streams arrives, either t is or is not included in the original transaction set. Below we discuss the update and maintenance rules under these two conditions. In the following proof, we assume X and Y are itemsets, T1 is the original set of transactions, T2 is the set of transactions after t arrives, $C_{T1}(X)$ is X's closure within transaction set T1, and $C_{T2}(Y)$ is Y's closure in transaction set T2.

Case 1: When t is in the original transaction set T1

For any new coming transaction t with the largest itemset X that already exists in the original transaction set T1, we have $g_{T1}(X) \neq \phi$. When $g_{T1}(X) \neq \phi$, for any itemset Y, $g_{T1}(Y) = \phi$. If $Y \subset X \Rightarrow g_{T1}(Y) \supset g_{T1}(X) \neq \phi$. This is a contradiction with $g_{T1}(Y) = \phi$. Therefore this condition does not happen. If $Y \not\subset X \Rightarrow g_{T2}(Y) = g_{T1}(Y) = \phi$. Thus, we do not need to discuss cases when $g_{T1}(Y) = \phi$. When $g_{T1}(X) \neq \phi$ and $g_{T1}(Y) \neq \phi$, we examine cases according to the following conditions: $Y \not\subset X$ and $Y \subseteq X$.

Case 1.A: When Y is a subset of X

When Y is a subset of X, $Y \subseteq X$, we divide it into two sub conditions to analyze: X is or is not in the DIU tree.

Case 1.A.1: When X is in the DIU Tree

When X is in the DIU tree, it is a closed itemset, therefore $C_{T1}(X) = X$. We have the following Lemmas 2 and 3. From these two lemmas, we show that if a closed itemset X which already exists in the DIU tree arrives, for any itemset Y, $Y \subseteq X$, if Y is originally closed, it will remain closed; if Y is originally unclosed, Y will remain unclosed, and we only need to update Y's support. Therefore, for most of the existing closed itemsets, we do not need to update the DIU tree structure; we simply update their supports, which consume a small amount of time.

Lemma 2 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) = X$ and $Y \subseteq X$ and $C_{T1}(Y) = Y$, then we have $C_{T2}(Y) = Y$.

Proof: Since $g_{T2}(Y) = g_{T1}(Y) \cup \{X\}$, we have $C_{T2}(Y) = f\bullet g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\})$. Because $Y \subseteq X$, $f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap X = Y \cap X = Y$. □

Lemma 3 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) = X$ and $Y \subset X$ and $C_{T1}(Y) \supset Y$, then we have $C_{T2}(Y) \supset Y$.

Proof: $C_{T2}(Y) = f(g_{T2}(Y)) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap \{X\}$. From Corollary 1, If $C_{T1}(X) = X$, $Y \subset X$, $C_{T1}(Y) \supset Y$. Given an item i, $i \in C_{T1}(Y)$, $i \notin Y$, we have $i \in X$. Therefore, $C_{T2}(Y) = C_{T1}(Y) \cap \{X\} \supseteq Y \cup \{i\} \supset Y$. □

Case 1.A.2: When X is not in the DIU Tree

When X is not in the DIU tree, it is not a closed itemset, therefore $C_{T1}(X) \supset X$. Similarly, we have the following Lemmas 4 and 5. From Lemma 4, we show that if a new closed itemset which is not originally in the DIU tree arrives and if its subsets are already in the DIU tree, they will remain closed, and thus we simply need to update their supports. From Lemma 5, we show that if a new closed itemset which is not originally in the DIU tree arrives, then we need to add it as a new closed itemset to the DIU tree.

Lemma 4 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) \supset X$ and $Y \subset X$ and $C_{T1}(Y) = Y$, then we have $C_{T2}(Y) = Y$.

Lemma 5 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) \supset X$ and $Y = X$, then we have $C_{T2}(Y) = Y= X$.

When $C_{T1}(Y) \supset Y$, and $Y \subset X$, we will perform the closure checking to decide Y's closure, which will be discussed further in Section 3.3.2.

Case 1.B: When Y is not a subset of X

When Y is not a subset of X, $Y \not\subset X$, we have the following Lemma 6. In Lemma 6, we show that if Y is not a subset of X, Y's closure does not change. That is to say that if Y is an unclosed itemset before X arrives, then Y will remain unclosed after X arrives; and, if Y is a closed itemset before X arrives, then Y will remain closed after X arrives. Thus, the DIU tree structure does not need to be updated, and we only need to update Y's support.

Lemma 6 Given $T2 = T1 \cup \{X\}$, if $Y \not\subset X$, then we have $C_{T2}(Y) = C_{T1}(Y)$. Proof: If $Y \not\subset X$, $T2 = T \cup \{X\}$, we have $g_{T2}(Y) = g_{T1}(Y)$. Because $C_{T2}(Y) = f{\bullet}g_{T2}(Y)$, $C_{T1}(Y) = f{\bullet}\, g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. □

Case 2: When t is not in the original transaction set T1

For any new coming transaction t with the largest itemset X that has not already appeared in the original transaction set T1, we have $g_{T1}(X) = \phi$. We discuss two sub cases according to the following conditions: $Y \not\subset X$ and $Y \subseteq X$.

Case 2.A: When Y is a subset of X

When Y is a subset of X, $Y \subseteq X$, we divide it into two sub conditions to discuss: Y exists in the original transaction set T1 or Y does not exist in the original transaction set T1.

Case 2.A.1: When Y is in the original transaction set T1

When Y is already in the original transaction set T1, then $g_{T1}(Y) \neq \phi$. Because $Y \subseteq X$, we have $g_{T2}(Y) = g_{T1}(Y) \cup \{X\}$. Therefore, $C_{T2}(Y) = C_{T1}(Y) \cap \{X\}$. We will perform the closure checking to decide Y's closure, which will be discussed further in Section 3.3.2.

Case 2.A.2: When Y is not in the original transaction set T1

When Y does not exist in the original transaction set T1, then $g_{T1}(Y) = \phi$. We have the following Lemma 7. In this lemma, we prove that when Y is a subset of X, if Y = X, then Y is a closed itemset in transaction set T2; and if $Y \subset X$, then Y is not a closed itemset in transaction set T2.

Lemma 7 Given $T2 = T1 \cup \{X\}$, if $Y = X$, then we have $C_{T2}(Y) = Y$; if $Y \subset X$, then we have $C_{T2}(Y) \subset Y$.

Proof: If Y = X, then $g_{T2}(Y) = g_{T2}(X) = \{X\}$, from the given condition, we know $g_{T1}(X) = \phi$. Therefore after X arrives, we have support(Y) = support(X) =1. Because $g_{T1}(X) = \phi$, all X's supersets' supports = 0; thus Y is a closed itemset after X arrives. If $Y \subset X$, then $g_{T2}(Y) = g_{T2}(X) = \{X\}$, from the given condition, we know $g_{T1}(X) = \phi$. Therefore we have support(Y) = support(X) = 1. Because X is Y's superset, and they have the same support, we have Y as unclosed in transaction set T2. □

Case 2.B: When Y is not a subset of X

When Y is not a subset of X, $Y \not\subset X$, we divide it into two sub conditions to discuss: Y is in the original transaction set T1 or Y is not in the original transaction set T1.

Case 2.B.1: When Y is in the original transaction set T1

If Y is already in the original transaction set T1, then $g_{T1}(Y) \neq \phi$. We have the following Lemma 8. Similar to Lemma 6, in this lemma we prove that when Y is not a subset of X, Y's closure does not change in transaction set T2.

Lemma 8 Given $T2 = T1 \cup \{X\}$, if $Y \not\subset X$, then $C_{T2}(Y) = C_{T1}(Y)$.

Case 2.B.2: When Y is not in the original transaction set T1

If Y is not in the original transaction set, then $g_{T1}(Y) = \phi$. If $Y \not\subset X$, we have $g_{T2}(Y) = g_{T1}(Y) = \phi$, which is meaningless to discuss.

From the above proofs, we can see that when a new transaction arrives, for most cases, the DIU tree structure does not change and we only need to update the associated itemsets' supports, which thus reduces the processing costs. There are only two cases that we need to perform the closure check: 1) when $g_{T1}(X) \neq \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(X) \supset X$, $C_{T1}(Y) \supset Y$, and $Y \subset X$; and 2) when $g_{T1}(X) = \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(Y) \supset Y$, and $Y \subseteq X$. We will discuss how to check for closed itemsets in the following section.

3.2.2 Closure Checking for Addition

The CFI-Stream algorithm checks whether an itemset is closed or not on the fly and incrementally updates the DIU tree based on the previous mining results with one scan of data streams. Below, we discuss the checking procedure when performing the addition operation on the DIU tree. In the following Theorem 1, we show that for any coming unclosed itemset Y, we can always find one and only one closed itemset in the DIU tree equal to Y's closure, such that $X_c = C(Y)$.

Theorem 1 For any itemset Y, if $C(Y) \supset Y$ and $g(Y) \neq \phi$, then we can always find one and only one closed itemset $X_c \in C$, where C is a set of existing closed itemsets that satisfies $C(Y) = X_c$, where $Y \subset X_c$.

Proof: To find this X_c, we first find X_1, such that $X_1 \supset Y$, and support(X_1) = support(Y). According to the definition of closed itemsets, X_1 always exists. If X_1 is not closed, continuing browsing its superset, we can find one X_2, where $X_2 \supset X_1$ and support(X_1) = support(X_2). Continuing this step for $X_1 \ldots X_n$, it is clear that we can find one X_c which is a closed itemset. Otherwise it conflicts with our assumption that $X_1 \ldots X_n$ are not closed itemsets. This X_c is the itemset that satisfies $C(Y) = X_c$.

We also want to prove that there is only one such X_c, where support(X_c) = support(Y) in the existing closed itemsets. Assume there is another X_{c2}, where support(X_{c2}) = support(Y) in the existing closed itemsets. We know that for two different closed itemset X_c, and X_{c2}, $g(X_c) \neq g(X_{c2})$. Because $Y \subset X_c$ and $Y \subset X_{c2}$, we also know that $g(Y) \supseteq g(X_c)$ and $g(Y) \supseteq g(X_{c2})$. Therefore, $g(Y) \supseteq g(X_c) \cup g(X_{c2})$. The only $X_{c2,}$ $X_{c2} \supset Y$ we can find in the existing closed itemsets should satisfy $g(Y) \supseteq g(X_c) \cup g(X_{c2})$, $g(Y) = g(X_c)$. From this we have $g(X_c) \supset g(X_{c2})$, then this X_{c2} can not have the same support as X_c. Because $g(X_c) \neq g(X_{c2})$, we have $g(Y) \supset g(X_c)$, $g(Y) \supset g(X_{c2})$. This conflicts with our assumption, support(X_c) = support(Y); so we could not find X_{c2}, thus X_c is unique.

We now prove $C(Y)= X_c$. For any $i \in C(Y)$, $i \notin Y$, from Lemma 1 we have $g(Y) \subseteq g(i)$. Because $Y \subset X_c$, we have $g(Y) \supseteq g(X_c)$. Therefore, we have $g(i) \supseteq g(X_c)$. From Lemma 1, we have $i \in C(X_c) = X_c$, therefore we have $C(Y) \subseteq X_c$. For any $i \in X_c$, $i \notin Y$, because support(Y) = support(X_c), and from the given conditions we know $Y \subset X_c$, so we have $g(Y) = g(X_c)$. Also because $i \in X_c$, from Lemma 1, we have $g(i) \supseteq g(X_c)= g(Y)$. Therefore, we have $g(i) \supseteq g(Y)$. From Lemma 1 we know $i \in C(Y)$, thus we have $X_c \subseteq C(Y)$. From the above discussion, $C(Y) \subseteq X_c$ and $X_c \subseteq C(Y)$, we have $X_c = C(Y)$. □

From Theorem 1, we know that for any itemset Y, $C(Y) \supset Y$, we can find X_{c0} with a minimum number of items in it and $X_{c0} \supset Y$. For any other $X_{c1} \supset Y$, from the above discussion we know that $g(X_{c0}) \supset g(X_{c1})$. Because $Y \subset X_{c0,}$ then $g(Y) \supseteq g(X_{c0}) \supset g(X_{c1})$. To find $X_c \subseteq C(Y)$, we have $g(X_c) = g(Y)$; only X_{c0} will fulfill this requirement. In this way, C(Y) can be found in the original transaction set T1. Below, we show how we use this C(Y) to check if Y is a closed itemset in transaction set T2 after X arrives.

Corollary 2 Given $T2 = T1 \cup \{X\}$, if $g_{T1}(Y) \neq \phi$, $Y \subseteq X$, $C_{T1}(Y) \supset Y$, $(C_{T1}(Y)/Y) \cap X = \phi$, then we have $C_{T2}(Y) = Y$.

Proof: $C_{T2}(Y) = f \bullet g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap f(\{X\}) = C_{T1}(Y) \cap X = Y$. □

From Corollary 2, we derive a way to check whether Y is closed in transaction T2 or not. If $(C_{T1}(Y)/Y) \cap \{X\} = \phi$, then Y is a closed itemset in T2. We use this condiction to perform the closed itemset checking on the fly when a new transaction in the data streams arrives.

3.3 Delete a Transaction in DIU Tree

In this subsection, we discuss the update and maintenance of the DIU tree for the deletion operation, which occurs when a transaction leaves the sliding window, and its closure check.

3.3.1 Conditions to Check for Closed Itemsets

First, we identify and prove the following condition in which we need to check whether an itemset is closed or not when an old transaction leaves the current sliding window: When the number of the transactions with same itemset of X is equal to zero, if Y is a subset of X, and Y is a closed itemset in the original transaction set, we need to check whether Y is currently closed or not. Below we prove why we only need to check for closed itemsets in the above condition.

When a transaction t, containing a set of items X, is deleted from the current sliding window, the number of transactions with the same itemsets of X is either greater than or equal to zero. Below, we discuss the update and maintenance rules under these two conditions.

In the following proof, we assume X and Y are itemsets, T_1 is the original set of transactions, T_2 is the set of transactions after itemset X leaves, $C_{T1}(X)$ is X's closure within transaction set T_1, and $C_{T2}(Y)$ is Y's closure under transaction set T_2.

Case 1: When the number of the transactions with the same itemsets X is greater than zero

When the number of transactions with the same itemset X is greater than zero, we have the following Lemma 9. From this lemma, we know that Y's closure does not change when the number of transactions with the same itemset X is greater than zero. That is to say that if Y is an unclosed itemset before X leaves, Y will remain unclosed after X leaves; and, if Y is a closed itemset before X leaves, Y will remain closed after X leaves.

Lemma 9 Given $T2 = T1 \setminus \{X\}$, $\{X\} \in T2$, we have $C_{T2}(Y) = C_{T1}(Y)$.

Proof: Because $\{X\} \subset T2$, if $g_{T2}(X) \setminus \{X\} \neq \phi$, we have $f(g_{T2}(X)) = f(g_{T2}(X) \setminus \{X\}) \cap X$, so $C_{T2}(X) = f(g_{T2}(X) \setminus \{X\}) \cap X \subseteq X$. According to the definition, $C_{T2}(X) \supseteq$

X, if $g_{T2}(X) \backslash \{X\} = \phi$, we have $g_{T2}(X) = \{X\}$, $f(g_{T2}(X)) = f(\{X\})$, and $C_{T2}(X) = X$. Therefore, we have $C_{T2}(X) = X$.

For $Y = X$, we have $C_{T2}(Y) = Y$, Y is a closed itemset in transaction set T2.

For $Y \subset X$, because $C_{T2}(X) = X$, $Y \subset X$, for $C_{T2}(Y) = Y$, we have $C_{T2}(Y) \subset X$; for $C_{T2}(Y) \supset Y$, from Corollary 1, we have $C_{T2}(Y) \subset X$. Therefore, $g_{T1}(Y) = g_{T2}(Y) \cup \{X\}$, so $C_{T1}(Y) = C_{T2}(Y) \cap \{X\}$. Because $C_{T2}(Y) \subset X$, $C_{T2}(Y) \cap \{X\} = C_{T2}(Y)$. Therefore, we have $C_{T2}(Y) = C_{T1}(Y)$.

For $Y \not\subset X$, $Y \neq X$, we have $g_{T2}(Y) = g_{T1}(Y)$. Because $C_{T2}(Y) = f\bullet g_{T2}(Y)$, $C_{T1}(Y) = f\bullet g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. □

Case 2: When the number of transactions with the same itemset X is equal to zero

When the number of the transactions with same itemset of X is equal to zero, $\{X\} \notin T2$, we divide this into the following two sub conditions to discuss: Y is not a subset of X and Y is a subset of X.

Case 2.A: When Y is not a subset of X

If Y is not a subset of X, we have the following Lemma 10. In this lemma, we prove that when {X} no longer exists in transaction set T2, Y is not a subset of X and Y's closure does not change in transaction set T2.

Lemma 10 Given $T2 = T1 \setminus \{X\}$, if $\{X\} \notin T2$, $Y \not\subset X$, $Y \neq X$, then $C_{T2}(Y) = C_{T1}(Y)$.

Proof: If $\{X\} \not\subset T2$, $Y \not\subset X$, $Y \neq X$, we have $g_{T2}(Y) = g_{T1}(Y)$. Because $C_{T2}(Y) = f\bullet\ g_{T2}(Y)$, $C_{T1}(Y) = f\bullet\ g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. □

Case 2.B: When Y is a subset of X

If Y is a subset of X, we discuss according to the following sub conditions: Y is a closed itemset in transaction set T1 and Y is not a closed itemset in transaction set T1.

Case 2.B.1: When Y is a closed itemset

When Y is a closed itemset in the transaction set T1, that is to say when $Y \subseteq X$, $C_{T1}(Y) = Y$, we need to perform the closure check, which we will discuss further in Section 3.3.2.

Case 2.B.2: When Y is not a closed itemset

When Y is not a closed itemset in transaction set T1, we have the following Lemma 11. In this lemma, we prove that when Y is a subset of X, $Y \subset X$, Y is not a closed itemset in transaction set T2.

Lemma 11 Given $T2 = T1 \setminus \{X\}$, if $Y \subset X$, $C_{T1}(Y) \subset Y$, then $C_{T2}(Y) \subset Y$.

Proof: Because $g_{T1}(Y) = g_{T2}(Y) \cup \{X\}$, $C_{T1}(Y) = f\bullet g_{T1}(Y) = f(g_{T2}(Y) \cup \{X\}) = C_{T2}(Y) \cap \{X\}$. Because $C_{T1}(Y) \supset Y$, $Y \subset X$, we have $C_{T2}(Y) \cap \{X\} \supset Y$. Therefore, $C_{T2}(Y) \supset Y$. □

From the above discussion, we can see that when an old transaction leaves the current sliding window, for most cases, the DIU tree structure does not change and we need to update only the associated supports, which thus reduces the update costs. There is only one case in which we need to perform the closure check: when $\{X\} \notin T2$, $Y \subseteq X$, and $C_{T1}(Y) = Y$. We will discuss how to check for closed itemsets in the following section.

3.3.2 Closure Checking for Deletion

The CFI-Stream algorithm checks whether an itemset is closed or not on the fly, and incrementally updates the DIU tree based on the previous mining results with one scan of data streams. Below, we discuss the checking procedure for the deletion operation. In the following Theorem 2, we show that for any itemset Y, if $Y \subseteq X$, $C_{T1}(Y) = Y$, $\{X\} \notin T2$, then we can always find $C_{T2}(Y)$ in the original closed itemsets.

Theorem 2 For any itemset Y, if $Y \subseteq X$, $C_{T1}(Y) = Y$, $\{X\} \notin T2$, then $C_{T2}(Y) \in C_{T1}$. That is to say, we can always find $C_{T2}(Y)$ in C_{T1}.

Proof: $C_{T1}[C_{T2}(Y)] = f(g_{T1}(f(g_{T1}(Y) \setminus \{X\})))$. Because $\{X\} \notin T2$, there is one {X} transaction in T1, we have $g_{T1}(Y) \setminus \{X\} \subseteq g_{T1}(f(g_{T1}(Y) \setminus \{X\})) \subseteq g_{T1}(Y)$. So we have either $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y) \setminus \{X\}$ or $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y)$.

In the first case, $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y) \setminus \{X\}$. Because $C_{T1}[C_{T2}(Y)] = f(g_{T1}(Y) \setminus \{X\}) = C_{T2}(Y)$, we have $C_{T2}(Y)$ as a closed itemset in C.

In the second case, $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y)$. Because $C_{T1}[C_{T2}(Y)] = f(g_{T1}(Y)) = C_{T1}(Y) = Y$. So we have $C_{T2}(Y) \subseteq Y$. Also because $Y \subseteq C_{T2}(Y)$, so we have $C_{T2}(Y) = Y$. So $C_{T2}(Y)$ is a closed itemset in C.

Hence, for both cases $C_{T2}(Y) \in C_{T1}$, we definitely can find $C_{T2}(Y)$ in C_{T1}.

In the following Lemma 12, we prove that when Y is a subset of X, $Y \subset X$, $\{Y\} \in T2$. Y is a closed itemset in transaction set T2.

Lemma 12 For any itemset Y, if $Y \subset X$, $\{Y\} \in T2$, we have $C_{T2}(Y) = Y$.

Proof: Because $g_{T2}(Y) = \{Y\} \cup (g_{T2}(Y) \setminus \{Y\})$, we have $C_{T2}(Y) = f(\{Y\}) \cap f(g_{T2}(Y) \setminus \{Y\}) \subseteq Y$. Also because $C_{T2}(Y) \supseteq Y$, we have $C_{T2}(Y) = Y$. □

From the above discussion, we can see that in the condition that we need to perform the closure checking for the deletion operation, if $\{Y\} \in T2$, the Y is closed in the new transaction set T2. Below we show how we perform the closure check when $\{Y\} \notin T2$, and to see if Y is a closed itemset in transaction set T2 after X leaves.

Corollary 3 If $Y \subseteq X$, $\{Y\} \notin T2$, for all $u_1, u_2, \ldots, u_i, \ldots, u_n$ which satisfies $C_{T2}(u_i) = u_i$, $Y \subset u_i$, and $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Proof: First, we prove $C_{T2}(Y) \subseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. Because $Y \subset u_i$, $C_{T2}(u_i) = u_i$ according to Corollary 1, $C_{T2}(Y) \subseteq u_i$. Therefore $C_{T2}(Y) \subseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Next, we prove $C_{T2}(Y) \supseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. For any transaction t, $t \in T2$, $\{Y\} \in t$, because $\{Y\} \notin T2$, so we can find $Z \supset Y$, $\{Z\} \in t$. We know $C_{T2}(Z) \supseteq Z \supset Y$, so $C_{T2}(Z) \in C_{T2}$. Because $u_1, u_2, \ldots, u_i, \ldots, u_n$ are all itemsets in C which includes Y. So we can find $C_{T2}(Z) = u_k$, so $g_{T2}(u_k) = g_{T2}(Z)$. So $t \in g_{T2}(Z) \in g_{T2}(u_k)$. Therefore, we have $g_{T2}(Y) \subseteq g_{T2}(u_1) \cup g_{T2}(u_2) \cup \ldots \cup g_{T2}(u_i) \cup \ldots \cup g_{T2}(u_n)$. So $C_{T2}(Y) \supseteq C_{T2}(u_1) \cap C_{T2}(u_2) \cap \ldots \cap C_{T2}(u_i) \cap \ldots \cap C_{T2}(u_n) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Therefore, we have $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. □

From Corollary 3, we derive a way to check Y's closure: if $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n = Y$, then Y is a closed itemset. We use this rule to perform the closure checking in the CFI-Stream algorithm on the fly when old itemsets leave the current sliding window.

3.4 The Algorithm

Based on our discussions in Sections 3.2 and 3.3, we derive an algorithm to perform online checking for closed itemsets over data streams. The CIF-Stream algorithm performs an addition operation when a new transaction arrives and a deletion operation when an old transaction leaves the current sliding window. By performing addition and deletion operations, the CFI-Stream algorithm checks each itemset in the transaction on the fly and updates the associated closed itemsets' supports. Current closed itemsets are maintained and updated in real time in the DIU tree. The closed frequent itemsets can be output any time at the user's request by traversing the DIU tree.

Algorithm 1 illustrates the addition procedure when an itemset X arrives. It first checks if X is in the current closed itemsets set C. If X is in C, it updatesX's support, and for all X's subsets Y belonging to C, it updates Y's supports (lines 3 to 8). Otherwise, if X is not in C and X has been included by at least one transaction in the original transaction set, it checks whether it is a closed itemset for itself and all its subsets (lines 9 to 36); and it updates the associated supports for all the closed itemsets (lines 37 to 40). If X is a newly arrived closed itemset and does not exist in the DIU tree, the algorithm adds it as a new node to the DIU tree (lines 27 to 31). Otherwise, if X is the added transaction itself, it adds X into the closed itemset (lines 10 to 15); if X is the subset of the added transaction, a closure checking is performed (lines 16 to 24). In the following algorithm description, X and Y represent itemsets, X_s and Y_s represent X's support and Y's support, Len(X) represents the length of the itemset X, which is the number of items in an itemset X, C represents the original closed itemsets in the DIU tree, and C_{new} represents new closed itemsets in the DIU tree after itemset X arrives.

Algorithm 1 CFI-Stream – Addition

```
1:  X_close = true; C_new = ϕ;
2:  procedure Add(X, C, C_new)
3:     if (X ∈ C)
4:        for all (Y ⊆ X and Y ∈C)
5:           Y_s ← support(Y, C) + 1;
6:        end for
7:        if (X_close = true)  return;
8:     else
9:        if (support(X, C) > 0 )
10:          if(C_new = ϕ )
11:             X_0 ← X;
12:             C_new ← X;
13:             X_close = false;
14:             X_s ← support (X, C) + 1;
15:          else
16:             X_c = ϕ;
17:             for all ( K ⊃ X and K ∈ C)
18:                if (len(K)<len(M) then M=K;
19:             end for
20:             X_c ← M;
21:             if ((X_c/X) ∩ X_0 = ϕ and X_c ≠ ϕ )
22:                C_new ← C_new ∪ X;
23:                X_s ← support(X, C) + 1;
24:             end if
25:          end if
26:       else
27:          if (C_new = ϕ ) then
28:             X_0 ← X;
29:             C_new ← X;
30:             X_s = 1;
31:          end if
32:       end if
33:    end if
34:    for all (m ⊂ X and Len(m) = Len(X)-1
35:          call Add(m, C, C_new);
36:    end for
37:    if (X = X_0)
38:       C ← C ∪ C_new;
39:       support(X, C) = X_s;
40:    end if
41: end procedure
```

Algorithm 2 illustrates the procedure to perform the deletion operation when an itemset X leaves the current sliding window. CFI-Stream first checks if X is in the current closed itemsets set C and its count is greater or equal to two; if so, it updates X's support and X's subsets' support belonging to C (lines 3 to 6). Otherwise, it checks the itemset X and all its subsets which are in the current closed itemset C to see whether they are still closed itemsets (lines 8 to 26) and updates the support for all its subsets that are in the current closed itemsets (lines 28 to 29). If the subset Y exists in transaction, Y should keep closed (lines 11 to 13). Otherwise a closure checking for the subset Y is performed (lines 14 to 22). In the following figure, $C_{obsolete}$ represents the itemsets that are no longer closed after transaction {X} leaves.

Algorithm 2 CFI-Stream – Deletion

```
1:  C_obsolete = ϕ;
2:  procedure Delete (X, C, C_obsolete)
3:     if (count(X) ≥ 2) then
4:        for all (Y ⊆ X and Y ∈C)
5:           Y_s ← support(Y, C) – 1;
6:        end for
7:     else
8:           length = Len(X);
9:           for all (len≥1)
10:             for all ( Y ⊆ X and Y ∈C and Len(Y) = length)
11:                if (count(Y) ≥2) then
12:                   Y_s ← support(Y, C) – 1;
13:                else
14:                   M = I;
15:                   for all ( U ⊃ Y and U ∈C)
16:                      M = M ∩ U;
17:                   end for
18:                   if (M = Y) then
19:                      Y_s ← support(Y, C) – 1;
20:                   else
21:                      C_obsolete= C_obsolete ∪ Y;
22:                   end if
23:                end if
24:             end for
25:             length = length-1;
26:          end for
27:    end if
28:    C ← C \ C_obsolete
29:    support(Y, C) = Y_s;
30: end procedure
```

The time complexity of CFI-Stream depends on the sliding window size w and the number of closed itemsets in the DIU tree |C|. The space complexity of CFI-Stream depends on the number of closed itemsets in the DIU tree. In the best case, there is only 1 closed itemset in the current sliding window, according to lemma 5, we only need to perform 1 add operation, both the time and space complexity is O(1). In the worst case, all the transactions in the current sliding window need to perform closure checking through traversing the DIU tree, the time complexity is $O(w \bullet |C|)$ and the space complexity is $O(|C|)$. However, in most cases, as we discussed in section 3.2 and 3.3, the closure checking is unnecessary, even in the case when closure checking is performed, most of the time only part of the DIU tree need to be traversed which is associated to the itemset being added or deleted. Therefore, on average the time complexity is far less than $O(w \bullet |C|)$.

4. PERFORMANCE EVALUATION

We compare our algorithm with Moment [4], which is the state-of-the-art algorithm to mine closed itemsets in data streams. For performance evaluation, two synthetic datasets T10.I6.D100K and T5.I4.D100K-AB are used. Each dataset is generated by the same method as described in [1], where the three numbers of each dataset denote the average transaction size (T), the average maximal potential frequent itemset size (I) and the total number of transactions (D), respectively. In all experiments, the transactions of each dataset are looked up one by one in sequence to simulate the environment of an online data stream. All our experiments were done on a 1.10 GHz Intel Pentium IV PC with 736 MB main memory, running on the RedHat Linux 8.0 operating system. The closed frequent itemsets generated by Moment and CFI-Stream are the same, which confirms the accuracy of the proposed algorithm.

Figure 2 shows the average processing time for Moment and CFI-Stream over the 100 sliding windows under different minimum supports for the dataset T10.I6.D100K. As the minimum support decreases, the running time for Moment increases, since the number of closed frequent itemsets and the boundary nodes increases. For CFI-Stream, the running time is independent of the support information since it discovers and maintains all closed itemsets in the DIU tree. As the number of closed itemsets that exists in the DIU tree increases, they do not need to be reprocessed; only their supports need to be updated incrementally, therefore it needs less processing time per transaction. Also we can see from Figure 2 that CFI-Stream runs much faster than Moment when the support threshold is relatively low. This is because the number of boundary nodes stored in the data structure of Moment increases when the support threshold drops; as the number of nodes to be processed and checked for node property increase, execution time increases. When the support threshold is relatively high, these two algorithms have comparable running time. Moment runs a little faster than the CFI-Stream as the threshold increases. This is because as the threshold creases, the number of the boundary nodes in Moment decreases, while the CFI-Stream processes the same number of all the closed itemsets independent of support information. This is especially beneficial when users have different specified support thresholds in their online queries.

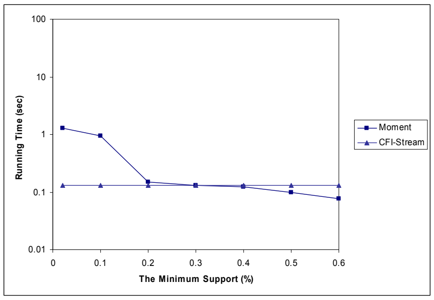

Figure 2. Runtime performance (T10.I6.D100K)

Figure 3 shows the memory usage in terms of the maximum number of itemsets of Moment and CFI-Stream for the dataset T10.I6.D100K. The memory usage for Moment increases when the minimum support decreases. This is because the number of itemsets it keeps track of increases. The memory usage remains almost the same when the support changes in CFI-Stream. This is because CFI-Stream stores all closed itemsets in the DIU tree independently of the support information. The overall memory usage is proportional to the number of closed itemsets in the dataset. Also we can see from the figure that the CFI-Stream algorithm consumes much less memory space than the Moment when the support threshold is low. This is because when the user defined support threshold is small, the number of nodes it maintains in the memory increases dramatically, which includes all the infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes. As the support threshold increases, the memory usage of Moment drops. These two algorithms consume almost the same amount of memory space; Moment takes slightly a smaller amount of memory space than CFI-Stream. This is because CFI-Stream stores all closed itemsets in the DIU tree so that the frequent closed itemsets can be output based on any user-specified thresholds in real time. We can see that CFI-Stream is especially efficient for dense datasets in which the ratio between the number of frequent closed itemsets and the corresponding number of frequent itemsets is large.

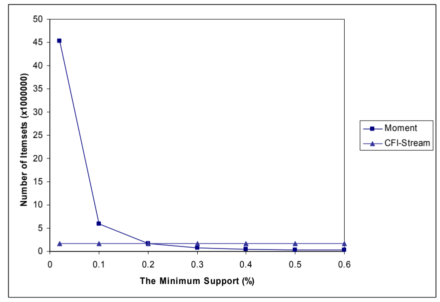

Figure 3. Memory usage (T10.I6.D100K)

Figure 4 shows the adaptability of CFI-Stream to the change in data streams. In this experiment, the dataset T5.I4.D100K-AB is used. The dataset is composed of two consecutive subparts. The first part is a set of 50,000 transactions generated by an item set A, while the second part is a set of 50,000 transactions generated by an item set B. There are no common items in the item sets A and B. In order to illustrate how rapidly the online closure checking method can adapt to the change in data streams, we use the coverage rate CR(X) proposed by Chang et al in [2]. It denotes the ratio of closed frequent itemsets introduced by an item set X in all closed frequent itemsets as follows:

$$CR(X) = \frac{\#\text{ of closed frequent itemsets induced by an item set X}}{|R|} \times 100(\%)$$

where |R| denotes the total number of closed frequent itemsets in a data stream. From Figure 4 we can see that, in the first 50,000 transactions which are generated by an item set A, all the closed frequent itemsets are introduced by the item set A, therefore the coverage rate CR(A) is a hundred percentage, while the coverage rate CR(B) is zero percentage. It reverse in the second 50,000 transactions which are generaged by an item set B. From this figure we can see that CFI-Stream adapts very rapidly to the change of a data stream. This is because the closed itemsets stored in the DIU tree is a complete and condensed representation of all the datasets' information in the data streams. This is favorable when processing high concept-drifting data streams.

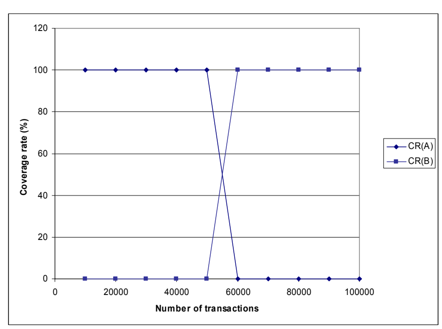

Figure 4. Coverage rate (T5.I4.D100K-AB)

5. CONCLUSIONS

In this paper we proposed a novel algorithm, CFI-Stream, to discover and maintain closed frequent itemsets in the current data stream sliding window. The algorithm offers an incremental method to check and maintain closed itemsets online. All closed frequent itemsets in data streams can be output in real time based on users' specified thresholds. Our performance studies show that this algorithm is able to mine data streams online with both time and space efficiency independent of support information, and it can adapt to the concept-drifting in data streams. Experimental results show that our method can achieve better performance than a representation algorithm for the state-of-the-art approaches in terms of both time and space overhead, especially when the minimum support is low, and the dataset is dense. In the future, we plan to extend our proposed algorithm to different data stream applications.

6. ACKNOWLEDGMENTS

This work is partially supported by (while serving at) NSF, the NASA grant No. NNG05GA30G, and the DoD-OSU grant. We thank Dr. Yun Chi at the University of California for providing us the Moment algorithm source code.

7. REFERENCES

[1] Rakesh Agrawal, Ramakrishnan Srikant; Fast Algorithms for Mining Association Rules; *Int'l Conf. on Very Large Databases*; September 1994.

[2] Joong Hyuk Chang, Won Suk Lee, Aoying Zhou; Finding Recent Frequent Itemsets Adaptively over Online Data Streams; *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*; August 2003.

[3] Joong Hyuk Chang, Won Suk Lee; A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams; *Journal of Information Science and Engineering*; July 2004.

[4] Yun Chi, Haixun Wang , Philip S. Yu , Richard R.; Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window; *IEEE Int'l Conf. on Data Mining*; November 2004.

[5] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, Philip S. Yu; Mining Frequent Patterns in Data Streams at Multiple Time Granularities; *Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT*; 2003.

[6] Sudipto Guha, Nick Koudas, Kyuseok Shim; Data Streams and Histograms; *ACM Symposium on Theory of Computing*; 2001.

[7] Hua-Fu Li, Suh-Yin Lee, and Man-Kwan Shan; An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams; *Int'l Workshop on Knowledge Discovery in Data Streams*; Sept. 2004.

[8] Chih-Hsiang Lin, Ding-Ying Chiu, Yi-Hung Wu, Arbee L. P. Chen; Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window; *SIAM Int'l Conf. on Data Mining*; April 2005.

[9] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego; Fast and Memory Efficient Mining of Frequent Closed Itemsets; *Knowledge and Data Engineering, IEEE Transactions*; January 2006.

[10] Gurmeet Singh Manku, Rajeev Motwani; Approximate Frequency Counts over Data Streams; *Int'l Conf. on Very Large Databases*; 2002.

[11] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal; Discovering frequent closed itemsets for association rules; *ICDT*;1999.

[12] J. Pei, J. Han, and R. Mao; Closet: An efficient algorithm for mining frequent closed itemsets; *SIGMOD International Workshop on Data Mining and Knowledge Discovery*, May 2000.

[13] J. Pei, J. Han, and J. Wang; Closet+: Searching for the best strategies for mining frequent closed itemsets; *In SIGKDD*, August 2003.

[14] Rafik Taouil, Nicolas Pasquier, Yves Bastide, Lotfi Lakhal; Mining Bases for Association Rules Using Closed Sets; *International Conference on Data Engineering*; 2000.

[15] Haixun Wang, Wei Fan, Philip S. Yu, Jiawei Han; Mining Concept-Drifting Data Streams using Ensemble Classifiers; *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*; August 2003.

[16] Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Aoying Zhou; False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams; *Int'l Conf. on Very Large Databases*;2004.

[17] Mohammed J. Zaki; Generating non-redundant association rules; *ACM SIGKDD international conference on Knowledge discovery and data mining*; 2000.

[18] Mohammed J. Zaki and C. J. Hsiao; Charm: An efficient algorithm for closed itemsets mining. *In 2nd SIAM International Conference on Data Mining*; April 2002.