Mining Recent Maximal Frequent Itemsets Over Data Streams with Sliding Window

Saihua Cai[1], Shangbo Hao[1], Ruizhi Sun[1], and Gang Wu[2]
[1]College of Information and Electrical Engineering, China Agricultural University, China
[2]Secretary of Computer Science Department, Tarim University, China

Abstract: *The huge number of data streams makes it impossible to mine recent frequent itemsets. Due to the maximal frequent itemsets can perfectly imply all the frequent itemsets and the number is much smaller, therefore, the time cost and the memory usage for mining maximal frequent itemsets are much more efficient. This paper proposes an improved method called Recent Maximal Frequent Itemsets Mining (RMFIsM) to mine recent maximal frequent itemsets over data streams with sliding window. The RMFIsM method uses two matrixes to store the information of data streams, the first matrix stores the information of each transaction and the second one stores the frequent 1-itemsets. The frequent p-itemsets are mined with "extension" process of frequent 2-itemsets, and the maximal frequent itemsets are obtained by deleting the sub-itemsets of long frequent itemsets. Finally, the performance of the RMFIsM method is conducted by a series of experiments, the results show that the proposed RMFIsM method can mine recent maximal frequent itemsets efficiently.*

Keywords: *Data streams, recent maximal frequent itemsets, sliding window, matrix structure.*

Received November 16, 2016; accepted April 25, 2018

1. Introduction

The scale of collected data information shows an explosive growth in various domains with the rapid development of Internet of Things (IoT) technology, information technology and network technology [1]. In order to analyze the collected data information better, the association rules [3] among data should to be mined effectively, and the generation of frequent itemsets is the most critical technique and procedure in mining association rules. The frequent itemsets [7, 12] mean the itemsets whose *support* is not smaller than predefined minimum support (denoted as *min_sup*). In general, the number of frequent itemsets generated by most frequent itemsets mining methods is very large because all frequent itemsets of a given dataset need to be mined, it can easily consume the memory usage. Due to the number of maximal frequent itemsets is relatively smaller and they can imply all frequent itemsets perfectly, in this case, the efficiency of time cost for mining maximal frequent itemsets is very good and the memory usage is also much smaller. Therefore, the problems of mining frequent itemsets can be transformed into the operation of mining maximal frequent itemsets.

The massive use of sensors makes the collected data exist in the form of data streams. One common definition of data streams is that it is made up of massive unbounded sequences with a large amount of data elements and existed in the form of continuous streams [11]. Data streams are generated in many aspects of applications, such as: sensor data streams are generated from sensor networks [4], online transaction data streams are generated from shops, network data streams are generated from website, etc. Due to people need to grasp the relevance of the information generated from online applications in real time, the mining process should be treated immediately.

The differences of data streams mining and static data mining are list as follows [8]. First, each element of data streams is allowed to be checked for most once. Second, memory usage of the analysis for data streams should be restricted finitely although new data elements generated continuously. Third, the newly generation of the data should be processed as fast as possible. Fourth, the up-to-data analysis result of data streams should be instantly available when user requested. To satisfy these requirements, the method of data streams mining should indulge the rapidity and the memory usage should be as small as possible.

However, the huge number of data streams makes it impossible to store all the data into main memory. Moreover, previous methods that studied for mining static datasets are not feasible for mining data streams, in this case, new structures and mining methods are eager for using to support one-time and continuous mining process. In this paper, we use the matrix structure to store the data information of data structures, and then propose an improved method called Recent Maximal Frequent Itemsets Mining (RMFIsM) to mine recent maximal frequent itemsets over data streams.

The rest of this paper is organized as follows. The related work is introduced in section 2. The definitions and problems statement are given in section 3, the structure and main idea of our proposed maximal

frequent itemsets mining method are described in section 4. The experimental analysis is presented in section 5, and the conclusion is given in section 6.

2. Related Work

At present, several methods have proposed to mine the frequent itemsets over data streams. The models of frequent itemsets mining can be divided into:

1. Landmark window model.
2. Damped window model.
3. Sliding window model according to the processing models of data streams.

For landmark window model, the researchers always focus on the data in the entire data streams, and get the global frequent itemsets through the analysis of historical data. Li *et al*. [8] referred to Apriori algorithm to present a method called Data Stream Mining for Maximal Frequent Itemsets (DSM-MFI), it used prefix tree structure to store the data information of data streams, and then the maximal frequent itemsets mining process was realized with the constructed prefix tree structure. INSTANT method was presented by Mao *et al*. [10], it defined some sub-operators of itemsets and maintained itemsets with different level of support in memory, the advantage of INSTANT method was that the maximal frequent itemsets could be displayed directly to user through a serious of sub-operations when the new transaction arriving.

For damped window model, each transaction has a corresponding value and the value decreases gradually with the increase of time, therefore, preserve and reduce the related information of historical data need to be considered in the control of the value. Chang and Lee [2] developed a method called estDec in 2003, this method examined each transaction in turn without the generation of any candidate, the occurred count of the itemsets that appeared in each transaction was maintained with a prefix-tree structure, and the effect of old transaction on current mining result was diminished by defining the parameter called debilitating factor. Lin *et al*. [9] presented the Mining Recently Frequent Itemsets with Variable Support over Data Streams (MRVSDS) algorithm to store frequent itemsets in current window into PFI-tree structure, the itemsets were deleted from PFI-tree when the degree of the transaction was less than *min_sup*. In addition, the authors also designed the Decaying Synopsis Vector (DSYV) structure to store the processed transaction, and the frequent itemsets were found by re-mining the transactions from DSYV when the current itemsets' *support* was less than historical *min_sup*.

For sliding window model, the focus is always on the recent transactions, therefore, the mining results are the local frequent itemsets over a certain period of time. Yang *et al*. [13] designed an efficient algorithm named DSM-Miner to mine maximal frequent itemsets over data streams, it used appropriate method to reduce the effects of old transactions, and then the Sliding Window Maximum frequent pattern Tree (called SWM-Tree) was proposed to maintain the latest pattern's information. In the process of mining maximal frequent patterns, DSM-Miner used appropriate pruning operations, calculation pattern of bit items group and "depth-first" search strategies, the experimental results showed that DSM-Miner was better in time performance and memory usage. A new algorithm that based on the prefix-tree data structure was proposed by Deypir *et al*. [5] to find and update frequent itemsets of the windows, a batch of transactions were used as the unit of insertion and deletion within the window to improve the performance, moreover, an effective traversal strategy for the prefix-tree and the suitable representation for each batch of transactions were used in the algorithm, the required information in each node of the prefix-tree was stored and the old batch of transactions were deleted directly.

However, some disadvantages also have existed in the proposed methods. The drawbacks of INSTANT algorithm [10] were the amount of arrays designed for maintaining all maximal frequent itemsets was very large and the cost of memory usage was also very expensive, moreover, no efficient superset or subset were used to check the newly identified maximal frequent itemsets of each array, therefore, the comparison times were increased very fast and the memory usage was enlarged rapidly when the average length of the transactions became longer.

3. Definitions And Problems Statement

In this section, we first provide some formal definitions of the important terms used in this paper and then give the problems statement.

3.1. Definitions

Let $I= \{i_1, i_2, i_3,\ldots, i_m\}$ be a finite set of m distinct items. The data streams $DS= [T_1, T_2, T_3, \ldots, T_n)$, where each transaction $T_j \in DS$ is a subset of I with a unique identifier TID. If the relation of itemset α and β is $\alpha \subseteq \beta$, α is called the sub-itemset of β and β is called the super-itemset of α. If the length of itemset is k, it is called k-itemset. Table 1 shows an example of data stream as the running example to clearly explain the definitions. In this example, assume that *min_sup* is 0.33 and the size of sliding window is 6.

Table 1. An example of data streams.

TID	Transaction	TID	Transaction
T_1	$\{i_1,i_2,i_3\}$	T_6	$\{i_1,i_2,i_3,i_5,i_6\}$
T_2	$\{i_1,i_2,i_4\}$	T_7	$\{i_1,i_2,i_5\}$
T_3	$\{i_2,i_3,i_5\}$	T_8	$\{i_1,i_2,i_3,i_4\}$
T_4	$\{i_1,i_2,i_3,i_5\}$	T_9	$\{i_2,i_3,i_5\}$
T_5	$\{i_1,i_3,i_5\}$	...	……

- *Support*: The frequency of itemset x_i in *DS* is defined as *support*, that is, *support* ($\{x_i\}$)= *count*(x_i, *DS*) / |S*W*|, where *count*(x_i, *DS*) is the number of contained itemset x_i in *DS* and |S*W*| is the size of sliding window.

For example, itemset $\{i_1\}$ is existing in T_1, T_2, T_4, T_5 and T_6 in current sliding window, therefore, *support* ($\{i_1\}$) = 5/6. Itemset $\{i_1, i_2\}$ is existing in T_1, T_2, T_4 and T_6 in current sliding window, therefore, *support* ($\{i_1, i_2\}$)= 4/6.

- *Frequent Itemsets* (*FIs*): The frequent itemsets mean that the itemsets' *support* is not less than the predefined minimal support threshold *min_sup*.

For example, itemset $\{i_1, i_3\}$ is existing in T_1, T_4, T_5 and T_6, *support* ($\{i_1, i_3\}$)=4/6>0.33, therefore, $\{i_1,i_3\}$ is a frequent itemset.

- *Infrequent Itemsets* (*IFIs*): The infrequent itemsets mean that the itemsets' *support* is less than the predefined minimal support threshold *min_sup*.

For example, itemset $\{i_2, i_4\}$ is existing in T_2, *support* ($\{i_2, i_4\}$)=1/6<0.33, therefore, $\{i_2,i_4\}$ is an infrequent itemset.

- *Maximal Frequent Itemsets* (*MFIs*): The itemsets are the maximal frequent itemsets should satisfy the following two conditions:

1. They are frequent itemsets.
2. No super-itemset of them is frequent.

For example, itemset $\{i_4\}$ is not a *MFI* due to *support* ($\{i_4\}$)=1/6<0.33. Itemset $\{i_1\}$ is not a *MFI* though *support* ($\{i_1\}$) = 5/6>0.33, the reason is that its super-itemset $\{i_1, i_2\}$ is frequent. Itemset $\{i_1, i_2, i_3, i_5\}$ is a *MFI* due to *support* ($\{i_1, i_2, i_3, i_5\}$)=2/6>0.33 and no super-itemset of it is frequent.

- *Dictionary order*: If the appeared sequence of itemset *A* is earlier than itemset *B* in dictionary, the dictionary order of itemset *A* and itemset *B* can be recorded as: *A* » *B*. Similarly, the next itemsets can be recored as: *A* » *ABD* » *ACD* » *BD* in dictionary order.

3.2. Problems Statement

For mining useful information over data streams, the final mining results should be send to users immediately, it means that any useful data should be processed in an efficient way, in this case, the real-time response is very important to users. In addition, the huge nature of data streams makes it impossible to store all the data information into main memory or even in secondary storage due to they can easily consume all resources of system and bring difficulties to the underlying mining tasks.

Specifically, the DSM-MFI method [8] took the structure of summary frequent itemset forest to store every sub-projection of affairs, and two main problems of DSM-MFI method could be included as:

1. Large memory storage was wasted to store the sub-projections for a part of sub-projections were not frequent.
2. Much time was wasted for deleting the sub-projections from summary frequent itemset forest to achieve the lower memory occupancy.

The size of prefix tree that generated by estDec method [2] was very large with the increasing number of frequent itemsets, and more seriously, the estDec method would stop working once the prefix tree occupies full of the memory. The drawback of TMFI method [6] was the infrequent 1-itemsets were also stored in matrix structure, therefore, some meaningless "extension" operation of infrequent itemsets also were conducted to gain longer itemsets.

In general, the time cost, the memory storage and the accuracy rate of mining process are the most important problems we should to deal with.

4. Mining Recent Maximal Frequent Itemsets

In this section, we refer TMFI method [6] to propose an improved method called RMFIsM to mine the recent *MFIs* over data streams. The RMFIsM method uses two matrixes (record as: matrix A and frequent matrix B) to store the information of each item, and the infrequent itemsets need to be deleted from matrix immediately to reduce the time cost and memory usage based on downward closure property.

4.1. The Structure of RMFIsM Method

Matrix A is constructed to store the information of each item of data streams, and frequent matrix B is built to record the information of frequent 1-itemsets.

The rows of matrix A stand for the information of transactions T_i and the columns of matrix A stand for the information of each item of $\{i_1, i_2, i_3,\ldots, i_m\}$, the size of matrix A is (n+1)*m, where row (n+1) records the *support* of each item. Specifically, the transactions are scanned in order when the current sliding window is not full, and $A_{d,k}$ is marked as 1 if item i_k is appeared in transaction T_d, otherwise, $A_{d,k}$ is marked as 0.

In order to effectively mine the recent information of data streams, old transactions need to be replaced by new ones directly. The position of new transaction T_d

is calculated by Equation 1, where n is the size of sliding window, and the information of transaction T_d is recorded in row n if the result of pos is 0.

$$pos = d\%n \quad (1)$$

Frequent matrix B is built to store the data information of frequent 1-itemsets in dictionary order, the original element of matrix B is 0 and the real size of matrix B is (k-1)*(k-1), where k is the number of frequent 1-itemsets. The construction process of matrix B is shown as follows: for frequent 1-itemsets i_p and i_q with the order of i_p » i_q, doing "logic and" operation process for every element of columns p and q in matrix A, $B_{p,q}$ is marked as 1 if the result of "logic and" for itemset $\{i_p,i_q\}$ is not less than *min_sup*, otherwise, $B_{p,q}$ is marked as 0.

Matrix A and frequent matrix B are the basis for mining maximal frequent itemsets, the pseudo-code of constructing matrix A and frequent matrix B is shown in Algorithm 1.

Algorithm 1: Construct matrix A and frequent matrix B

```
Input: Data streams, n(the maximal |SW|), m(maximal number of different items), min_sup
Output: matrix A, frequent matrix B
for (|SW|=1 to n)
{
  for (k=1 to m)
  {
    if (ik in Td)
      Ad,k=1
    else
      Ad,k=0
    }
}
return matrix A
for (k=1 to m)
{
  if (support(ik) ≥min_sup)
    add ik to matrix B
  else
    delete ik
}
for (k=1 to |B|)
{
  for (s=k+1 to |B|)
  {
    if (support({ik,is}) ≥min_sup)
      Bk,s=1
    else
      Bk,s=0
  }
}
return matrix B
```

4.2. Downward Closure Property

The downward closure property is an important part of RMFIsM method, it is the foundation of pruning strategy for reducing the meaningless "extension" process to save the time cost and memory usage.

- *Theorem* 1. If X^k is a frequent k-itemset, then, any nonempty sub-itemset X^{k-1} of X^k is also frequent.
- *Proof*. Since $X^{k-1} \subseteq X^k$, the transactions contains itemset X^k must contains the itemset X^{k-1}, that is: $TID(X^k) \subseteq TID(X^{k-1})$, it follows that: $support(X^{k-1}) \geq support(X^k) \geq min\_sup$. Hence, any nonempty sub-itemset X^{k-1} of X^k is also frequent if X^k is a frequent itemset.
- *Theorem* 2. If X^k is an infrequent k-itemset, then, any super-itemset X^{k+1} of X^k is also infrequent.
- *Proof*. Since $X^k \subseteq X^{k+1}$, the transactions contains itemset X^{k+1} must contains the itemset X^k, that is: $TID(X^{k+1}) \subseteq TID(X^k)$, it follows that: $support(X^{k+1}) \leq support(X^k) \leq min\_sup$. Hence, any super-itemset X^{k+1} of X^k is also infrequent if X^k is an infrequent itemset.

It can be easily known from downward closure property that the "extension" process of infrequent itemsets is meaningless, thus, the downward closure property should to be considered in every step of maximal frequent itemsets mining. More specifically, the infrequent itemsets that existing in matrix A should not add into frequent matrix B as the basic element of "extension" process for RMFIsM method, that is, if 1-itemset i_p is an infrequent itemset, its super-itemsets are impossible being the frequent itemsets, therefore, i_p should not appear in matrix B to reduce the time cost and memory usage in both constructing matrix B and calculating the *support* value of these meaningless extended itemsets.

4.3. The Main Idea of RMFIsM Method

The main idea of RMFIsM method can be included into next three parts:

1. Extend the short frequent itemsets into long itemsets.
2. Calculate the *support* value for the extended long itemsets and save the frequent long itemsets into maximal frequent itemsets library *MFIs_L*.
3. check and move the frequent sub-itemsets of the extended frequent long itemsets out from *MFIs_L*. Note that, each itemset need to be checked before "extension" process to discard the infrequent itemsets to further improve the mining efficiency.

Once matrix A is constructed and each element is written into matrix A, the *support* value of each item is calculated and written in row (n+1), and the frequent 1-itemsets are stored into *MFIs_L*. After constructing matrix B and the corresponding items are written into matrix B, the frequent 2-itemsets where the item is marked in 1 are stored into *MFIs_L*, and then all 1-itemsets are checked and each sub-itemset of frequent 2-itemsets are moved out from *MFIs_L*.

If itemset $\{i_{k1}, i_{k2}, ..., i_{kp}\}$ is a frequent p-itemset, the "extension" process of frequent p-itemset into (p+1)-

itemset can be summarized as follows. Frequent *p*-itemset $\{i_{k1},i_{k2},\ldots,i_{kp}\}$ can be extended into (*p*+1)-itemset if and only if every B(ky,k(*p*+1))= 1, where $y \in [1, p]$. Next, doing "logic and" operation for the corresponding (*p*+1) column to calculate the *support* value, itemset $\{i_{k1},i_{k2},\ldots,i_{kp},i_{k(p+1)}\}$ is retained and stored into *MFIs_L* if its *support* value is not less than *min_sup*, otherwise, it is discarded directly. If the current extended (*p*+1)-itemset is frequent, all *p*-itemsets are checked and each sub-itemset is moved out from *MFIs_L*. Repeat the above "extension" operation until no itemset can be further extended. The specific pseudo-code is shown in Algorithm 2.

Algorithm 2: RMFIsM

```
Input: Frequent matrix B
Output: MFIs
call Algorithm 1
delete each frequent sub-itemsets of frequent 2-itemsets
for (k=1 to |B|)
{
  foreach (B(k_y,k_(p+1))=1)  // y ∈[1,p]
  {
    extend p-itemset of {i_k1,...,i_kp}  into (p+1)-itemset of {i_k1,...,i_kp,i_k(p+1)}
    calculate support({i_k1,...,i_kp,i_k(p+1)})
    if support({i_k1,...,i_kp,i_k(p+1)}) ≥min_sup
      add {i_k1,...,i_kp,i_k(p+1)} into MFIs_L
      move every sub-itemsets of {i_k1,...,i_kp,i_k(p+1)} out from MFIs_L
    else
      delete {i_k1,...,i_kp,i_k(p+1)}
  }
}
return MFIs
```

4.4. An Example of RMFIsM Method

In order to describe our proposed RMFIsM method better, we take the example that shown in Table 1 to illustrate the specific mining process of maximal frequent itemsets, the *min_sup* is set into 0.33 and the size of sliding window is set into 6.

Matrix A is constructed and each data information of transactions is marked into when they pass the sliding window, the original information of each item (T_1-T_6) is shown in Figure 1-a. When the sliding window is full, we built matrix B and implement maximal frequent mining process. When the new transactions flowing into the sliding window, the oldest transaction is covered by the latest one directly based on Equation (1) to get better time efficiency. Figure 1 shows the change process of matrix A that T_1 is covered by T_7 and T_2 is covered by T_8, the new matrix A is shown in Figure 1-b.

	i_1	i_2	i_3	i_4	i_5	i_6
T_1	1	1	1	0	0	0
T_2	1	1	0	1	0	0
T_3	0	1	1	0	1	0
T_4	1	1	1	0	1	0
T_5	1	0	1	0	1	0
T_6	1	1	1	0	1	1
support	0.83	0.83	0.83	0.17	0.67	0.17

a) Original matrix structure.

	i_1	i_2	i_3	i_4	i_5	i_6
T_7	1	1	0	0	1	0
T_8	1	1	1	1	0	0
T_3	0	1	1	0	1	0
T_4	1	1	1	0	1	0
T_5	1	0	1	0	1	0
T_6	1	1	1	0	1	1
support	0.83	0.83	0.83	0.17	0.83	0.17

b) New matrix structure.

Figure 1. The change process of matrix A.

Next, we take the transactions of {T_7, T_8, T_3, T_4, T_5, T_6} as the example to explain the *MFIs* mining process more clearly, the whole process is divided into next steps.

- Retaining the 1-itemsets whose *support* value are not less than *min_sup* and saving them into *MFIs_L*, these frequent 1-itemsets are the basic elements of matrix B. Here, the frequent 1-itemsets in *MFIs_L* are {i_1}, {i_2}, {i_3}, {i_5}.
- Taking the frequent 1-itemsets that saved in *MFIs_L* to construct matrix B, the row of matrix B is the front (*n*-1) elements and the column of matrix B is the last (*n*-1) elements, where *n* is the size of frequent 1-itemsets. Thus, the row of matrix B is {i_1, i_2, i_3} and the column of matrix B is {i_2, i_3, i_5}. Next, the *support* of each 2-itemset ({i_1, i_2}, {i_1, i_3}, {i_1, i_5}, {i_2, i_3}, {i_2, i_5}, {i_3, i_5}) is calculated and their *support* values are marked into matrix B. Then, the frequent 2-itemsets are saved into *MFIs_L*. The specific information of matrix B is shown in Figure 2.

	i_2	i_3	i_5
i_1	1	1	1
i_2	0	1	1
i_3	0	0	1

Figure 2. The structure of matrix B.

- After constructing matrix B, the infrequent itemsets need to be deleted first and each sub-itemsets of frequent 2-itemsets need to be moved out from *MFIs_L*. For the example, frequent 1-itemsets {i_1} and {i_2} are the sub-itemsets of frequent 2-itemset {i_1, i_2}, then, moving {i_1} and {i_2} out from *MFIs_L*. Continue this operation until no frequent

1-itemsets is the sub-itemset of frequent 2-itemsets. Here, the itemsets in *MFIs_L* are {i_1, i_2}, {i_1, i_3}, {i_1, i_5}, {i_2, i_3}, {i_2, i_5} and {i_3, i_5}.

- Then, the frequent 2-itemsets need to be extended into 3-itemsets. The frequent 2-itemset {i_1,i_2} is first selected as the conditional potential itemset, due to B(i_1,i_3)=1 and B(i_2,i_3)=1, {i_1,i_2} can be extended into {i_1,i_2,i_3} and it is saved into *MFIs_L* due to *support*({i_1,i_2,i_3})=0.5>0.33. Repeat the same process to gain the frequent 3-itemsets {i_1, i_2, i_5}, {i_1, i_3, i_5}, {i_2, i_3, i_5}, and they are saved into *MFIs_L*. After gaining all frequent 3-itemsets, each frequent 2-itemset is checked and all sub-itemsets need to be moved out from *MFIs_L*.
- Next, the frequent 3-itemsets need to be extended into 4-itemsets. The frequent {i_1,i_2,i_3} is first selected as the conditional potential itemset, due to B(i_1,i_5)=1, B(i_2,i_5)=1 and B(i_3,i_5)=1, {i_1,i_2,i_3} can be extended into {i_1,i_2,i_3,i_5} and it is saved into *MFIs_L* for *support*({i_1,i_2,i_3,i_5})= 0.333>0.33. After gaining the frequent 4-itemsets, each frequent 3-itemset is checked and each sub-itemset need to be moved out from *MFIs_L*.

  After above steps, the *MFIs_L* is {i_1, i_2, i_3, i_5}.

5. Experimental Analysis

To verify the efficiency of our proposed RMFIsM method, the estDec method [2], the TMFI method [6] and the DSM-MFI method [8] are compared in our experiment. All experiments are conducting on a machine running Windows 7 with an Intel dual core i3-2020 2.93 GHz processor, the development environment is Microsoft Visual Studio 2010. The performance of RMFIsM method is analyzed on synthetic sparse datasets of *T10.I4.D1000K* and synthetic dense dataset of *T30.I20.D1000K* that generated by IBM data generator, where |*T*| means the average size of the transactions, |*I*| means the potential size of frequent itemsets and |*D*| means the total number of transactions, K means one thousand.

Experiments are conducted to investigate the efficiency of the RMFIsM method both in time cost and memory usage with different value of *min_sup*, different size of sliding window and different number of transactions, the experiments are also conducted to test the accuracy rate of RMFIsM method. Each group of experiments is repeated for 50 times, and the average time and memory usage are calculated.

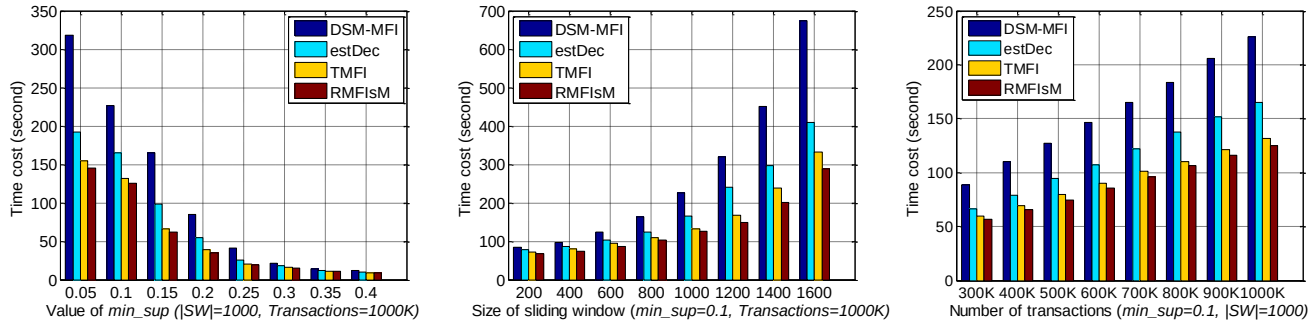
5.1. Time Cost for RMFIsM Method

The time cost for mining recent *MFIs* on sparse dataset *T10.I4.D1000K* with different value of *min_sup* is shown in Figure 3-a. The time cost on *T10.I4.D1000K* with different size of sliding window is shown in Figure 3-b. The time cost on *T10.I4.D1000K* with different number of transactions is shown in Figure 3-c. The time cost on dense dataset *T30.I20.D1000K* is shown in Figure 4-a, Figure 4-b, and Figure 4-c separately.

It can be seen from Figure 3-a and Figure 4-a that the time cost of RMFIsM, DSM-MFI, estDec and TMFI methods shows a decreasing trend with the increasing value of *min_sup*. The time cost of our proposed RMFIsM method is the lowest of the compared four methods, the reason is that in the process of mining *MFIs*, RMFIsM method just implements the "logic and" operation of each data information that stored in matrixes, which reduces the operations of iteration, sorting and pruning, moreover, the infrequent itemsets are discarded directly in RMFIsM method to avoid meaningless "extension" operation. Compared with DSM-MFI method, the saved time of RMFIsM algorithm is great in the first and becomes smaller gradually with the increasing value of *min_sup*, the reason is that the total frequent itemsets are decreasing significantly accompanied with large value of *min_sup*. Compared with dataset *T10.I4.D1000K*, the time cost of *T30.I20.D1000K* on MFIs mining process is much more, the reason is that the itemsets in dense dataset *T30.I20.D1000K* are more likely frequent for their larger *support* value.

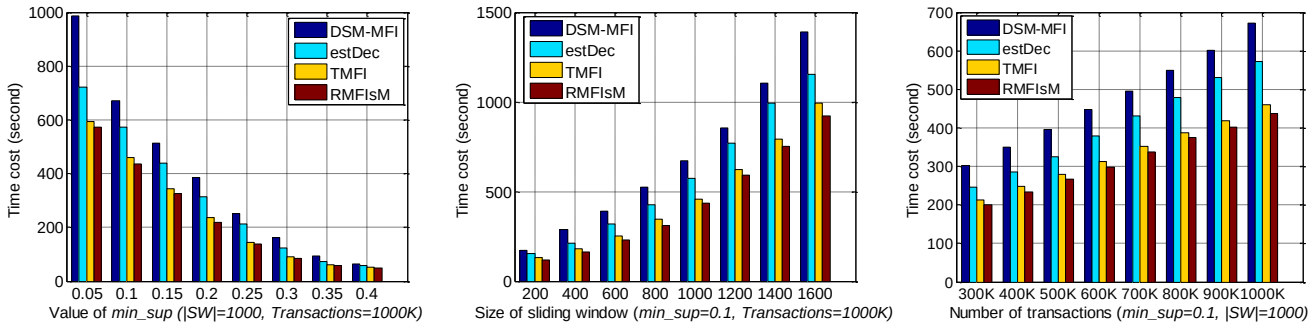
It can be seen from Figures 3-b and 4-b that with the increasing size of sliding window, the time cost of the compared four methods shows an increasing trends, the reason is that the number of frequent itemsets is rising rapidly as |*SW*| is becoming larger gradually. The time cost of our proposed RMFIsM method is the lowest of the four methods, and the time cost on *T30.I20.D1000K* is much larger than that on *T10.I4.D1000K*.

We can obviously see from Figures 3-c and 4-c that the time cost of compared four methods is increasing with the increased number of transactions, the reason is that the frequent itemsets increase gradually when the number of transactions is rising. The time cost of RMFIsM method is less than DSM-MFI, estDec and TMFI methods, and the time cost on *T30.I20.D1000K* is much more than that on *T10.I4.D1000K*.



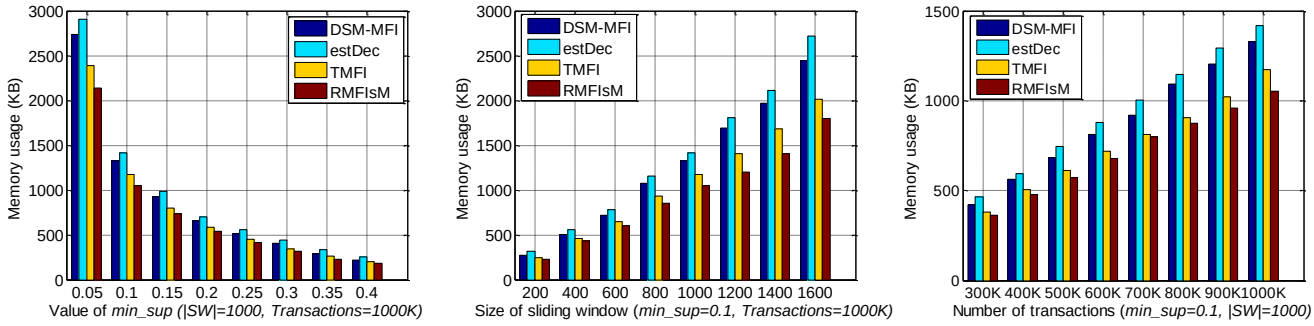
a) Different *min_sup*. b) Different sizes of sliding window. c) Different numbers of transactions.

Figure 3. Time cost on sparse dataset *T10.I4.D1000K*.



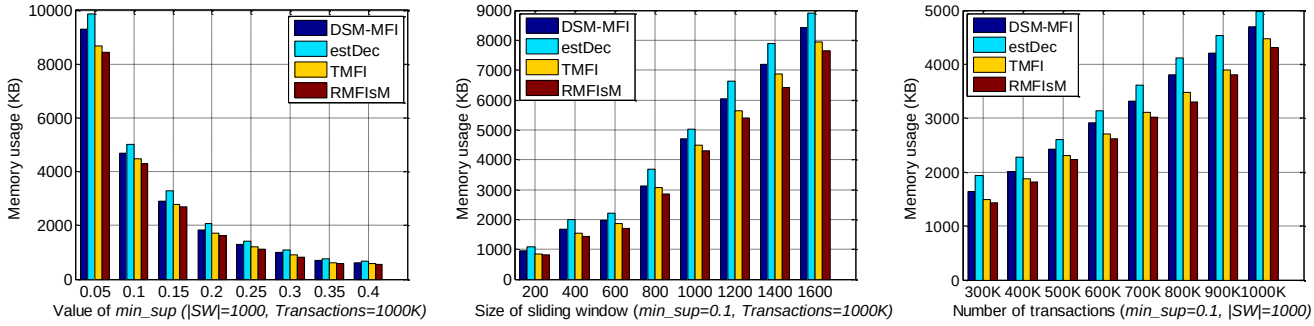
a) Different *min_sup*. b) Different sizes of sliding window. c) Different numbers of transactions.

Figure 4. Time cost on dense dataset *T30.I20.D1000K*.



a) Different *min_sup*. b) Different sizes of sliding window. c) Different numbers of transactions.

Figure 5. Memory usage on sparse dataset *T10.I4.D1000K*.



a) Different *min_sup*. b) Different sizes of sliding window. c) Different numbers of transactions.

Figure 6. Memory usage on dense dataset T30.I20.D1000K.

5.2. Memory Usage for RMFIsM Method

The memory usage is an important factor to measure the efficiency of our proposed RMFIsM method. The experiment to test the peak memory usage is also conducted with different value of *min_sup*, different size of sliding window and different number of transactions, the parameters used in this experiment is same with that in subsection 5.1, and the experimental results are shown in Figures 5 and 6.

We can see from Figures 5-a and 6-a that with the increasing value of *min_sup*, the peak memory usage of the compared four algorithms shows a decreasing trend. It is owing to that the number of frequent 1-

itemsets is decreasing gradually with the larger value of *min_sup*, therefore, the number of intermediate generated itemsets in *MFIs* mining process is also reduced much. The peak memory usage of our proposed RMFIsM method is lowest of the four methods, the reason is that the infrequent itemsets have been discarded in the beginning of RMFIsM method, so the meaningless "extension" operation hasn't been conducted to occupy the additional memory storage. Compared with sparse dataset *T10.I4.D1000K*, the memory usage of MFIs mining process on dense dataset *T30.I20.D1000K* is much more.

Figures 5-b and 6-b show that the peak memory usage of the compared four methods grows up gradually with the increasing size of sliding window, the reason is that the number of frequent itemsets becomes much larger with the extending size of sliding window. The peak memory usage on sparse dataset *T10.I4.D1000K* is much smaller than on dense dataset *T30.I20.D1000K*.

We can see from Figures 5-c and 6-c that the peak memory usage of RMFIsM, DSM-MFI, estDec and TMFI methods is increasing smoothly with the increasing number of transactions and the occupied peak memory usage is linearly related to the number of transactions. In the compared four methods, the peak memory usage of RMFIsM method is lower than estDec, TMFI and DSM-MFI methods in a certain extent. The peak memory usage on dense dataset *T30.I20.D1000K* is also much larger than that on sparse dataset *T10.I4.D1000K*.

Table 2. Accuracy rate of RMFIsM method.

Dataset min_sup	T10	T30	Dataset |SW|	T10	T30	Dataset Transactions	T10	T30
0.05	87.2%	89.6%	200	91.3%	92.2%	300K	92.1%	93.4%
0.1	92.3%	93.4%	400	91.6%	92.6%	400K	92.4%	93.2%
0.15	95.2%	96.1%	600	91.9%	92.8%	500K	92.3%	93.5%
0.2	96.4%	96.9%	800	92%	93.1%	600K	92.2%	93.5%
0.25	96.8%	97.3%	1000	92.3%	93.5%	700K	92.4%	93.3%
0.3	97.1%	97.5%	1200	92.5%	93.6%	800K	92.1%	93.4%
0.35	97.2%	97.6%	1400	92.7%	93.7%	900K	92.2%	93.2%
0.4	97.3%	97.8%	1600	92.8%	93.9%	1000K	92.3%	93.5%

5.3. Accuracy Rate of RMFIsM Method

The accuracy rate of our proposed RMFIsM method is also tested with different value of *min_sup*, different size of sliding window and different number of transactions, the set of experimental parameters is same with subsection 5.1 and the experimental result is shown in Table 2.

We can see from Table 2 that with the arising value of *min_sup*, the accuracy rate of the mining results is improving slowly both on datasets *T10.I4.D1000K* and *T30.I20.D1000K*, the reason is that the number of frequent itemsets shows a decreasing trend with the increasing value of *min_sup*, which results the influence of infrequent itemsets disappearing gradually. Furthermore, with the increasing size of sliding window, the accuracy rate of RMFIsM method is in rising trend, and the accuracy rate is relatively stable in general. Moreover, the accuracy rate of RMFIsM method is smooth between 92.1% to 92.4% on sparse dataset *T10.I4.D1000K* and between 93.2% to 93.5% on dense dataset *T30.I20.D1000K* with the increasing number of transactions, it is obvious that the number of transactions is a small factor that impact the accuracy rate of *MFIs* mining. The accuracy rate result indicates that our proposed RMFIsM method is suitable for mining the maximal frequent itemsets over online data streams under the larger value of *min_sup*.

6. Conclusions

It is often difficult to quickly mine the recent frequent itemsets over huge scale of data streams. In this paper, we propose an improved approach called RMFIsM to mine the maximal frequent itemsets instead of to mine all frequent itemsets. We first construct two matrixes to store the data information of each transaction and the information of frequent 1-itemsets. The frequent (p+1)-itemsets are mined by the "extension" process of frequent p-itemsets, the current maximal frequent itemsets are stored into *MFIs_L* and each sub-itemsets of frequent long itemsets are moved out from *MFIs_L*. Through the compared experimental with DSM-MFI, estDec and TMFI methods, it can be easily found that our proposed RMFIsM method is more effective both in time cost and memory usage, and the accuracy rate of *MFIs* mining is also very high.

References

[1] Calders T., Dexters N., Gillis J., and Goethals B., "Mining Frequent Itemsets in A Stream," *Information Systems*, vol. 39, pp. 233-255, 2014.

[2] Chang J. and Lee W., "Finding Recent Frequent Itemsets Adaptively Over Online Data Streams," *in Proceedings of 9th International Conference on Knowledge Discovery and Data Mining*, Washington, pp. 487-492, 2003.

[3] Deng Z., "Diffnodesets: An Efficient Structure for Fast Mining Frequent Itemsets," *Applied Soft*

Computing, vol. 41, pp. 214-223, 2016.
[4] Deypir M. and Sadreddini M., "A Dynamic Layout of Sliding Window for Frequent Itemset Mining Over Data Streams," *Journal of Systems and Software*, vol. 85, no. 3, pp. 746-759, 2012.
[5] Deypir M., Sadreddini M., and Tarahomi M., "An Efficient Sliding Window Based Algorithm for Adaptive Frequent Itemset Mining over Data Streams," *Journal of Information Science and Engineering*, vol. 29, no. 5, pp. 1001-1020, 2013.
[6] Guidan F. and Shaohong Y., "A Frequent Itemsets Mining Algorithm Based on Matrix in Sliding Window Over Data Streams," *in Proceedings of 3rd International Conference on Intelligent System Design and Engineering Applications*, Hong Kong, pp. 66-69, 2013.
[7] Han M., Ding J., and Li J., "TDMCS: An Efficient Method for Mining Closed Frequent Patterns over Data Streams Based on Time Decay Model," *The International Arab Journal of Information Technology*, vol. 14, no. 6, pp. 851-860, 2017.
[8] Li H., Lee S., and Shan M., "Online Mining (Recently) Maximal Frequent Item sets Over Data Streams," *in Proceedings of 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, Tokyo, pp. 11-18, 2005.
[9] Lin M., Hsueh S., and Wang C., "Interactive Mining of Frequent Patterns in A Data Stream of Time-Fading Models," *in Proceedings of 8th International Conference on Intelligent Systems Design and Applications*, Kaohsiung, pp. 513-518, 2008.
[10] Mao G., Wu X., Zhu X., Chen G., and Liu C., "Mining Maximal Frequent Itemsets From Data Streams," *Journal of Information Science*, vol. 33, no. 3, pp. 251-262, 2007.
[11] Nori F., Deypir M., and Sadreddini M., "A Sliding Window Based Algorithm For Frequent Closed Itemset Mining Over Data Streams," *Journal of Systems and Software*, vol. 86, no. 3, pp. 615-623, 2013.
[12] Shin S., Lee D., and Lee W., "CP-Tree: An Adaptive Synopsis Structure for Compressing Frequent Itemsets Over Online Data Streams," *Information Sciences*, vol. 278, pp. 559-576, 2014.
[13] Yang J., Wei Y., and Zhou F., "An Efficient Algorithm for Mining Maximal Frequent Patterns over Data Streams," *in Proceedings of 7th International Conference on Intelligent Human-Machine Systems and Cybernetics*, Hangzhou, pp. 444-447, 2015.

Saihua Cai is a Ph.D. student in College of Information and Electrical Engineering, China Agricultural University, China. He received the MS degree from Jiangsu University, China, in 2016. His major research interests include uncertain data management, data mining, outlier detecting and software testing.

Shangbo Hao is a Master Student in College of Information and Electrical Engineering, China Agricultural University, China. His research interests include pattern mining and outlier detecting.

Ruizhi Sun is a Full Professor in College of Information and Electrical Engineering, China Agricultural University, China. He received his Ph.D. degree in Computer Science and Technology from Tsinghua University, Beijing, China, in 2003. His major research interests include agricultural data acquisition and processing technology, computer network and applications, workflow management and cloud computing.

Gang Wu is an associate professor in Secretary of Computer Science Department, Tarim University, China. His research interests mainly involve agriculture information processing technology, data mining, agricultural remote sensing application.