The Nucleus **50**, No. 3 (2013) 229-240

The Nucleus
A Quarterly Scientific Journal of Pakistan Atomic Energy Commission
NCLEAM, ISSN 0029-5698

MINING TOP-K FREQUENT CLOSED ITEMSETS IN DATA STREAMS USING SLIDING WINDOW

*Z. REHMAN and M. SHAHBAZ

Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, Pakistan

(Received May 06, 2013 and accepted in revised form September 02, 2013)

Frequent itemset mining has become a popular research area in data mining community since the last few years. There are two main technical hitches while finding frequent itemsets. First, to provide an appropriate minimum support value to start and user need to tune this minimum support value by running the algorithm again and again. Secondly, generated frequent itemsets are mostly numerous and as a result a number of association rules generated are also very large in numbers. Applications dealing with streaming environment need to process the data received at high rate, therefore, finding frequent itemsets in data streams becomes complex. In this paper, we present an algorithm to mine top-k frequent closed itemsets using sliding window approach from streaming data. We developed a single-pass algorithm to find frequent closed itemsets of length between user's defined minimum and maximum-length. To improve the performance of algorithm and to avoid rescanning of data, we have transformed data into bitmap based tree data structure.

Keyword: Data stream mining, Closed frequent itemsets, Approximate mining, Sliding window

1. Introduction

An unbounded sequence of transactions arriving at high speed is called data stream for examples data streams are generated while network monitoring and analysis, network intrusion detection, web click stream mining, financial applications, telecommunication call records, retail store transactions and many more. Data mining community is focusing to process such dynamic and continuously changing data. There are some inherent problems being faced by researchers while dealing with data streams. Firstly, memory is limited and we are unable to store all incoming data due to continuity and high data rate. Secondly, applications need to look at data only once and it is not feasible to scan data multiple times to detect patterns because of the limited speed and possible limited processing power. Thirdly, mining of incoming data streams should be performed at very fast rate i.e. before the arrival of new transaction. Fourthly, mining results and analytical outputs should be available within a small time frame after the data is being recorded and received for analysis. Finally, error rate of stream processing should be reduced to minimum possible level. Due to these inherent characteristics or problems, traditional data mining algorithms are not suitable to be applied directly on streaming data [1]. There is a need to modify and extend traditional algorithms to deal with data streams. Therefore, researchers from databases and data mining communities have focused on the issues and are continuously trying to improve data analysis strategies in streaming environment. Researcher have to face the challenge to optimize the accuracy and efficiency therefore algorithms developed for streaming data normally perform more efficiently while sacrificing some accuracy and producing some approximate results [2].

Frequent itemsets mining on streaming data is being explored extensively by the researchers recently [3-8]. Association rule is one of the data mining techniques commonly used to extract hidden correlation among the data [9]. These rules are normally generated from the frequent itemsets and reflect hidden information and relation in the data. Association rules can be defined as, if there is a set of transactions and each transaction ***T*** contains some items ***I*** then association is presented as under:

$$A \Rightarrow B \mid A \& B \in I$$

A set containing items from ***I*** is called itemset so ***A*** and ***B*** are itemsets in the above association rule. These rules reflect that if some relation exists or is being observed then there is always a chance that same incident will occur again. For example 94% times an event ***A*** is detected, a virus attack ***B*** occurs in the system. In traditional association rule mining two inherent problems exist. Firstly, tuning

* Corresponding author :xahoor@gmail.com

of minimum support value is a difficult task because we don't want to miss any low occurring important rules and/or itemsets and having a cap on the total number of generated rules. Setting and tuning minimum threshold value in real environment is a difficult task and need a lot of experience and expertise. Furthermore, in cases when someone is only interested to find 30 most frequent itemsets will be constrained to tune *min-supp* value. Therefore, it is needed to bypass this constraint of minimum support adjustment both in traditional and data streaming environments. Secondly, a lot of rules are being generated due to increased number of frequent itemsets [10-11]. In order to reduce number of frequent itemsets, "closure property" is considered that amazingly decreases number of frequent itemsets without losing important information. The idea of frequent closed itemsets was introduced by Pasquier et al.[12].Closed itemset also cover its subsets with the same support and there is no need to keep those subsets that have support equal or less than their superset support. In a case where a frequent itemset is like $(a_1, a_2 \ldots\ldots a_n)$ then frequent subitemsets will be equal to$2^n - 1$. Closed itemset mining only generate one itemset $(a_1, a_2 \ldots\ldots a_n)$ if all of its sub itemsets support is less than or equal to its support. Thus closed itemset mining reduces number of frequent itemsets resulting less number of association rule generation and improving better resource utilization in computing environment. It is also beneficial for real environment implementation and reduces complexity of the problem.

Most of the algorithms developed to mine closed frequent itemsets use *min-supp* threshold [3, 13, and 14].Wang et al. [10] proposed a technique to find top-k closed frequent itemsets without using *min-supp* constraint. In their approach, itemsets of minimum length *l* were extracted and only top-k become frequent closed itemsets using FP-tree pruning and traversing technique. Traditional data mining or specifically frequent itemset mining using apriori algorithm scans data multiple times to discover associations. In streaming data environment the problem of mining frequent itemset becomes more complex because data streams are continuously arriving at a high pace. Due to unavailability of data for multiple scans because of the time and processing speed constraints it is needed to develop one-pass algorithm to extract useful information from data streams. Also because of continuous and rapid data arrival current frequent itemsets might become infrequent or infrequent itemsets become frequent in the next or any subsequent time slot. Keeping all data streams in memory is practically impossible because of its huge volume and its limited usage in future. Therefore, frequent itemset mining in data streams need novel algorithms to overcome these problems and constraints.

To find frequent itemsets in continuous and high volume streaming data, many efforts have been made [3,8,14-19]. As it is almost impossible to keep all data streams in memory therefore only current data from some time-interval can be used. To use most current data, window based models have been proposed [20-27]. There are three different time-window models used for mining frequent itemsets namely landmark model, tilted-time window model and sliding window model [28]. *Landmark model* consider data from some specific landmark (time) to the current time and perform mining tasks only on this block of data. *Tilted-time window model* also known as damped model is a variation of landmark model. This model considers data from start of stream to the current time but assign some weights on the bases of their arrival time. Current or most recent data is assigned a larger weight value and this weight goes on decreasing while moving towards older data. In contrast to landmark and time-tilted model, *sliding window based model* considers the most recent fixed-sized block of data. Size of window in sliding window model can be determined on the bases of fixed number of transactions (transaction-sensitive) or from some time marker to current time (time-sensitive). Sliding window model utilize data available in the current window for mining purposes. When a window expires, a new window of transactions takes place of expired window to provide current most data for mining.

Researchers from data mining community are inspired from the performance and compactness of FP-tree as it scans the data for only two times to extract frequent itemsets present in the data. There are many variations of FP-tree algorithm for streaming data which scans the data for only once to discover the frequent itemsets [22,26,29-32]. Most of these algorithms find approximate results and compactness of FP-tree is also lost. Tanbeer et al [26] made an effort to develop an algorithm to find exact results in the current window without approximation and also developed a strategy to achieve compactness with one scan of transactional data. A data structure Compact Pattern Stream tree (CPS-tree) was developed to efficiently insert newly arriving data into the CPS-tree and similarly deleting the old data from the

CPS-tree. Due to this data structure, overall performance for finding frequent itemsets in data streams improves with exactness. Two basic problems i.e. tuning of *min-supp* and large number of frequent itemset generation are still there to address in this algorithm. To overcome *min-supp* overhead and to fix number of frequent itemsets generation, top-k technique has been proposed [4, 10, 33].

The rest of paper is organized as follows. In section 2, problem of top-k frequent itemset mining in data streams is explained. The same section also covers the proposed techniques and algorithms developed to find solution of top-k frequent itemsets mining in streaming data. Algorithm for mining top-k closed frequent itemsets and other concepts for data streams are presented in section 3. Section 4 explains the proposed algorithm and methodology with a scenario example to visualize the performance and execution of algorithm. Section 5 describes experimentation and results. The last section will cover the conclusions and future work related to this research.

2. Related Work and Preliminaries

In this portion of the paper, we will describe and explain key terms, problem definitions and previous contribution in this area in a more formal way.

2.1 Related Work

In cases when we are not interested to find all frequent itemsets, it is to find all the frequent itemsets and also it required extra efforts to find minimum threshold value which requires some level of expertise and multiple runs of algorithm on the same data to tune this threshold. To overcome these bottlenecks, the concept of Top-k frequent itemsets emerged. In Top-k, we find only k itemsets having maximum support value thus avoiding defining minimum support value. Fu et al.[34] developed an algorithm to mine top-k frequent itemsets without defining the support threshold value for the first time. Cheung and Fu et al.[4]developed an algorithm called Build-Once and Mine-Once (BOMO), based on FP-tree that mines the K most interesting itemsets. This algorithm dynamically updates the threshold value to find K most interesting itemsets. All these algorithms need multiple scans of database to find top-k frequent itemsets while in the streaming environment it is not optimal to scan database again and again. In order to handle the streaming data mining we need to develop single-pass algorithms to find frequent itemsets.

Leung et al.[22] proposed a method based on FP-Tree and developed a tree structure DSTree. This algorithm uses sliding window approach to find frequent itemsets and each batch of transactions in the current window is maintained in a prefix tree. Each node of the tree represents an item and sorted in a canonical order. After expiry of a batch (pane), one of the oldest batches is removed from the tree to create space for the newly arriving batch. Li et al.[35] developed an algorithm called MFI-TransSW based on Apriori algorithm that mines frequent itemsets in the current window. They used bit strings to represent a transaction and to represent the presence of an item in the transaction is shown by ‘1’ while its absence is represented by ‘0’. Left shift operation is performed on the bit string to remove oldest transaction to create space for the newly arriving transaction. As MFI-TransSW is based on Apriori, therefore it generates candidate itemsets and performs testing to mine frequent itemsets and in turn creating problems like scalability and performance in the streaming environment. Tanbeer et al.[26] designed an algorithm that is based on prefix tree called CPS-Tree. This algorithm works like DSTree [22] algorithm but there is an extra functionality of dynamic reconstruction of tree. Reconstruction of tree improves memory utilization and performance of mining frequent itemsets. The prefix tree is monitored continuously and when there are enough changes in the items order in the tree then reconstruction process begins. Lin et al. [36] proposed a method to divide window of transaction into batches (or panes) to find frequent itemsets over the sliding window. Similarly Mozafari et al. [25] developed a method in which sliding window is incorporated and each window is divided into different panes to mine frequent itemsets locally in each pane and then considered for advance analysis in the entire window. Mahmood Deypir et al.[37] proposed a list based method LDS that uses vertical layout of sliding window to mine frequent items in data streams. Three lists are used to continuously monitor the newly arriving stream of transactions and adjusted accordingly to keep current state of frequent itemsets in the active window. This leads to an efficient mining process because current states of lists are incorporating recent window layout and requires less conversion time.

Some efforts have been made to find special type of frequent itemsets in the data streams. Chi et al. [38] proposed an algorithm Moment to find closed frequent itemsets. Similarly Cheng et al. [39] developed IncMine algorithm to find approximate frequent closed itemsets in a sliding window fashion. Li et al. [40] proposed an algorithm called estMax to find non-derivable frequent itemsets in the data streams. Concept drift based frequent pattern mining in streaming data is proposed by Koh et al. [41]. This algorithm keeps on checking transaction streams to detect any concept shift and once any concept shift is found, it will start frequent itemsets mining process. Wang et al.[10]proposed an algorithm TFP that finds top-k frequent closed itemsets not less than user defined min-l size based on FP-tree. Firstly, this algorithm is for static databases and not suitable for streaming environment where data is continuously changing and secondly, in applications where transaction size is large enough this algorithm will be finding frequent itemsets of available maximum size e.g. if itemsets of size 100 is available and min-l value is supposed to be 4 then it will be finding all frequent itemsets more than or equal to size 4. This will be really limiting the performance in the streaming databases. Therefore, a new approach to find top-k closed frequent itemsets over sliding window is proposed. This algorithm finds top-k closed frequent itemsets of sizes between some minimum and maximum length without defining minimum support threshold. Our algorithm is based on prefix tree structure and a dynamic approach is defined to increase support value to find top-k frequent itemsets.

2.2 Preliminaries

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items in some time interval. Suppose that *X* is a non-empty itemset of literals/items from *I*. *l* is the length of itemset *X* reflecting its size in terms of items $i_j \varepsilon I$.

A transaction is represented by a tuple <*tid, X*>, where *tid* is the identifier of transaction and *X* is itemset. A transaction is represented as $T = \{tid, (i_1, i_2, \ldots, i_j)\}$.

T_{DB} is a database of transactions. An itemset *X* is a part of a transaction $T = \{tid, (i_1, i_2, \ldots, i_j)\}$ iff $X \subseteq (i_1, i_2, \ldots, i_j)$. A continuous and unbounded sequence of these transactions is called transaction database as give below:

$$T_{DB} = \{T_1, T_2, \ldots\ldots, T_k\}$$

Where T_1 is the first and T_k is the most recent transaction in the transaction database.

Definition 1. Data stream is a collection of transaction arriving continuously $T_{DB} = \{T_1, T_2, \ldots\ldots, T_k\}$ and $SW_{DB} = \{W_1, W_2, \ldots\ldots, W_k\}$ is a set of windows. W_1 is the first window while W_k is the most recent windows. Each transaction sensitive sliding window SW = { P1 , P2 } is a window that consists of two panes P_1 and P_2 and each pane is represented as shown below:

$$\begin{Bmatrix} P1 = \{Ti+1, Ti+2, Ti+3\} \\ P2 = \{Ti+4, Ti+5, Ti+6\} \end{Bmatrix} \forall\, i = 0, 3, 6, 9, \ldots$$

This window will slide after the arrival of each new pane P that consists of next three (3) transactions. To create room for newly receiving pane P, first pane P_1 will be removed and P_2 will be shifted to the position P_1 so that new pane can be inserted into the P_2 position. Although the sliding of window is after each pane but actually the size of window is the sum of both panes i.e. P_1 and P_2.

$$S = \sum_{i=1}^{2} Pi$$

$Sup(X)w_i$ represents support of an itemset *X* in the window W_i i.e. current window. $Sup(X)w_i$ is the count of transactions containing *X* in the current window W_i.

Definition 2. If $Sup(X)w_i$ is above or equal to some predefined threshold value then itemset *X* is called frequent itemset (*FI*) .

Definition 3. An itemset *X* is called closed itemset if there is no proper subset *Y* of *X* that has support equal to support of *X* i.e. if $Y \subseteq X$ and support of *Y* is less than that of *X* then itemset *Y* will not be in *FI*. Similarly *top-k closed itemset* between some minimum length *min-l* and maximum length *max-l* finds only *k* closed itemsets in the range of *min-l* and *max-l*.

Mining top-k frequent itemsets is relatively a new idea and has not been extensively used for stream mining because of its complexity and requirement for real environment. Hua-Fu li et al. [27] and Wong et al. [42] used top-k frequent itemsets mining in data streams. To minimize number of frequent itemsets, minimum length technique proposed by Wang et al. [10] and maximum length of itemset technique was developed by Tsai et al. [11]. While finding frequent itemsets with length above some *min-length*, some important information might lose. Similarly in the cases where *max-length* for frequent itemsets is defined and set to some large value then large number of frequent itemsets will

Table 1. Transaction database T_{DB}.

TID	Items	Sorted-Items
1	a, b, d, e, f, g, j	b, d, a, f, j, e, g
2	a, b, c, d, f, h, i	b, d, a, c, f, h, i
3	b, d, e, f, g, i, j	b, d, f, i, j, e, g
4	a, c, e, h, i	a, c, h, i, e
5	a, b, c, d, h, j	b, d, a, c, h, j
6	b, c, d, f, g, h, I, j	b, d, c, f, h, i, j, g
7	a, b, c, d, e, f, g	
8	b, d, e, g, i, j	
9	b, c, d, f, i, j	

W_1

be generated. To overcome this problem of large number of frequent itemsets generation, lower and upper marker technique is used to manage number of frequent itemsets generation for those applications where only itemsets between some minimum and maximum sizes are required. Our focus in this research is to develop a tree structure to mine top-k frequent itemsets from streaming data and to develop a strategy to perform correct insertion and deletion of transactional data in an efficient way. There is no specific research to solve the addressed issues for data streams without minimum support threshold.

We have a transactional database T_{DB}, sliding window database SW_{DB} where each window consist of two panes i.e. SW = { P1 , P2 } and K is some integer representing the number of frequent itemsets required to be mined. Required task of this algorithm is to find top-k closed frequent itemsets of size more than *min-l* and less than *max-l* from the T_{DB} over the sliding window *SW* in an efficient manner.

Table 1 shows transaction ID, transaction items and sorted transactions on the bases of support count for window W_1. Let us consider that we are supposed to find top-3 frequent closed itemsets between some *min-l* = 2 and *max-l* = 4. To sort each transaction, frequency count of each item is counted in the entire transaction database T_{DB} and then order of items in each transaction is rearranged in frequency descending order as shown in the third column of Table 1. Closed frequent itemsets between *min-l* = 2 and *max-l* = 4 in first window W_1 of this transaction database T_{DB} are: bd = 5, bda = 3, bdac = 2

As stated in the previous section that different efforts have been made to find frequent itemsets in both static and streaming data, it is quite clear that FP-growth based algorithms are more efficient but problem is how to fit that algorithm to find top-k frequent closed itemsets in the streaming environment.

3. Development of Mining Strategy

In this section, we introduce a complete method to develop an efficient algorithm to find top-k closed frequent itemsets in the streaming data using the sliding window approach. The methodology consists of three major phases i.e. scanning of initial transaction database, initialization of panes in the window and then sorting transactions in frequency descending orders in terms of items. Once the transactions are sorted in frequency descending order, an efficient structure development based on FP-tree to store necessary information about the transactions called FPS-tree with sliding window approach that deals with the removal of the oldest pane of transactions and insertion of the newly arriving batch of transactions. Finally the top-k closed frequent itemsets mining phase starts that dynamically adjust support to find top-k frequent itemsets between some *min-l* and *max-l*.

3.1 Bit-vector Representation

Once the initial batch of transaction is received, a bit-vector is generated for each item in the transaction to process these transactions in an efficient manner both at current and next phases of the algorithm. Once the initialization of window is complete, next phase of FPS-tree data structure generation is started. Details of tree-based structure are in the next section. Finally, the sliding window phase is executed to delete the oldest pane of transactions from the generated tree upon the arrival of new batch of transactions and then insertions of new batch took place.

Table 2. Bit-vector representation and sorted order of transactions.

Item	Bit-vectors	Support	Sorted	Sorted-window
a	110,110	4	b	b, d, a, f, j, e, g
b	111,011	5	d	b, d, a, c, f, h, i
c	010,111	4	a	b, d, f, i, j, e, g
d	111,011	5	c	a, c, h, i, e
e	101,100	3	f	b, d, a, c, h, j
f	111,001	4	h	b, d, c, f, h, i, j, g
g	101,001	3	i	
h	010,111	4	j	
i	011,101	4	e	
j	101,011	4	g	

Li et al.[13] used bit-vector to represent items in the transactional data streams and stored all necessary information in the sliding window. This representation helps to reduce memory utilization and improves efficiency to process items in the sliding window environment. The process of bit-vector representation is described by example as follows.

In each pane of a window, presence of some specific item x is represented by 1 while absence is represented by 0. For example, transaction database shown in Table 1, bit-vectors for item a and b in both panes i.e. P_1 and P_2 (in the entire window W_1) are BV(a) = 110,110 and BV(b) = 111,011 respectively. It is clear from the transaction database that item 'a' is present in T_1, T_2, T_4, T_5, while it is absent in the transactions T_3 and T_6. Therefore presence is represented by 1 and absence by 0 in the current window W_1 and similarly the item b is present in transactions T_1, T_2, T_3, T_5, T_6 and absent in only one transaction T_4. Bit-vector representation of transactions in Table 1 is shown in Table 2.

Once we have converted transactions into bit-vector representation, it becomes fairly easy to sort transactions in frequency descending order. Before inserting transactions into FP-tree based structure, sorting of transactions in frequency descending order really helps to improve the process of finding top-k frequent itemsets.

FP-growth algorithm need to scan database twice to build a compact tree in frequency descending order as discussed by Han et al.[43]. During the first scan, it counts the frequency of each item in the database and produces a list of items in frequency descending order. In the second scan of database it generates a FP-tree that contains of frequent itemsets in a compact way so that frequent itemsets generation can be achieved in an efficient manner. In streaming data environment, it is difficult to process transactions in an efficient way so that resources can be made available for the continuously arriving transactions. Therefore, it is needed to design algorithms that need only one scan of transactions to extract frequent itemsets. In this approach, we transform incoming transactions into a bit-vector to count frequency of each item and then generate transactions in frequency descending order as shown in column *sorted-window* of Table 2. On the bases of *sorted-window* column, transactions are inserted into the FP-tree structure for the first window of transactions. Each window is further decomposed into 2 panes i.e. P_1 and P_2 and when a new batch (pane) of transaction is received, window will be sliding to update its current database. There is information for each transaction of each pane in the FP-tree therefore it is easier to update items information when a window expires and slides forward i.e. older pane will be simply removed from the FP-tree to create space for the upcoming pane and each item pane information in the FP-tree is updated

3.2 FPS-tree Construction

The construction of FPS-tree to find top-k closed frequent itemsets between *min-l* and *max-l* consists of two phases. In first phase it inserts sorted transactions into the tree structure. Second phase starts after the expiry of the first window. When window expires, items of the oldest pane are removed from the FPS-tree and if the remaining branches are in frequency descending order according to the new sorted window then it will keep those branches (unchanged) otherwise those branches are adjusted according to the new sort order of transactions.

Construction of FPS-tree is explained with the help of data shown in Table 2. Let us consider that each window W consists of 2 panes i.e. P_1 and P_2 and each pane contains 3 transactions therefore, there are 6 transactions in each window W. On the bases of support column, items in the transactions are sorted as shown in sorted-item column of Table 2 and then sorted transactions are generated. Each sorted transaction is inserted into the FPS-tree as a branch and if any of the already existing branches contains prefix of new transaction then relevant bit is set to 1 for all shared items in the prefix path and the remaining suffix of the new transaction is inserted by initiating a sub-branch. Relevant bits in all other branches are set to 0 as shown in the Figure 1.

Initially the construction of FPS-tree starts while the entire window W_1 is received i.e. both P_1 and P_2 panes have been received. After the preprocessing of received transactions, transactions in the sorted-window column are inserted into the FPS-tree. Figure 2 shows the structure of FPS-tree after the insertion of both P_1 and P_2 panes i.e. window W_1.

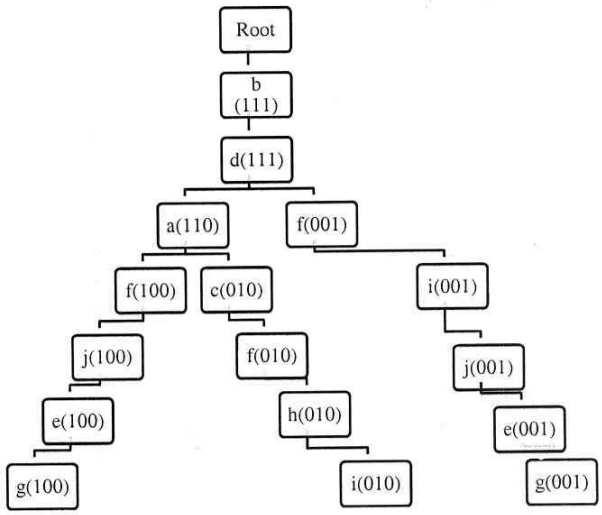

Figure 1. FPS-tree after insertion of pane P_1.

Each node in the tree contains information that in which transaction it is present and which transaction it is missing or absent e.g. **a:**$_{110,010}$ reveals that a is found in transaction T_1, T_2, T_5 and not found in transactions T_3, T_4,and T_6 with prefix of bd as shown in Figure 2. First three bits represent pane P_1 and last three bits represent pane P_2.

Algorithm: Construction Process of FPS-tree

Input:Streaming_data, Size_window SW, Size_pane P, Sorted_windowW

Output: Sorted_tree FPS-tree for current set of transaction

Begin

1. T is a tree with null initialization;
2. W ← Ø is the window status at start;
3. Sort_order ← Frequency_descending;
   // Processing first window
4. While (w ≠ SW) do
5. Call insert_transactions (T_p);
6. W = W + 1;
7. End While;
   // Processing transactions at each window slide
8. Repeat
9. Remove the pane P_1 from the tree T;
   // Removal of oldest pane to create space
10. Call insert_transactions (T_p);//Insert process
11. End;

End

// Pane insertion method
Insert_transactions (Trans)
Begin

1. P ← Ø
2. While (p ≠ pane_size) do
3. Retrieve the transactions from the current window turn by turn;
4. Insert the retrieved transactions into tree T;
5. P = P + 1;
6. End While

End

3.3 Deletion of Items and Restructuring of FPS-tree

On the expiry of window W, first pane P_1that contains transactions T_1, T_2 and T_3 will be removed from the FPS-tree to create space for the newly arriving pane of transactions i.e. T_7, T_8 and T_9. To start the deletion phase, left-most leaf-node is processed first e.g. g in our case as shown in Figure 2. The bit-victor values of leaf-nodes for pane P_1 are checked and set to 0 if it is/are already set to 1 and then the same bits in the entire prefix path are also set to 0. The nodes in the pane P_1, where all the bits are set to 0, are removed and at the same time shift pane P_2 to the position of pane P_1 to create space for the next incoming pane. In the cases where after the removal of pane P_1, if all the bits for the pane P_2 are already set to 0 then all those nodes are also removed from the FPS-tree

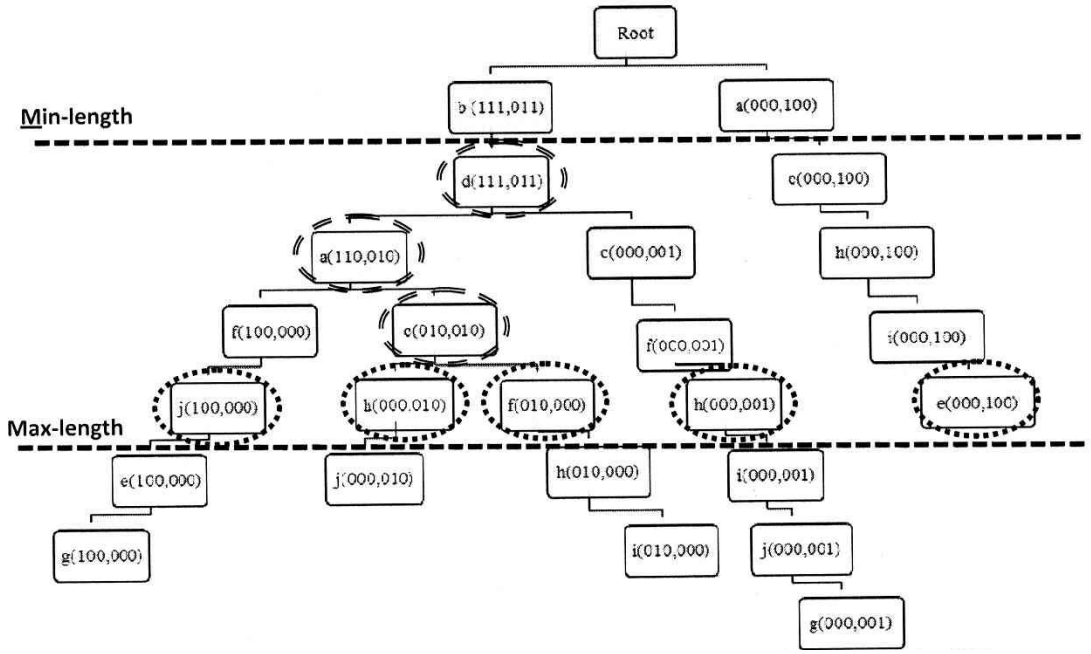

Figure 2. FPS-tree after insertion of both panes P_1 and P_2 (Complete window W_1).

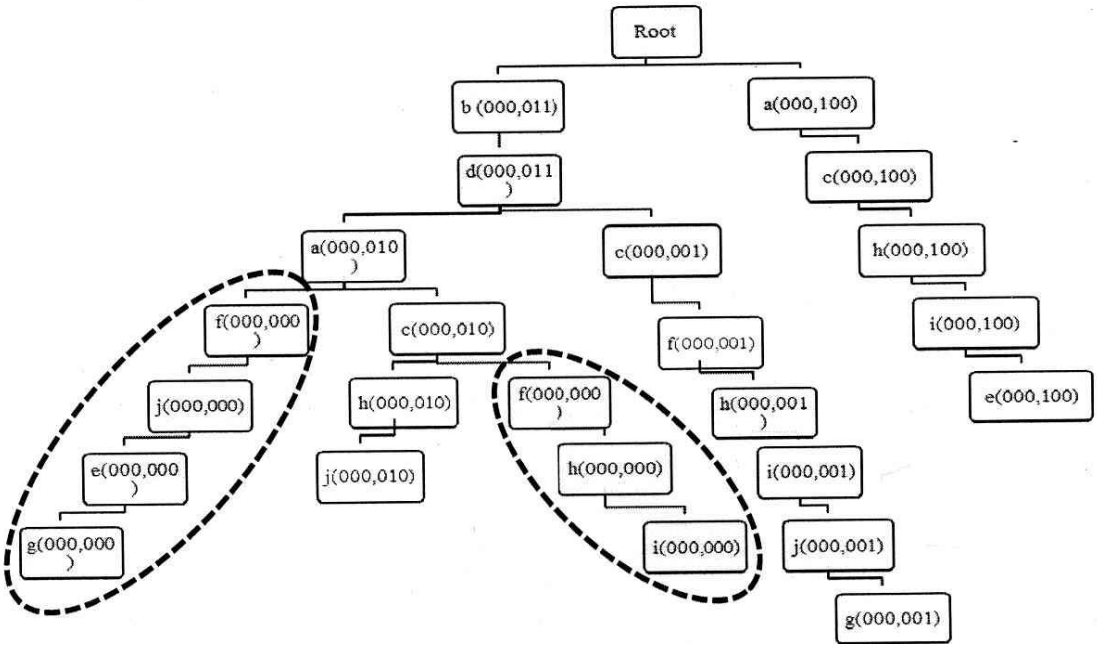

Figure 3. Branches marked for removal after expiry of window W_1.

e.g. in the path b, d, a, f, j, e, g, all bits of pane P_2 in the nodes g, e, j and f are already 0 therefore, after the removal of pane P_1, these nodes are removed from the FPS-tree as shown in Figure 2. Figure 3 shows the braches that have all bits set to zero after the expiry of pane P_1 and are removed from the FPS-tree. After the removal of pane P_1, new FPS-tree structure is shown in Figure 4a and it can accept new incoming pane to store.

Table 3. Bit-vector representation and new sorted order of transactions after window slide.

Item	Bit-vectors after removal	Bit-vectors after insertion	Support	Sorted item	Sorted-window
a	110	110,100	3	b	c, i, a, e, h
b	011	011,111	5	c	b, c, d, j, a, h
c	111	111,101	5	d	b, c, d, i, j, f, g, h
d	011	011,111	5	i	b, c, d, a, e, f, g
e	100	100,110	3	j	b, d, i, j, e, g
f	001	001,101	3	a	b, c, d, i, j, f
g	001	001,110	3	e	
h	111	111,000	3	f	
i	101	101,011	4	g	
j	011	011,011	4	h	

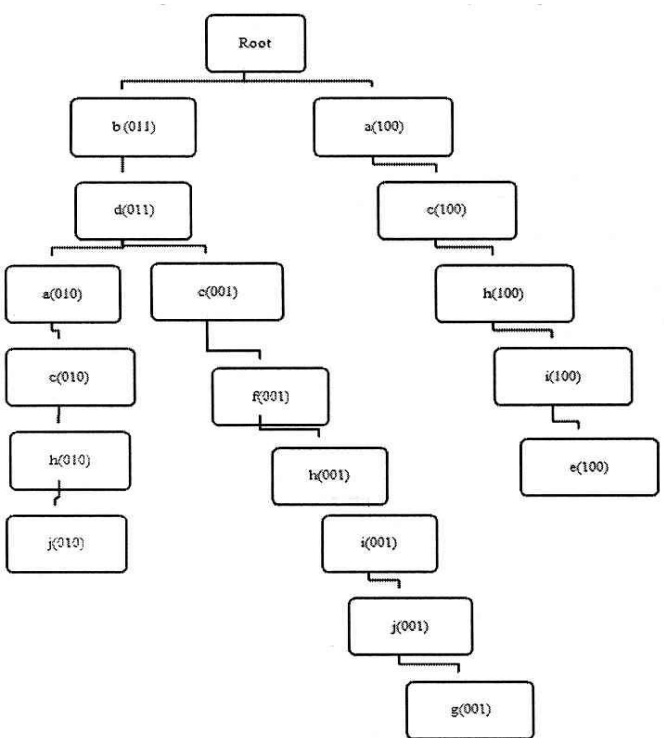

Figure. 4a. FPS-tree after the removal of pane P_1.

Table 3 shows the bit-vectors for each item after the removal of pane P_1 and similarly the bit-vectors after the receiving of new pane P_2. Support count for each item is updated after the pane P_2 is processed and new sort order of items is generated as depicted in the Table 3. According to the new sort order of items, sorted-window of transactions is generated in frequency descending order. Once all these values have been generated, all branches in the FPS-tree are checked to verify whether those are in frequency descending order or not on the bases of new sort order of items. In case, a branch is already according to the new sorted list is remained unchanged otherwise reordering of that branch is initiated to update the FPS-tree. For example the branch b, d, a, c, h, j is not according the new sort order of items therefore it is removed from the FPS-tree and reinserted as per new frequency descending order i.e. b, c, d, j, a, h, as shown in Figure 4b. The updated FPS-tree according to the new sort order is shown in Figure 4b.

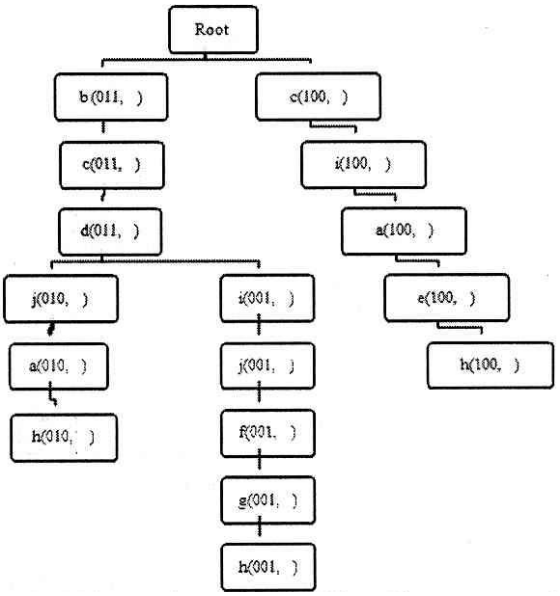

Figure. 4b. FPS-tree after removal of P_1 and in new sort order

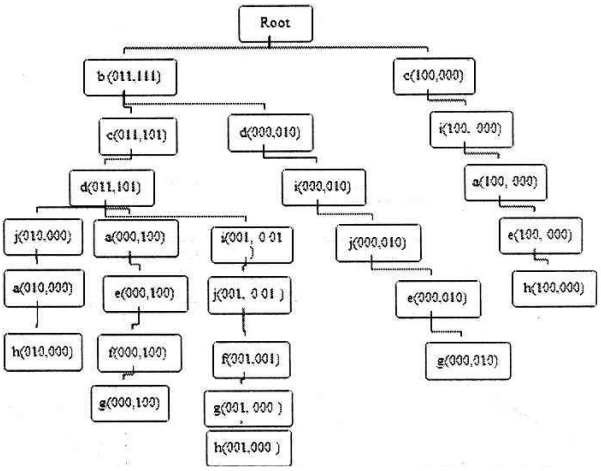

Figure 5. Tree after insertion of new pane and slide of window.

After the completion of restructuring phase of FPS-tree, the insertion phase begins for newly arrived pane P_2. During the insertion process, existing FPS-tree is checked for prefix path for each transaction and if there is any path then it is shared and relevant bits are set to 1 to increase frequency count of those nodes and the remaining suffix path is added as a sub-branch and similarly relevant bits are set to 1 as well. For example while processing the first transaction (b, c, d, a, e, f, g) of pane P_2, the prefix path b, c, d is already there in the FPS-tree therefore it is shared and relevant bits of b, c and d are set to 1 and the remaining suffix i.e. a, e, f and g is added as a sub-branch and relevant bits are also set to 1 as shown in the Figure 5.

3.4 Finding Top-k Closed Frequent Itemsets

To find top-k closed frequent itemsets, it is most important to explain closed frequent itemsets and closed-node. Closed itemset is already defined and explained in section 2.2, therefore, here we need to introduce closed-node only. A node in the FPS-tree is called a closed-node if there is no child with the same support e.g. if X is a parent-node with support of 4 and Y is the child of X with support count of 3 then X is called closed-node but if support of Y is also 4 then X will not be a closed-node.

As the efficiency and performance of any algorithm is directly dependent on the facts that how efficient are the sub-processes of that algorithm therefore to get better performance and efficiency, we developed a strategy to find closed-nodes at early stages. Once we achieve the sorted-window as shown in both Table 2 and Table 3, insertion phase of transactions in FPS-tree is started. Closed-node finding process is integrated with this insertion phase as follows. When first transaction from the sorted-window is inserted into the FPS-tree, the leaf-node is marked as closed-node. An entry of frequency count with itemset is created, as shown in the Table 4, if leaf node is greater in length from *min-l* and less than *max-l*. Similarly, while inserting the next transaction into the FPS-tree, it is checked that whether there is any prefix path that can be shared. If some prefix path is already in the FPS-tree for new transaction, that is being inserted then corresponding bits of shared prefix path are set to 1 and remaining suffix is added as sub-branch of existing one. The node at which sub-branch is created is also marked as closed-node along with the leaf-node (closed-node) and entry in the Table 4 is created if this closed node is between *min-l* and *max-l*. To visualize the process of finding closed-node, Figure 3 depicts that all closed-nodes are represented by double circles. Table 4 represents the closed-itemsets while Table 5 represents same closed-itemsets in the sorted form using merge sort.

We are interested to find top-k itemsets, therefore itemset with maximum support count is found from the closed nodes first. Once the closed itemset with maximum support is calculated, we decrease the count of k by one and next itemset is searched accordingly. This process continues until the value of k becomes zero. This process of

finding top-k reduces number of comparisons and overall efficiency is improved.

Table 4. Closed node count between min-l and max-l.

Support count	Closed-itemsets
5	b, d
3	b, d, a
1	b, d, a, f, j
2	b, d, a, c
1	b, d, a, c, f
1	b, d, a, c, f
1	b, d, c, f, h
1	a, c, h, l, e

Table 5. Frequency descending closed node count in between min-l and max-l.

Support count (Sorted)	Closed-itemsets
5	b, d
3	b, d, a
2	b, d, a, c
1	b, d, a, f, j
1	b, d, a, c, f
1	b, d, a, c, f
1	b, d, c, f, h
1	a, c, h, l, e

4. Conclusion

In this paper we have proposed an algorithm to find closed frequent itemsets in sliding window environment for data streams. To improve processing and mining processes, a FP-tree based compact tree structure FPS-tree is devised using a bit vector representation of the items and sorted items lists are maintained. A dynamic approach to increase support in finding top-k closed frequent itemsets is also presented. Finding frequent itemsets of some specified lengths is achieved by defining *min-length* and *max-length* parameters to restrict size of itemsets to be mined. This limiting factor really improved the performance of the FPS-tree in the cases where users are interested in those itemsets that are between the length ranges. We have performed different experiments to verify the performance of FPS-tree and compared it with recently proposed efficient algorithms like LDS, FCI-Max and TOPSIL-Miner. Results reflect that FPS-tree outperformed all the three algorithms on different datasets.

References

[1] B. Li, “Fining Frequent Itemsets from Uncertain Transaction Streams (2009) pp. 331–335.

[2] B. Yang and H. Huang, Knowledge and Information Systems **23**, No. 2, (May 2009) p. 225

[3] J. H. Chang and W. S. Lee, Finding Recent Frequent Itemsets Adaptively Over Online Data Streams (2003) p. 487.

[4] Yin-Ling Cheung and Ada Wai-Chee Fu, IEEE Transactions on Knowledge and Data Engg. **16**, No. 9, (Sept. 2004) p. 1052

[5] P. Songram and V. Boonjing, N-Most Interesting Closed Itemset Mining (2008) p. 619

[6] D. Lee and W. Lee, Finding Maximal Frequent Itemsets over Online Data Streams Adaptively (2005) pp. 266–273.

[7] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, Finding (Recently) Frequent Items in Distributed Data Streams (2004) pp. 767–778.

[8] G. S. Manku and R. Motwani, Approximate Frequency Counts Over Data Streams, Proceedings of the 28th International Conference on Very Large Data Bases, (2002) pp. 346–357.

[9] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, VLDB’94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, (1994) pp. 487–499.

[10] J. Wang, J. Han, Y. Lu and P. Tzvetkov, IEEE Transactions on Knowledge and Data Engg. **17**, No. 5 (2005) 652.

[11] P. S. M. Tsai, Expert Systems with Applications **37**, No. 10 (2010) 6968.

[12] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, Discovering Frequent Closed Itemsets for Association Rules (1999) p. 398

[13] H.-F. Li, C.-C. Ho and S.-Y. Lee, Expert Systems with Applications **36**, No. 2 (2009) pp. 2451

[14] F. Ao, J. Du, Y.n Yan, B. Liu and K. Huang, An Efficient Algorithm for Mining Closed Frequent Itemsets in Data Streams, IEEE 8th International Conference on CIT (2008) p. 37.

[15] C. C. Aggarwal, Data Streams Models and Algorithms. New York: Springer (2007).

[16] M. M. Gaber, A. Zaslavsky and S. Krishnaswamy, ACM SIGMOD Record, **34**, No. 2 (2005) 18.

[17] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong and Y.-K. Lee, CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining, Advances in Knowledge Discovery and Data Mining **5012** (2008) 1022.

[18] T. Hu, S. Y. Sung, H. Xiong and Q. Fu, Discovery of Maximum Length Frequent Itemsets, Information Sciences **178**, No. 1 (2008) 69.

[19] J. Han, H. Cheng, D. Xin and X. Yan, Data Mining and Knowledge Discovery **15**, No. 1 (2007) 55.

[20] Jia-dong Ren and Ke Li, Online Data Stream Mining of Recent Frequent Itemsets Based on Sliding Window Model (2008) pp. 293–298.

[21] H.-F. Li, C.-C. Ho, M.-K. Shan and S.-Y. Lee, Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window (2006) p. 2672.

[22] C. Leung and Q. Khan, DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams (2006) pp. 928.

[23] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window (Nov 2004) p. 59.

[24] L. Jin, D. J. Chai, Y. K. Lee and K. H. Ryu, Mining Frequent Itemsets over Data Streams with Multiple Time-Sensitive Sliding Windows (2007) p. 486.

[25] B. Mozafari, H. Thakkar and C. Zaniolo, Verifying and Mining Frequent Patterns from Large Windows over Data Streams (2008) p. 179.

[26] S.K. Tanbeer, C. F. Ahmed, B.-S. Jeong and Y.-K. Lee, Information Sciences **179**, No. 22 (2009) 3843.

[27] H.-F. Li, Expert Systems with Applications **36**, No. 7 (2009) 10779.

[28] C. Lin, D. Chiu, and Y. Wu, Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window, SDM (2005).

[29] A.J.T. Lee and C.-S. Wang, Information Sciences **177**, No. 17 (2007) 3453.

[30] C. Leung and Q. Khan, Efficient Mining of Constrained Frequent Patterns from Streams (2006) 61.

[31] M. J. Zaki and C.-J. Hsiao, IEEE Transactions on Knowledge and Data Engg. **17**, No. 4 (2005) 462.

[32] J. Han, J. Wang, Y. Lu, and P. Tzvetkov, Mining Top-K Frequent Closed Patterns without Minimum Support, Proceedings of IEEE International Conference on Data Mining (2002) p. 211.

[33] A. Metwally, D. Agrawal, and A. Abbadi, Efficient Computation of Frequent and Top-k Elements in Data Streams, Database Theory, 3363, T. Eiter and L. Libkin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, (2004) pp. 398

[34] A. W. Fu, R. W. Kwong, F. Renfrew, W. Kwong, and J. Tang, Mining N-most Interesting Itemsets (2000)

[35] H.-F. Li and S.-Y. Lee, Expert Systems with Applications **36** No. 2, (2009) 1466.

[36] C. Lin, D. Chiu, and Y. Wu, Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window, SDM (2005).

[37] M. Deypir and M. H. Sadreddini, Journal of Systems and Software **85**, No. 3 (2012) 746.

[38] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, Knowledge and Information Systems, **10** No. 3 (2006) 265.

[39] J. Cheng, Y. Ke, and W. Ng, Journal of Intelligent Information Systems **31**, No. 3, (2007) 191.

[40] H. Li and H. Chen, Data & Knowledge Engineering **68**, No. 5 (2009) 481.

[41] J.-L. Koh and C.-Y. Lin, Concept Shift Detection for Frequent Itemsets from Sliding Windows over Data Streams, Database Systems for Advanced Applications, **5667**, L. Chen, C. Liu, Q. Liu, and K. Deng, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg (2009) p. 334.

[42] R. C. Wong and A. W. Fu, Mining Top-K Itemsets Over a Sliding window Based on Zipfian Distribution, SIAM International Conference on Data Mining (2005).

[43] J. Han, J. Pei, and Y. Yin, Mining Frequent Patterns without Candidate Gneration (2000) pp. 1–12.