Resource Adaptive Technique for Frequent Itemset Mining in Transactional Data Streams

Chandrika.J[†] and K.R.Ananda Kumar[††],

MCE Hassan SJBIT Bangalore

Summary
Mining Frequent itemsets from transactional data streams is a very challenging task as it has to handle continuous, unbounded, and ordered sequence of data elements generated at a rapid rate in a data stream. In order to enhance the analysis of stream data it is essential to extract frequent itemsets from more recent data. For this purpose a sliding window mechanism is used. Further the usage of memory resources should be taken care of regardless of the amount of data generated in the stream. The proposed algorithm RA-FIG (Resource Adaptive Frequent Item Generation) accounts for the computational resources like memory available and dynamically adapts the rate of processing based on the available memory. Extensive experimental analysis shows that proposed algorithm is efficient in terms of resource utilization and accuracy when finding recent frequent itemsets from a data stream.

Key words:
Transactional data stream, sliding window, frequent itemset, Resource adaptation, Bit sequence representation.

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. Extracting frequent itemsets from data streams [1][2] is a crucial step in association analysis. It is very beneficial to extract frequent itemsets from transactional data streams as it finds applications in many areas such as business decision support and direct marketing. For example, identifying groups of products that are frequently purchased together helps a company to formulate its marketing strategies. 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 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. This 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.
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

Manuscript received October 5, 2012
Manuscript revised October 20, 2012

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 at a time. Sliding takes place on processing each batch to get the next batch to be processed. Such a batch is taken in a window called **basic window**, such 'k' number of basic windows are taken to form a main window called **sliding window**.

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.

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.*

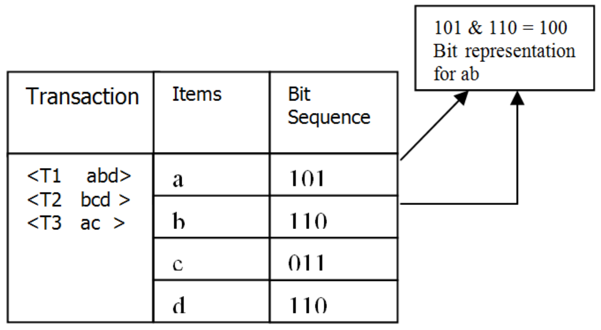
Definition 3: An itemset *X* is called a **frequent itemset** (*FI*) if **sup**(*X*)*TransactionSW* >= *s*×*w*, where *s* is a user-defined minimum support threshold (MST) in the range of [0,1]. The value *s*×*w* is called the **frequent threshold** of *TranactionSW* (*FTTransactionSW*).

Definition 4: A frequent itemset of length k is called k frequent itemset.

In a variation of the same problem Error threshold will be considered, which decides by what amount the infrequent itemset having negligible variation from the threshold can be considered as frequent. 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 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 1.



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

Figure 1. 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. None of these algorithms performs resource adaptation 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 RA-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. The second phase generates the bit sequence representation for each basic window and forms 4-frequent itemsets using the bitwise AND operation. The results are saved in a ***hash table***. 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 in terms of memory allocated, execution speed and the rate of incoming transactions. The assessment 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.. Based 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 = (Execution time in seconds * input rate) + sin ($\frac{\pi}{180}$ * memory allocated) …………………… (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. 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

Output:

- Set of frequent itemsets.

```
    TransactionSW=NULL; /*Transaction sliding
                    window consists of 'w' transaction*/
     BW=w/k;    /*k basic windows form one sliding
                    window.*/

Repeat
     If TransactionSW is not FULL
     Repeat    //window initialization phase
       Read the incoming transactions from the
       buffer into the basic window.
     Until all k basic windows are filled.
     Else //discard the older transactions by left shift
     Do bit-wise shift to left on the bit sequence
     Of each basic window
     Accommodate the incoming transaction from the
     buffer into the basic window.
     For each k BW do
        For each item 'x' in the transaction do
        Generate the bit sequence transformation of 'x'
       End for
For each bit-sequence Bit(x) in TransactionSW do
    Record the count in hash table; /*Count of one-
                                   itemset*/
End for
For each 'k' BW do /*Frequent itemset generation
                             phase*/
   For each item 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.
       For each x1, x2 such that x1!=x2 do
           Bit(x1) AND Bit(x2) AND remaining items
           Record the count in Three-Itemset table
      End for
      For each x1, x2,x3 such that x1!=x2!=x3 do
          Bit(x1) AND Bit(x2) AND remaining items
          Record the count in Four-Itemset table
      End for
 End for
  Sup=s*Total no. of transactions processed;
      // Printing the frequent itemsets.
  For each x in hash table
          If count(x)>Sup;
                 Print x as the frequent item.
  End for.
  /* 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.
 End if
Until transactions arrive in the buffer.
```

The main novel concept that makes our algorithm overshadow all other concepts and algorithms is resource adaptation. 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 the input threshold then sliding window 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. If it is greater than 0.4 sliding window size is reduced so as to 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. 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.

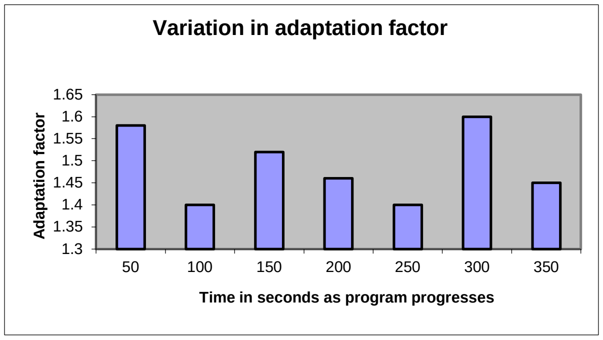


Fig 2. Adaptation factor variation resulting from sliding window size adjustment

The processing time of RA FIG is compared with MFI-TRANSW, it is observed that the execution time of RA 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 RA 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.

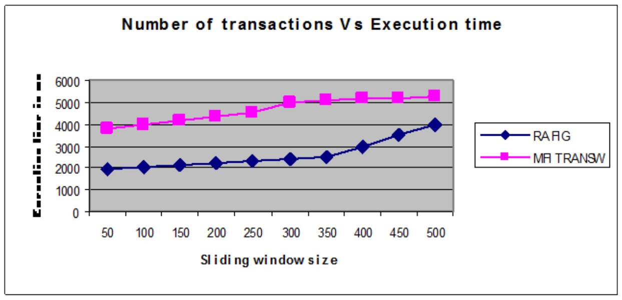


Fig 3. Execution time based on the number of transactions

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.

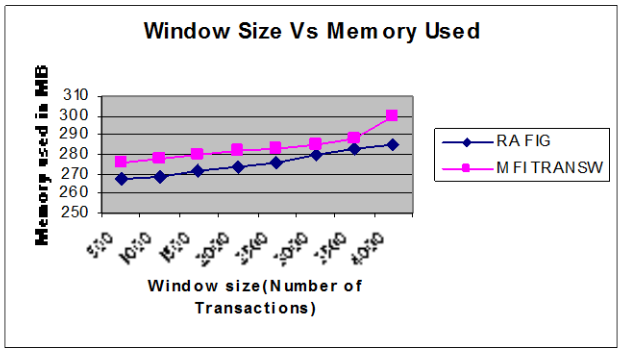


Fig 4. Memory consumption of the proposed algorithm

6. Conclusion

In this paper, we have proposed an algorithm for mining frequent itemsets over online data streams with a transaction-sensitive sliding window. 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. 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. Due to these measures the time efficiency 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. Online Mining of Changes from Data Streams: Research Problems and Preliminary Results. In *Proc. of the Workshop on Management and Processing of Data Streams*,((2003).

[2] G. Manku and R. Motwani, “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. (2002)

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

[4] Jia Ren, Ke LI ,” 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, (July 2008)

[5] Hua-Fu Li, Chin-Chuan Ho, Man-Kwan Shan, and Suh-Yin Lee,” 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 (October 8-11, 2006)

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

[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, Finding recent frequent itemsets adaptively over online data streams. In *Proc. of the 9th ACM SIGKDD*, pp. 487-492, (2003).

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

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

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

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

[13] J. Chandrika, Dr. K. R. Ananda Kumar,” A Novel Conceptual Framework for Mining High Speed Data Streams”, International Conference on Business, Engineering and Industrial Applications ICBEIA2011, Kuala Lumpur, Malaysia 5 - 8 (June 2011).

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