FREQUENT ITEMSET MINING IN TRANSACTIONAL DATA STREAMS BASED ON QUALITY CONTROL AND RESOURCE ADAPTATION

J. Chandrika[1], Dr. K. R. Ananda Kumar[2]

[1]Dept. of Computer Science and Engineering, MCE, Hassan, India,
[1]jc@mcehassan.ac.in
[2]Dept. of Computer Science and Engineering, SJBIT, Bangalore,
[2]kra_megha@hotmail.com

ABSTRACT

The increasing importance of data stream arising in a wide range of advanced applications has led to the extensive study of mining frequent patterns. Mining data streams poses many new challenges amongst which are the one-scan nature, the unbounded memory requirement and the high arrival rate of data streams.Further the usage of memory resources should be taken care of regardless of the amount of data generated in the stream. In this work we extend the ideas of existing proposals to ensure efficient resource utilization and quality control. The proposed algorithm RAQ-FIG (Resource Adaptive Quality Assuring Frequent Item Generation) accounts for the computational resources like memory available and dynamically adapts the rate of processing based on the available memory. It will compute the recent approximate frequent itemsets by using a single pass algorithm. The empirical results demonstrate the efficacy of the proposed approach for finding recent frequent itemsets from a data stream.

KEYWORDS

Transactional data stream, sliding window, frequent itemset, Resource adaptation, Bit sequence representation, Methodical quality.

1. INTRODUCTION

A transactional data stream is an unbounded continuous stream of records that contains a set of items together with a unique identification number for every transaction. Mining frequent patterns (or itemsets) have been studied extensively in the literature of data mining [1][2]. Frequent patterns can be very meaningful in data streams. In network monitoring, frequent patterns can correspond to excessive traffic which could be an indicator for network attack. In sales transactions, frequent patterns relate to the top selling products in a market, and possibly their relationships. Due to numerous applications of this sort mining frequent patterns from data stream has been a hot research area. If we consider that the data stream consists of transactions, each being a set of items, then the problem definition of mining frequent patterns can be written as - Given a set of transactions, find all patterns with frequency above a threshold *s*. Traditional mining algorithms assume a finite dataset over which the algorithms are allowed to scan multiple times. Therefore it is necessary to modify these algorithms in order to apply these algorithms to transactional data streams. An important property that distinguishes data stream mining from traditional data mining is the unbounded nature of stream data, which precludes multiple-scan algorithms. Traditional frequent itemsets mining algorithms are thus not applicable to a data stream [3].

According to the stream processing model, the research of mining frequent itemsets in data streams can be divided into three categories, snapshot windows, sliding windows and Landmark windows. In snapshot window model the frequent items are detected in a fixed range of time so that model is not used frequently. In landmark window model knowledge discovery is performed based on the values between the specific timestamp called landmark and the present. In the sliding window model knowledge discovery is performed over a fixed number of recently generated data elements which is the target of data mining. Sliding window model is a widely used model to perform frequent itemset mining since it considers only recent transactions and forgets obsolete ones. Due to this reason, a large number of sliding window based algorithms have been devised [4][5] [7][8][9][10]. However, only a few of these studies adaptively maintain and update the set of frequent itemsets [7][8][9] and others only store sliding window transactions in an efficient way using a suitable synopsis structure and perform the mining task when the user requests.

Resource adaptation refers to adjusting the processing speed in accordance with the available resources so that the algorithm will not run out of computational resources like memory. In this study, a novel approach based on resource adaptation for mining frequent itemsets over data streams is proposed which operate under sliding window model. The basic algorithm considered for this work is MFI_TRANSW [5] which uses an efficient bit sequence representation for representing the transactional data items. For each item X in the current transaction-sensitive sliding window a bit-sequence with W bits, denoted as Bit(X), is constructed. If an item X is in the i-th transaction of current sliding window, the i-th bit of Bit(X) is set to be 1; otherwise, it is set to be 0. Once the bit sequence representation is available finding the frequent itemset becomes easy as the count of one's indicate the number of transactions in which an itemset is present. Bitwise logical AND operation is performed to get an itemset of length greater than one. The MFI_TRANSW algorithm doesn't take into consideration any resource requirements while processing the stream for the generation of frequent itemset. The proposed algorithm enhances this basic algorithm by performing resource adaptive computation.

In addition to resource adaptation quality of mining results is another important aspect to be dealt with. A high quality algorithm always will result in false negative result. That is an infrequent itemset will never be included in the result set. The quality of mining results can be demonstrated based on precision and accuracy. Most of the previous works deal with these two aspects separately. In this work we emphasize both on the resource usage and the quality of the output.
The rest of the paper is organized as follows. The next section formally states the problem and introduces the basic terminologies. In section 3, some previous related works on frequent itemset mining are reviewed. Section 4 presents the proposed approach and section 5 empirically compares the approach to a related algorithm MFI_TRANSW. Finally, section 6 concludes the paper.

2. PRELIMINARIES

It is necessary to give a formal definition for the transactional data stream before defining the frequent itemset mining problem. Let Y = {*i*1, *i*2, …, *im*} be a set of **items**. A **transaction** *T* = (*tid*, *x*1*x*2....*xn*), *xi* belongs to Y, for 1 <= *i* <= *n*, is a set of items, while *n* is called the **size** of the transaction, and *tid* is the unique identifier of the transaction. An **itemset** is a non-empty set of items. An itemset with size *k* is called a *k-itemset.*

Definition 1: A **transaction data stream** *TDS* = *T*1, *T*2, …, *TN* is a continuous sequence of transactions, where *N* is the tid of latest incoming transaction *TN*.
In order to process the elements of the transactional data stream by considering the recent transactions a **sliding window** mechanism has been implemented for data processing, that is, the

incoming transactions are processed batch by batch. This means that a group of say b transactions (basic window) are processed at a time. Sliding takes place on processing all basic windows of the sliding window to get the next batch of data to be processed. Such a batch of transaction taken in a window called **basic window**, such 'k' number of basic windows are taken to form a main window called **sliding window**. The mechanism adopted for processing the elements of transactional data stream is depicted by the diagram shown below. The diagram indicates how a sliding window is composed of basic windows. The intuition behind splitting the main window into a number of sub windows is that at the time of window sliding its possible to skip a batch of records instead of only a single oldest record. This will ensure a better way of processing. So the window is not a transaction sensitive window as in the case of MFI_TRANSW.

In essence, a set of transactions which are processed at a time forms a basic window. A basic window will be composed of fixed number of transactions. A group of basic windows form a sliding window. In data streams, with the continually arriving of transactions, the new basic window is inserted into the sliding window, and at the same time the oldest basic window is deleted from the sliding window, so the sliding window is always formed by the latest *k* basic windows.

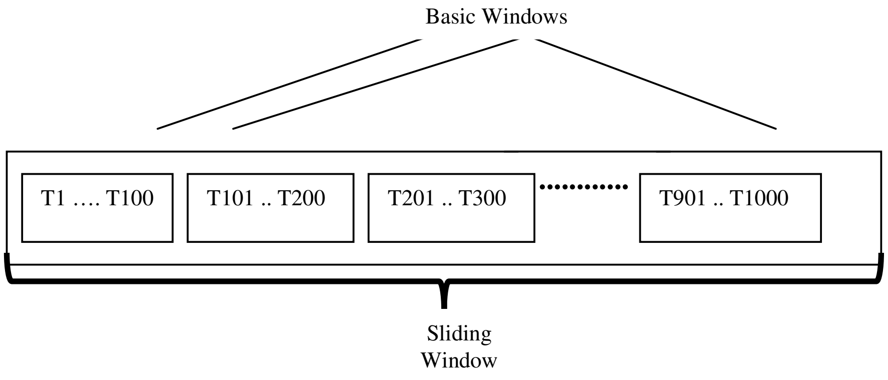


Fig1. The Sliding Window Model

Definition 2: The window at each slide has *w* number of transactions, and *w* is called the *size* of the basic window and w*number of basic windows will give the size of the sliding window. The **support** of an itemset *X* over *TransactionSW*, denoted as **sup**(*X*)*TransactionSW*, is the number of transactions in *TransactionSW* containing *X* as a *subset.*

Definition 3: An itemset *X* is called an **exact frequent itemset** (*FI*) if **sup**(*X*)*TransactionSW* >= $s \times w$, where *s* is a user-defined minimum support threshold (MST) in the range of [0,1]. The value $s \times w$ is called the **frequent threshold** of *TranactionSW* (*FTTransactionSW*).

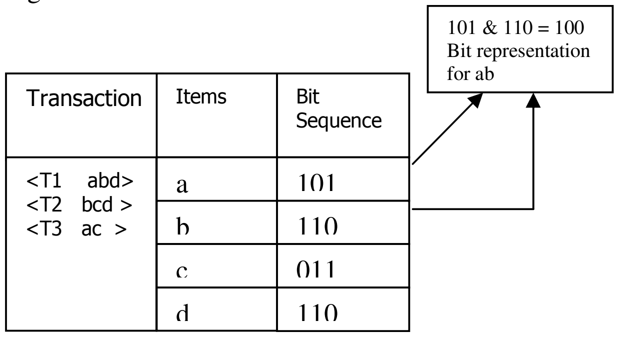
Definition 4: A frequent itemset of length K is called K- frequent itemset.

Definition 5: An itemset X in the given sliding window can be considered as a frequent itemset if **sup**(*X*)*TransactionSW* < $s \times w$ but **sup**(*X*)*TransactionSW* >=($s \times w$ – *€),* where € is user defined minimum error threshold.

With these preliminary definitions the problem under consideration can be formally stated as follows:

Given a transactional data stream the problem is to find all the recent k-frequent itemsets with the user specified minimum support threshold. It is also necessary to track the run time resources consumed like amount of memory and CPU time so that at any instance of time if the computation cannot be carried further with the available resources the input rate will be adjusted by modifying the sliding window size.

The computation of frequent itemsets can be done efficiently by representing all the items in the transactions of the data stream by bit sequence representation [5]. If there are three transactions in the given window then the length of the bit sequence will be three bits. First bit represents first transaction and it is set to one if the item is present in the transaction. The same is repeated for all the bits of the transaction. For instance if a sliding window is composed of three transactions T1, T2, T3 containing different itemsets as shown in Figure 2, the corresponding bit sequence representation is derived for every item by considering the transactions in which the items are present. This is illustrated in figure 2.



Transaction	Items	Bit Sequence
<T1 abd> <T2 bcd > <T3 ac >	a	101
	b	110
	c	011
	d	110

Figure 2. Bit Sequence Representation of Items.

Once the bit sequence representation is ready for all the distinct items in the given set of transactions it is easy to derive itemsets of length two by performing bitwise AND operation. The concept is illustrated in Figure 2, where the bit sequence representation for itemset ab is obtained by performing bitwise AND on the bit sequence representation of item a and the bit sequence representation for item b. The same procedure can be repeated for deriving bit sequence representation of other itemsets of length two. Once 2 itemsets are available they can be used to generate the bit sequence representation of 3 itemsets and so on. The bit sequence representation enables the computation of frequent item sets easily because counting the number of 1's in the bit sequence will give us the exact count of number of transactions that contain the itemset. This count can be compared with the support threshold to find if the itemset is frequent or not.

3. Related Work

There are a number of previous studies which addresses the problem of mining the frequent itemsets from the data streams. In [2], Manku and Motwani have developed two single-pass algorithms, Sticky-Sampling and Lossy Counting, to mine frequent items over the data streams. Jia-dong et.al have proposed an approximation algorithm[4] that stores the frequent itemsets in a compact data structure, further the outputs of the mined results are stored in a heap. The algorithm MFI_TRANSW proposed by Li et. al [5] uses the bit sequence representation for the items which enables finding the count of itemsets by simple logical and operation. In [7] Chang et. al have proposed a mining algorithm for finding frequent itemsets over sliding windows in a data stream based on the theory of Combinatorial Approximation to approximate the counts of itemsets from some summary information of the stream. The concept of resource awareness in data stream mining is first introduced by Gaber et. al [6]. They have proposed light weight techniques for classification and clustering. Their adaptation method is to adjust the processing rate depending on the amount of output produced which is called as Algorithm output granularity. The approach suggested however is not applicable to frequent itemset mining. In [8] algorithm called estDec is proposed to find significant item-set in a data stream that is represented by using a prefix tree. Consequently, the resulting set of frequent itemsets in a data stream can be found instantly at any moment. In [9] a ***CP-tree*** (***Compressed-Prefix tree***) to replace the role of a prefix tree in the *estDec* method. The algorithm also introduces the extended version of the *estDec* method, namely *estDec*+ for a CP-tree. This is another algorithm that performs resource adaptation by reducing the space requirements for the CP Tree by periodically merging the nodes of the tree. Caixia Meng has proposed an algorithm to find frequent itemsets based on first storing necessary information about every transaction and then calculating the frequency in the next step. Because of delay in calculation the algorithm is named as defer counting[10]. Mahmood Deypir et. al have developed an algorithm for frequent itemset mining which operates in both transactional and time sensitive sliding window model [11]. The algorithm proposed uses a prefix tree structure for maintaining the elements of the stream. Zhayong Qu et.al have proposed MFIBA[12], the algorithm that computes the frequent itemsets by using bitwise representation of transactions and by performing bitwise and operation. This algorithm uses an array structure to store the frequent itemsets. Yu et. al [15] have proposed a single pass false negative approach for finding frequent itemsets from transactional streams based on the concept of chernoff bound. They will calculate the number of transactions o be considered for determining the support count based on a statistical measure called chernoff bound. None of these algorithms performs resource adaptation taking into account the quality factors under the sliding window model.

4. PROPOSED ALGORITHM

The algorithm RA-FIG is based on the idea of MFI_Transw[5] where a bit sequence representation is used for the transaction. Bitwise AND is performed to find the frequent itemsets. The algorithm is capable of generating four itemsets. The same can be extended to an itemset of any desired length.

The proposed algorithm RAQ-FIG will function in three phases, the first phase is concerned with filling in the sliding window with k batches of recent transactions. This is done by first storing the incoming transactions inside a buffer in the main memory. The first w transactions are transferred to first basic window inside the sliding window. The next w transactions are filled into the next basic window of the sliding window. This way all the k basic windows in the sliding window will be filled in. This phase is called as window initialization phase.

The second phase generates the bit sequence representation for each basic window and forms 4-frequent itemsets using the bitwise AND operation. While finding 1 itemset we consider only the exact frequent itemset. This will ensure that the final result will not have any false negative results based on Apriori property [15]. Each time a frequent itemset is generated it will be recorded in a ***hash table*** together with the frequency count.

The third phase is concerned with adaptation. The adaptation is performed based on the framework proposed by our earlier work [13]. According to this framework we need to make an assessment of current resource consumption and the quality of the output achieved so far. The resource consumption will be made in terms of memory allocated, execution speed and the rate of incoming transactions. The output quality is assessed in terms of error threshold € and user defined minimum support threshold *s*. The value of ratio between s and € indicates the deviation of the result got from the actual result. It is always necessary to keep this deviation to the minimum so as to get accurate results. In our algorithm the initial value of $€_0$ specified by the user will be adapted dynamically based on the existing resource availability. Due to this it is possible to achieve a bound on the error. This is the meaning of quality control in the proposed algorithm.

The assessment of resource consumption and quality is made periodically. At any instance if the assessment made indicates that the computational resources are stressed we need to adjust the parameters in the algorithm called the adaptation parameter in order to keep the computation going on. The overall intention is to keep the computation up with the currently available resources. The adaptation parameter in the algorithm that can be altered in order to conserve the computational resources is the sliding window size. As the sliding window size is increased more amount of computation will be performed and vice versa. Depending on the current resource consumption either the sliding window size is increased or decreased. The current resource usage is monitored by computing an adaptation factor (AF). The Adaptation factor is a scalar measure that indicates the current resources usage. It considers three important factors namely execution time, input rate and the memory allotted. Execution time is the average runtime of all the basic windows in a sliding window. The input rate is the rate at which the dynamic data arrive. The memory allocated is the average memory consumed by the transactions in the sliding window. Mathematically the computation of adaptation factor is according to equation (1).

$$AF = (\text{Execution time in seconds} * \text{input rate}) + \sin\left(\frac{\pi}{180} * \text{memory allocated}\right) \ldots\ldots\ldots \quad (1)$$

Sine function used in equation (1) helps us in maintaining the value between 0 and 1. Altering the sliding window size is based on the adaptation factor. Depending on the incoming rate of the transaction and the CPU time sliding window size can be either increased or decreased. The initial sliding window is composed of ten basic windows each of which holds hundred transactions. As the algorithm continues the size of sliding window changes and proportionally the basic window size is varied. This variation in the window size leads to adjusting the number of transactions processed in accordance with the available memory. In addition to varying the sliding window size we also manipulate the error threshold € because a larger value of € may result in very large number of frequent itemsets. The detailed algorithm is outlined below:

Input:

- TDS(Transaction data stream)
- s : minimum threshold support in the range[0-1] eg:0.45
- w: the sliding window size
- $€_0$ initial error threshold.
- $€_{max}$ maximum allowable error
- k number of basic windows

Output:

```
    Set of frequent itemsets.
TransactionSW=NULL; /*Transaction sliding window consists of 'w' transaction*/

BW=w/k;          /*k basic windows form one sliding window.*/

€ = €0;
1.   Repeat

      Read the incoming transactions from the buffer into the basic window.

     Until all k basic windows are filled.

2.   For each k BW do
                    For each item 'x' in the transaction do
                     Generate the bit sequence transformation of 'x'
     End for
```

```
3.  For each bit-sequence Bit(x) in TransactionSW do
                    Record the count in hash table;  /*Count of one-itemset*/
    End for
4.  For each 'k' BW do  /*Frequent itemset generation phase*/
            For each x do
                    AND bit(x) with rest of other items that appear in BW.
                    Record the count in hash table /* count of two-itemsets*/
    End for.
5   For each x1, x2 such that x1!=x2 do
                    Bit(x1) AND Bit(x2) AND all other remaining items
                      Record the count in Three-Itemset table
    End for
6   For each x1, x2,x3 such that x1!=x2!=x3 do
                    Bit(x1) AND Bit(x2) AND all other remaining items
                      Record the count in Four-Itemset table
    End for
7   /* Sliding Window size Adaptation phase */
     Adaptation factor AF is devised conserving memory and execution time
          If(AF are not in the desired limits)
             Alter the SW size proportional to the estimated threshold reading.
             Alter € based on AF subject to a limit of €max
    End if

8   Sup=s*Total no. of transactions processed;
         // Printing the frequent itemsets.
    For each x in hash table
                  If count(x)>Sup or count(x)>Sup-€;
                      Print x as the frequent item.
    End for.

9.  If there are new transactions arriving, Left shift the sliding window contents to
    make room for a batch of new transactions and go to step2.
10  Repeat steps 1-9 as long as transactions arrive from the stream.
```

Steps 1-2 will perform the window initialization phase. Steps 3-6 will compute the frequent itemsets. Step 7 will perform the adaptation that is variation to sliding window size and the error threshold. Step 8 is used for printing the output. Step 9 is for processing a new batch of transactions.

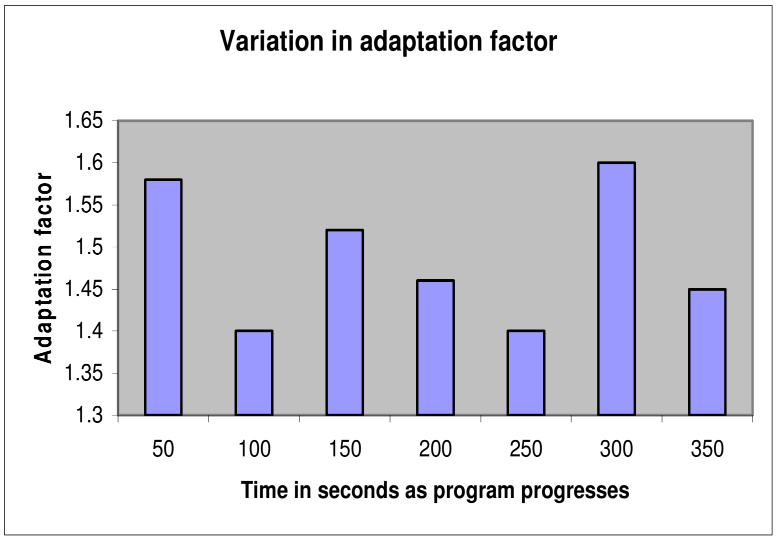
The main novel concept that makes our algorithm overshadow all other concepts and algorithms is resource adaptation and quality control addressed together. Resource adaptive module of the proposed algorithm will consider the execution time and memory to decide performance, thereby deciding efficiency. Initially after each basic window data processing execution time and memory consumption for that particular basic window is recorded. If the value of the adaptation factor exceeds a preset value then sliding window size is varied. According to the empirical evaluation it is observed that a value of 0.4 results in good performance. As such in our experiments the threshold value is set to 0.4. If the current value of adaptation factor falls below this sliding window size is increased and also € is increased by 10% subject to a maximum of $€_{max}$. If adaptation factor is greater than 0.4 sliding window size is reduced as well as € is reduced by 10% so that number of processed and generated itemsets will be reduced and we will conserve the resources.

Varying the window size will balance the memory consumption with the maximum number of transactions processing each time. As the sliding window size is directly proportional to basic window size, on altering sliding window, size of the basic window is also altered and hence achieving the required execution time within the available memory.

5. EXPERIMENTAL EVALUATION

The proposed algorithm is applied to the retail data set available in Frequent itemset mining dataset repository [14]. The data set contains the retail market basket data from an anonymous Belgian retail store. Data collected is over approximately 5 months duration. The total amount of receipts being collected equals 88,163. Each record in the data set contains information about the date of purchase (variable 'date'), the receipt number (variable 'receipt nr'), the article number (variable 'article nr'), the number of items purchased (variable 'amount'), the article price in Belgian Francs (variable 'price' with 1 Euro = 40.3399 BEF) and the customer number (variable 'customer nr'). The elements of the retail data set are looked up in sequence to simulate the environment of a data stream. The algorithm is implemented in on a 1.80 GHz Pentium(R) PC machine with 512 MB memory running on Windows XP. The initial execution starts with sliding window size being set equal to 1000 with ten basic windows each accommodating one hundred transactions. The value for the minimum support threshold is set to 0.3. As the execution proceeds the size of the window will be varied to keep the computation going on. The following graph in figure 3 illustrates the variations made to the sliding window size based on the adaptation factor. A higher value for the adaptation factor indicates greater consumption of computational resources as such a reduction will be made to the sliding window size. Conversely the sliding window size is increased for a lower value of adaptation factor. The empirical studies indicate that the optimal value for the adaptation factor is 0.4. If there is a deviation from this threshold the sliding window size will be varied.

The variation in the value of adaptation factor by the change made to the sliding window size is indicated by the graph in figure 3.



.

Fig 3. Varying the sliding window size based on the adaptation factor

It can be inferred from the graph that as the adaptation factor reduces sliding window size is increased; this will cause the adaptation factor to go up. On happening of such a situation the window size will be reduced. This will continue for the entire history of the data stream. The overall impact is that the computation of the frequent itemsets will continue with available memory resources at an acceptable response time. The system will never degrade drastically because of non availability of memory and CPU resources. Hence the overall scalability of the system improves.

The processing time of RAQ-FIG is compared with MFI-TRANSW, it is observed that the execution time of RAQ-FIG is significantly lesser than that of MFI-TRANSW. The reduction in the execution time is the result of adaptation and the use of hash table for saving the potential frequent itemsets. Another factor that improves the running time of the proposed algorithm is that the window sliding takes place after a batch of transactions as opposed to MFI TRANSW algorithm.. The MFI TRANSW algorithm uses a transaction sensitive sliding window that slides forward for every new transaction that arrives in the input. This obviously consumes more time in the window sliding phase. The RAQ-FIG algorithm will first fill in all the basic windows performs computation and will then slide the window forward by b transactions, where 'b' indicates the basic window size. This will cause the transactions in oldest basic window of the sliding window to be deleted to make room for new set of dynamically arriving transactions.

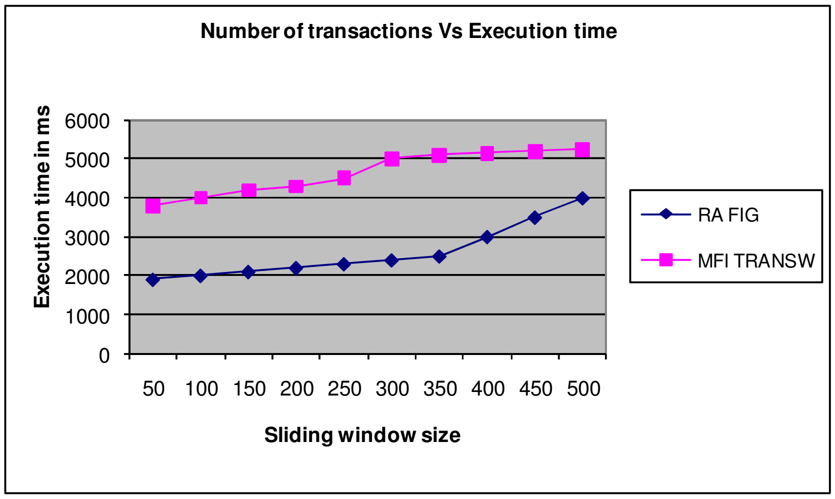


Fig 4. Execution time based on the number of transactions

The memory consumption of RA-FIG over MFI-TRANSW is indicated in the graph below. Even this shows a better resource utilization of RA FIG over MFI TRANSW.

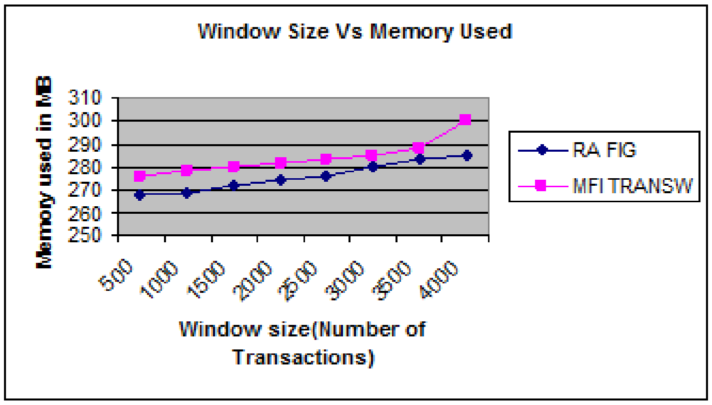
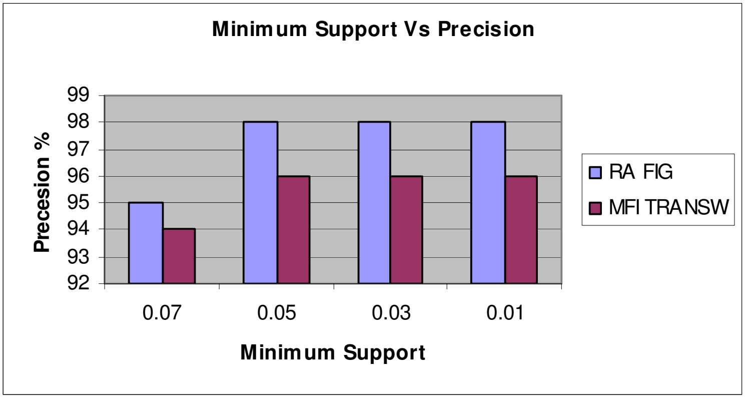
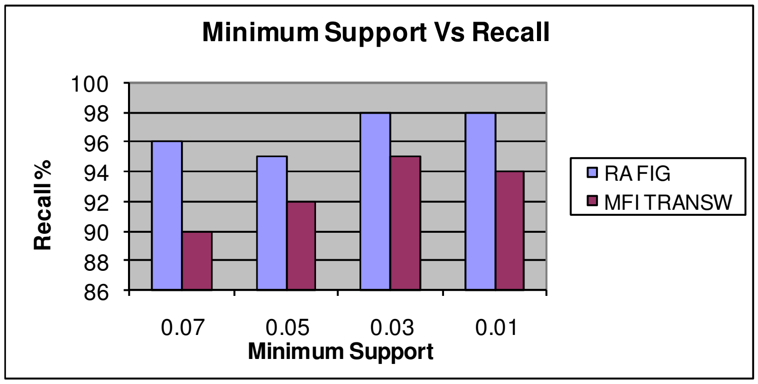


Fig 5. Memory consumption of the proposed algorithm

To measure the quality of the results, we use two metrics—the **recall** and the **precision**. Given a set *T* of true frequent itemsets and a set *O* of frequent itemsets obtained in the output by the algorithm, the recall, denoted by **R**, is $|T\backslash O|/|T|$ and the precision, denoted by **P**, is $|T\backslash O| / |O|$. If the recall equals 1, the results returned by the algorithm contain all the true results. That means no false negatives are returned. If the precision equals 1, all the results returned by the algorithms are some or all of the true results. That means no false positives are returned. An algorithm is said to be *accurate* if *both* the recall and the precision are near to 1.00. The recall and precision values for the proposed algorithm and the MFI_TRANSW algorithm is shown by the graph below:





6. CONCLUSION

In this paper, we have proposed an algorithm for mining frequent itemsets over online data streams with a sliding window model. The sliding window is composed of basic windows. Instead of sliding the window for every transaction, the window will move forward by b transactions where b indicates the size of the basic window. An efficient bit sequence representation is used for computing the frequent itemsets.All the potential frequent itemsets retrieved until the given time will be saved in the hash table. Further the resource adaptive computation is made to decide the size of the sliding window. The algorithm also adjusts the error threshold that controls the output quality dynamically based on the available resources. Due to these measures the accuracy of output results and also running time of the algorithm will improve. The Experiments show that the proposed algorithm not only attain highly accurate mining results, but also run significantly faster and consume less memory than the existing algorithms for mining frequent itemsets over online data streams.

REFERENCES

[1] G.Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang,and P.S. Yu. (2003)Online Mining of Changes from Data Streams: Research Problems and Preliminary Results. In Proc. of the Workshop on Management and Processing of Data Streams.

[2] G. Manku and R. Motwani, (2002) "Approximate frequency counts over data streams," In: Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong, China: Morgan Kanfman, pp. 346-357.

[3] J. Han, H. Cheng, D. Xin, & X. Yan. (2007) "Frequent pattern mining: current status and future directions", Data Mining and Knowledge Discovery, vol. 15(1), pp. 55–86,

[4] Jia Ren, Ke LI , (July 2008) " Online Data Stream Mining Of Recent Frequent Itemsets Based On Slidig Window Model ",Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming,

[5] Hua-Fu Li, Chin-Chuan Ho, Man-Kwan Shan, and Suh-Yin Lee, (October 8-11, 2006) " Efficient Maintenance and Mining of Frequent Itemsets over Online Data Streams with a Sliding Window", IEEE international Conference on Systems, Man, and Cybernetics Taipei, Taiwan

[6] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, ,(2005) "Resource aware Mining of data streams", Journal of universal computer science,vol II ,pp 1440-1453

[7] K.FJea,C ,W Li,T P Chang, "An efficient approximate approach to mining frequent itemsets over high speed transactional data streams ",Eighth International conference on intelligent system design and applications, DOI 10.1109/ISDA2008.74

[8] J.H. Chang and W.S. Lee, (2003).Finding recent frequent itemsets adaptively over online data streams. In Proc. of the 9th ACM SIGKDD, pp. 487-492,

[9] D Lee ,W Lee. (2005). "Finding Maximal Frequent Itemsets over Online Data Streams Adaptively", Fifth Intl. Conference on Data Mining

[10] Caixia Meng, (2009)"An efficient algorithm for mining frequent patterns over high speed data streams", World congress on software engineering, IEEE

[11] Mahmood Deypir & Mohammad Hadi Sadreddini, (2011) " An Efficient Algorithm for Mining Frequent Itemsets Within Large Windows Over Data Streams ",International Journal of Data Engineering (IJDE), Volume (2) : Issue (3)

[12] Zhayang Qu, Peng Li and Yaying Li, (2010) "A High efficiency algorithm for mining frequent itemsets over transactional data streams", International Conference on intelligent control and information processing, IEEE, Dalian , China .

[13] J. Chandrika, Dr. K. R. Ananda Kumar, (June 2011)." A Novel Conceptual Framework for Mining High Speed Data Streams", International Conference on Business, Engineering and Industrial Applications ICBEIA2011, Kuala Lumpur, Malaysia

[14] Frequent Itemset Mining Dataset Repository http://fimi.ua.ac.be/data/

[15] J. X. Yu, Z. Chong, H. Lu, A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In Proc. VLDB'2004