I.J. Information Technology and Computer Science, **2016, 10, 63-71**
Published Online October 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.10.08

Mining Frequent Itemsets with Weights over Data Stream Using Inverted Matrix

Long Nguyen Hung*
E-mail: ntthlong@gmail.com

Thuy Nguyen Thi Thu*
*Informatics Department, Vietnam University of Commerce, Hanoi, Vietnam
E-mail: NguyenTthuthuy@gmail.com

***Abstract*—**In recent years, the mining research over data stream has been prominent as they can be applied in many alternative areas in the real worlds. In this paper, we have proposed an algorithm called MFIWDSIM for mining frequent itemsets with weights over a data stream using Inverted Matrix [10]. The main idea is moving data stream to an inverted matrix saved in the computer disks so that the algorithms can mine on it many times with different support thresholds as well as alternative minimum weights. Moreover, this inverted matrix can be accessed to mine in different times for user's requirements without recalculation. By analyzing and evaluating, the MFIWDSIM can be seen as the better algorithm compared to WSWFP-stream [9] for mining frequent itemsets with weights over data stream.

***Index Terms*—**Data mining, frequent itemset with weight, data stream, sliding window, inverted matrix.

I. Introduction

Data mining has been interested by many researchers, and its applications are successful applied to many areas in society. Nowadays, in many applications, data is not performed as static data. They are dynamic, continue, and sometimes they are extended continuously into undefined upper/lower boundary [1,3,13,14,15,16]. This dynamical data can be called as a data stream. The examples are very popular in the real world such as data in a network traffic analysis, in a web click stream, in a network of intrusion detection, or in an on-line transaction analysis. There are three challenges in mining over data stream: (1) Finding the frequent itemsets in the large sphere in power function representation); (2) explosion of data as it is updated and extended continuously while the used memory has been limited; (3) and the performed time needs to be as fast as possible

Mining frequent itemsets or mining frequent itemsets with weights is one of basic duties in data mining. The traditional methods used for frequent itemsets with weights in static data cannot be used for data stream as its disadvantages of memory space and performances.

In [12], Tsai P. S. M. has firstly proposed a new approach of mining frequent itemsets over data stream using weighted sliding window model. He also has proposed two algorithms based on Apriori so-called as WSW and improved WSW-Imp algorithms.

In [9], it has proposed WSWFP-stream according to FP-growth [6,7] to improve the time and the saving memory compared to the ones derived from Tsai P. S. M. [12].

In this paper, basing on the idea of Inverted Matrix algorithm proposed by [10] (in which data can be transferred to be represented in transactional array), we proposes an algorithms of MFIWDSIM. This algorithm uses the inverted matrix and an idea based on FP-growth [6,7] for mining frequent itemsets with weights over data stream.

This paper focus on the following sections. In section II, we present related works. Section III introduces the problem of mining frequent itemsets with weights over data stream using inverted matrix. Section IV shows a new proposed algorithm, MFIWDSIM (Mining Frequent Itemsets with Weights over Data Stream using Inverted Matrix). Last section gives the conclusions and further works.

II. Related Works

Recently, data mining has become the popular research interested by many worldwide researchers. In [4], the term of frequent itemsets with weights over static database was first proposed using MINWAL algorithm. In MINWAL, the authors defined a support level with weights. The value of itemset's support with weights has been calculated by a product of an itemset's support and the average of sub-items' weights. The challenge in here is related to Apriori property [2] using downward closure property. This property is broken as if the alternative itemsets will have alternative weights respectively. It means that the sub-items of the frequent itemsets with weights might not be the frequent sub-itemsets with weights. To keep the Apriori property, MINWAL used k-support bound concept. That means the support value for the itemset candidates generated in the k^{th} level should be greater than or equal to k-support bound. However, the use of k-support bound in [4] to prune the searching sphere might cost the time.

In [8], it proposed an algorithm of AWFIMiner for mining frequent itemsets with adaptive weights in a static

database. This algorithm is improved from AWFPM algorithm, in which a new measurement unit is used effectively for pruning index itemsets compared to the one mining with adaptive weights.

In recent years, mining over stream data has been an important research. There are many researches on frequent itemsets over data stream using alternative models such as Landmark model [11], Titled-time window model [5] and Sliding window model [12].

The landmark model [11] considers all data in one window. It involves all transactions from a fixed point of past time to present, and these transactions are treated the same.

The tilted-time window model in [5] is a modification of the landmark model. It considers the data at the time of a system starting up to present. The processing time is divided into alternative time slots, and the data is split into different batches by the time. A batch, that is closer to present, has been assigned a higher of weight compared to the old one which has to be a fine granularity [5].

In [12], Tsai P. S. M. has introduced an approach for mining frequent item-sets over a data stream based on the weighted sliding window model. Two algorithms of WSW and WSW-Imp algorithms are given which are based on Apriori property (downward closure property) [2]. WSW-Imp algorithm is an improvement of the first one. This means if X_p and X_q are two (k-1)-combinational itemsets in order to generate k-itemsets candidates of c used the combination method in Apriori algorithm, then the support with weight c will be greater than the supports with weights of X_p and X_q.

In [9], it has proposed WSWFP-stream algorithm basing on FP-growth [6,7] to improve effectively on memory space and time compared to both algorithms derived from Tsai P. S. M. [12]. To achieve a better performance, a new idea has been generated basing on representing data on the inverted matrix [10]. An algorithm of MFIWDSIM is proposed with use both of inverted matrix [10] and FP-growth approach [6, 7].

III. Model of Mining Frequent Itemsets with Weights over Data Stream

The following content describes the related definitions to problem of mining frequent itemsets over data streams derived from Tsai P. S. M. [12].

Given I as itemset, $I=\{i_1,i_2,\ldots,i_M\}$. A sub-itemset $X \subseteq I$, includes k alternative items so-called as *k-itemset* or an itemset with the length of k.

For simplification, an itemset $\{i_1,i_2,\ldots,i_q\}$ would be written as $i_1i_2\ldots i_q$. For example, the itemset {a,b,c} is replaced by *abc* in short. A transaction is a tuple $t=(TID,X)$ where *TID* is an identification index, and X is an itemset. A database of transaction DT is a set of transaction in the process.

A stream of transactions DS is an infinite range of transactions, $DS=\{ti_1,ti_2,\ldots,ti_m,\ldots\}$ meanwhile t_{ij}, $i=1,2,\ldots$; $j=1,2,\ldots$ is a transaction at time T_i. A sliding window W_{T_i} over data stream is a set of batches of transactions considered at the time T_i. Assume that at the time of T_i (i=1,2,…), the sliding W_{T_i} is divided into N batches of B_{ij} (i=1,2,…; j=1,2,…,N). Each batch is assigned with individual positive real number of weight as $\alpha_j, 0<\alpha_j<1, \sum_{j=1}^{N}\alpha_j = 1$. This means every itemsets in the batch's transactions will have the same weight.

Definition 1. The support with weight of an itemset X over data stream DS at the time T_i is *SWsupp(X)*, estimated by following equation:

$$SWsupp(X) = \sum_{j=1}^{N} F_{ij}(X) \times \alpha_j \quad (1)$$

Where $F_{ij}(X)$ is the frequency of X in j^{th} batch at the time T_i.

Definition 2. The support with minimum weights over data stream DS at T_i is calculated as:

$$\gamma = \mu \times \sum_{j=1}^{N} Card(B_{ij}) \times \alpha_j \quad (2)$$

Where $Card(B_{ij})$ is number of transactions of j^{th} batch at the time T_i, and $\mu (0<\mu<1)$ is the threshold of minimum support defined by user.

Definition 3. At the time T_i, given itemset of $X \subseteq I$, with the minimum weight of γ, X is so-called as frequent itemset with weight over data stream DS using sliding window model if:

$$SWsupp(X) \geq \gamma \quad (3)$$

Therefore, X can be seen as satisfied γ. Otherwise, X is not satisfied γ.

Table 1. Data stream at the time T_1

Tid	Trans.	Batch	Weight
T1	b c e	B_{13}	0.2
T2	a b c		
T3	c d e		
T4	a c e	B_{12}	0.3
T5	b c		
T6	c e		
T7	a b e		
T8	a b c	B_{11}	0.5
T9	c e		
T10	c d e		

W_{T_1}

Example 1: Given data in Table 1. Window W_{T_1} of at the time T_1 includes 10 transactions divided into 3 batches of B_{11}, B_{12}, B_{13}. The weights are

$\alpha_1 = 0.5, \alpha_2 = 0.3, \alpha_3 = 0.2$ respectively, and the minimum support threshold is $\mu = 20\%$.

We have, number of transactions in batches as:

$$Card(B_{11}) = 3, Card(B_{12}) = 4, Card(B_{13}) = 3.$$

The frequency of "ab" in batches as:

$$F_{11}(ab) = 1, F_{12}(ab) = 1, F_{13}(ab) = 1.$$

Therefore, the support with weight of itemset of "ab" as:

$$SWsupp("ab") = 1 \times 0.5 + 1 \times 0.3 + 1 \times 0.2 = 1.0$$

The minimum support at the time T_1:

$$\gamma = \mu \times \sum_{j=1}^{N} Card(B_{ij}) \times \alpha_j = 20\% \times (3 \times 0.5 + 4 \times 0.3 + 3 \times 0.2) = 0.66$$

As $SWsupp("ab") = 1.0 > \gamma = 0.66$ then "ab" is frequent itemset with weight over data stream. On the other words, itemset of "ab" is satisfied γ.

The frequency of "be" in batches as:

$$F_{11}(be) = 0, F_{12}(be) = 1, F_{13}(be) = 1.$$

Therefore, the support of "ab" itemset as:

$$SWsupp("be") = 0 \times 0.5 + 1 \times 0.3 + 1 \times 0.2 = 0.5$$

As $SWsupp("be") = 0.5 < \gamma = 0.66$ then "be" is not a frequent itemset with weight over data stream. Or itemset "be" is not satisfied γ.

Definition 4. At the time T_i, Mining frequent itemsets with weight over data stream means to find a set of *WSWFI* including all frequent itemsets with weight. This means as finding the following set:

$$WSWFI = \{X / X \subseteq I, SWsupp(X) \geq \gamma\}$$

The following lemma, derived from [2], shows a characteristic of the frequent itemset over data stream, which is always satisfied Apriori property. This will be a basic knowledge for mining frequent itemset with weight over data stream.

Lemma 1 [12]. If X is a frequent itemset with weight over data stream *DS* then all sub-itemsets $Y \subseteq X$ will be the frequent itemsets with weights over data stream of *DS*.

IV. Mining Frequent Itemset with Weight Algorithm over Data Stream Using Inverted Matrix

In this section, we propose MFIWDSIM algorithm for mining frequent itemset with weight algorithm over data stream using inverted matrix. MFIWDSIM algorithm is built basing on the transferring data to the Inverted Matrix derived from [10] and the FP-growth approach [6,7]. The MFIWDSIM includes two phases: The first is building inverted matrix, the second phase is mining on the inverted matrix.

A. Building inverted matrix

How to build

By applying the building inverted matrix method in [10], MFIWDSIM re-organizes data (transactions in the batches) in current window over data stream into inverted matrix saved in computer memory. The organizing data in inverted matrix will help the user in mining by accessing to array addresses of data. On the other hand, we can mine the data much time as we can with alternative support thresholds as it is already saved in computer memory.

The algorithm will save data in computer memory with two components as data indexes and an inverted matrix.

- Data Index: Use Integer numbers to assign to single items after sorting in alphabet order.
- Inverted Matrix: This matrix includes a set of rows assigned with single indexed items. Each cell (itemset) has three components as: the first is a frequency of attending in the batch, the second and the third components are the row and the column of the next itemset in the matrix. In case, if it is a last item then the second and the third will be nulls $(\varnothing, \varnothing)$.

Building inverted matrix algorithm use twice scanning on the data stream.

The first scanning: Sort the transaction in alphabet order for items. Then, assign index values for single-ordering items.

The second scanning: Scanning transaction in each batch and transferring data into inverted matrix can be done as follows:

- With the index: Define the location for the first item in the transaction in the batch.
- With each cell in inverted matrix: Find the first empty cell (from left to right) at the respective row of first index item. Assign this cell with three components of: frequency of attending in batch SC_j, and two others as row, column of r_{next}, c_{next} formed as:

$$[SC_j]\ (r_{next},c_{next}) \qquad (*)$$

Where SC_j=1, if itemset is attended in j^{th} batch. In case, there is a same item in the next transaction (same batch), then the SC_j=SC_j+1. The (r_{next}, c_{next}) is row and column addresses of the cell for the next itemset in the same transaction. This address will be null if it is a last item in the transaction (assigned as (Ø,Ø)).

The assigning works for cells will be repeated until we fill the addresses up for the last item in the transaction in batch.

Do the same for the next batch of a data stream.

Building Inverted Matrix Algorithm

Algorithm 1: Building Inverted Matrix.

Input: At time T_i;
Data stream *DS*;
Window W_{T_i} is divided into *N* batches of B_{ij} (i=1,2,…; j=1,2,…,*N*);

Output: IM_{T_i} is inverted matrix at T_i, is represented over data stream *DS*;

Method:

//First scanning

1. **Scan for** ($T \subseteq DS$)
2. **Begin**
3. In each transaction, sort itemsets in alphabet ordering;
4. Define all single item attended over data stream;
5. Build inverted matrix index component;
6. **End**;

//Second scanning

7. **Scan for** ($T \subseteq DS$)
8. **Begin**
9. Consider each transaction in the batch of B_{ij}, Assume that the alphabet ordering item list is: *List*=<a_1,a_2,…,a_t>;
10. Define address of (r_1,c_1) - first empty cell (from left to right) at the a_1 respectively in the batch of *j*;
11. Assign the item of a_1 at the address (formed in (*)) as : [SC_j] (r_2,c_2)
12. **For** i=2 **to** t **do**

//Consider all the rest items in the *List*.

13. **Begin**
14. Define address of (r_i,c_i) in order to write single item of a_i in *List*;
15. **If** ($i \le t-1$) **then** assign information of a_i in the batch of *j*, at address of (r_i,c_i) as [SC_j] (r_{i+1},c_{i+1})
16. **else** assign information of a_t in the batch of *j*, at address (r_t,c_t) as [SC_j] (Ø,Ø);
17. **End**;
18. **End**;
19. **Return** IM_{T_i} ;
20. **End**. // End of algorithm 1

Building inverted matrix example

Example 2. Consider the data in Table 1. The current window at time T_1 has 10 transactions in 3 batches of B_{11}, B_{12}, B_{13} and the batches weights as $\alpha_1 = 0.5, \alpha_2 = 0.3, \alpha_3 = 0.2$ respectively, the minimum support of $\mu = 20\%$.

Table 2. Index number in inverted matrix

Item	Loc
a	1
b	2
c	3
d	4
e	5

Table 3. Inverted Matrix to transfer data Table 1

Item	Loc	Transactional Array									
		B_{13}			B_{12}				B_{11}		
		1	2	3	4	5	6	7	8	9	10
a	1	[1] (2,2)			[1] (3,4)	[2] (2,5)			[1] (2,8)		
b	2	[1] (3,1)	[2] (3,2)		[1] (3,5)	[2] (5,6)			[1] (3,8)		
c	3	[1] (5,1)	[2] (Ø,Ø)	[3] (4,1)	[1] (5,4)	[2] (Ø,Ø)	[3] (5,5)		[1] (Ø,Ø)	[2] (5,8)	[3] (4,8)
d	4	[1] (5,2)							[1] (5,9)		
e	5	[1] (Ø,Ø)	[2] (Ø,Ø)		[1] (Ø,Ø)	[2] (Ø,Ø)	[3] (Ø,Ø)		[1] (Ø,Ø)	[2] (Ø,Ø)	

The building inverted matrix process as follows:

First Scanning: at each transaction, sort items in alphabet order, assign index number for each ordering items (see Table 2).

Second scanning: Table 3 shows the transferring data from a data stream into an inverted matrix. For example, to consider the first transaction of T1="bce", in Table 2, we have, a location of item of "b" as 2, item of "c" as 3, and item of "e" as 5. Therefore, there are three cells representing for three items located in the rows of 2, 3 and 5 respectively (in the inverted matrix). The first empty cell at the row 2 will be addressed as (2,1). The information in this cell will be [1] (2,2) (this means after considering the first transaction, item "b" is attended one time at the batch of B_{13}. Address of (2,2) shows the next cell of representing of item "c" in transaction T1).

Consider the first empty cell in row 3. Its address is (3,1). Therefore, the cell information is [1] (5,1) (frequency of item "c" is 1 at the batch B_{12}. (5,1) shows the address of the next item "e" in transaction T1). As considering to the first empty cell in row 5, its address is (5,1). So the cell information is [1] (Ø,Ø) (As item "e" is attended one time in the batch B_{11}, and item "e" is the last item in the transaction of T1).

Do the same for transactions of batches B_{12}, B_{11}. Note that when considering the first item in the first transaction of the next batch, its address should be saved in the cell having a respective row with the item's location. The column address of this cell is the first column of next batch.

Table 3 below shows the transferring data from data stream at the time T_1 to the inverted matrix. It is clear in the Table 3 that the representing data shows the necessary information to mine frequent itemsets with weights over inverted matrix instead of mining them over data stream. The detail will be shown in the following properties and building algorithm.

Using some properties given in [8], there are some properties for building trees based on FP-tree and mining on tree following the FP-growth [6,7].

Assume that IMFP-tree(x) is a IMFP-tree of item x. This tree is built basing on FP-tree [6,7].

Property 1. The highest of IMFP-tree(x) (for single-item of x) is: $M\text{-}ID_x$+1, where ID_x is the location of x.

For example, the highest of IMFP-tree(b) is 4 (see in Table 3).

Property 2. The distributional frequency of item in all batches is shown on the last cells in row of batch (respectively with the inverted matrix index number).

For example, as shown in Table 3, distributional frequency of item "a" in batches is [1,2,1] - one time in batch 3 (B_{13}), twice in batch 2 (B_{12}), and one time in batch 1 (B_{11}) - given in the cell address of (1,8), (1,5) – last cell in the batch, and (1,1). Distributional frequency of item "e" in batches is [2,3,2] given in cell address of (5,9), (5,6) and (5,2).

Property 3. Number of samples in building IMFP-tree(x) for x at an index position is number of cells the row in the matrix, where these cells do not reference to the null addresses.

For example, at row 3 in Table 3, the IMFP-tree(c) has 6 samples in building as there are 9 cells in the row including 3 of them referenced to the null addresses.

Property 4. Distributional frequency of item in batch in building IMFP-tree(x) (at j^{th} batch) equals 1 at j if $SC_j \geq 1$, or equals 0 in the rest positions.

For example, building IMFP-tree(a), the distributional frequency as: one sample with abc:0,1,0 in 3rd batch, 2 samples as ace:0,1,0 and abe:0,1,0 in 2nd batch, one sample abc:1,0,0 in 1st batch.

Property 5. When building MFP-tree for the next items (which had alphabet order), this tree do not contain the previous-alphabet items.

This means the next tree has a reducing size as its previous alphabet items are eliminated.

For example, building IMFP-tree(c) will not include items of "a" and "b", or building IMFP-tree(d) will not include items of "a", "b" and "c".

Property 6. There is no need to build the IMFP-tree(x) if x is the last item in alphabet order.

For example, the IMFP-tree (e) is not necessary to build as if "e" is last item and all cells' address reference to the null.

B. Mining the inverted matrix

Algorithm

After building a inverted matrix, the mining process will be performed according to alphabet ordering items. For each single-item satisfied γ, the algorithm is performed as follows: (1) Define the samples in inverted matrix; (2) Building IMFP-tree for each item according to FP-tree [6,7]. Note that at each node, there is a save of list of frequencies in each batch (see also in [8]); (3) Mining IMFP-tree for this item based on FP-growth [6,7] to find the frequent itemsets with weights; (4) Remove IMFP-tree and conditional tree (if existing) for the item. Repeat the same process for next items in the matrix.

Algorithm 2: Mining Inverted Matrix.
Input: At time T_i;
Inverted Matrix IM_{T_i};
Weighted table of batches $\alpha_j (j = 1,..,N)$;
Minimum support threshold over data stream μ;
Output: L is a list of frequent itemset with weight of inverted matrix IM_{T_i};

Method:
1. At time T_i: Consider window of W_{T_i}. Define number of transaction in the batch;
2. Update weighted table for batches;
3. Compute the support with minimum weight γ

based (2);

4. From inverted matrix IM_{T_i}, define C_1 (frequent itemset with weight) satisfied γ.
5. $L=C_1$;
6. **For** i:=1 **to** M **do**
   // M is number of items
7. **Begin**
8. For x at row i
9. **For** j:=1 **to** $Card(IM_{T_i})$ **do**
   // $Card(IM_{T_i})$ is index number of inverted mate
10. **If** (value of ($a_{ij} \neq \varnothing$) **then**
11. **Begin**
12. Follow the tracking of referenced cells (base (*)), we can find k samples:
    $Samp_x=\{S_1:f_1,\ldots f_N;\ldots;S_k:f_1,\ldots f_N\}$;
    // Where there is only one f_p=1 at the item located at p^{th}, f_p=0 in otherwise.
13. **End**;
14. Build IMFP-tree(x) from $Samp_x$;
15. Create a conditional tree for item x;
16. Build the list item candidates from conditional tree of x;
17. Build $FP(x)$ is a set of itemset satisfied γ;
18. $L=L\cup FP(x)$;
19. Remove IMFP-tree(x) and conditional tree of x;
20. **End**;
21. **Return** L;
22. **End**. // End of algorithm 2

Example of mining inverted matrix

Mining inverted matrix in Table 3 to find frequent itemset with weight over data stream can be seen as follows:

At time T_1: Consider window of W_{T_1}. Define number of transaction in batches and number of single items, update the weights.

Calculate the support with minimum weight γ according to (2):

$$\gamma = \mu \times \sum_{j=1}^{N} Card(B_{ij}) \times \alpha_j = 0.66$$

As property 2, items of "a", "b", "c", "d" and "e" have the distributional frequency as: <*a*:1,2,1;*b*:1,2,2;*c*:3,3,3;:*d*:1,0,1;*e*:2,3,2> and the support with weights respectively as:

<*a*:1.3;*b*:1.5;*c*:3.0;*d*:0.7;*e*:2.3>.

All items are satisfied γ. Therefore, there is no eliminated itemset. The list of single items as frequent itemset with weight over data stream will be $L=C_1=\{a,b,c,d,e\}$.

Therefore, we can build and mine the IMFP-tree for the items of "a", "b", "c" and "d" (there is no tree for "e" as it is the last item in the list-based property 5).

- *Building and Mining IMFP-tree(a).*

According to property 3, in Table 3, item "a" is located in row 1. There are 4 "not-empty" cells (these cells addresses do not reference to the next cell of null address). Therefore, the number of samples will be 4 to build IMFP-tree(a).

According to property 4, the samples in building tree will be: {*abc*:0,0,1;*ace*:0,1,0;*abe*:0,1,0;*abc*:1,0,0}. Then, the tree can be built in Fig. 1(a).

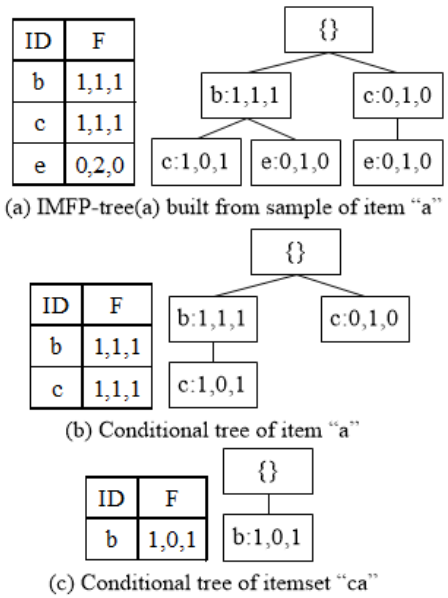

Fig.1. IMFP-tree(a) and conditional trees

From index table in Fig. 1(a), there are 2-itemsets (together with a") as <*ea*:0,2,0;*ca*:1,1,1;*ba*:1,1,1> and the support with weights respectively as <*ea*:0.6;*ca*:1.0;*ba*:1.0>. It is clear that item "ea" did not satisfied γ. Remove item "e" from IMFP-tree(a), we have the tree in Fig. 1(b). Update "ca" and "ba" into L, we have, $L=\{a,b,c,d,e,ca,ba\}$.

Continue mining by developing the 2-itemsets of "ca" and "ba". By mining conditional tree of "ba", we have a null tree. By mining the conditional tree of "ca", we have a tree in Fig. 1(c).

There is a 3-itemset candidate of {*bca*:1,0,1} and the support with weight of "bca" is <*bca*:0.7> satisfied γ. By mining the conditional tree of "bca" we have a null tree. Therefore, $L=\{a,b,c,d,e,ca,ba,bca\}$.

Remove IMFP-tree(a) and its conditional tree after mining.

- *Building and mining IMFP-tree(b).*

There are 5 samples in building IMFP-tree(b) (see in Table 3 at row 2). They:

{*bce*:0,0,1;*bc*:0,0,1;*bc*:0,1,0;*be*:0,1,0;*bc*:1,0,0} (not including "a" according to property 5). From the samples we have IMFP-tree(b) in Fig. 2(a).

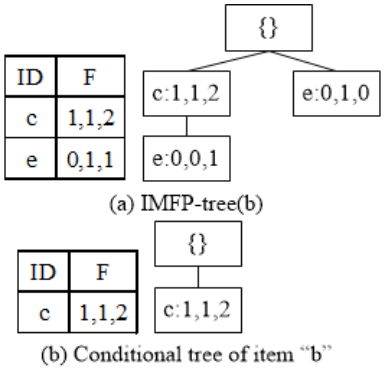

Fig.2. IMFP-tree(b) and conditional tree

From index table, we have 2-itemset together with "b" without "a" as <*eb*:0,1,1;*cb*:1,1,2> and the support with weights respectively as <*eb*:0.5;*cb*:1.2>. The itemset of "eb" is not satisfied γ. Remove item "e" from IMFP-tree(b) we have a conditional tree in Fig. 2(b). Update itemset of "cb" to *L*, we have *L*={*a*,*b*,*c*,*d*,*e*,*ca*,*ba*,*bca*,*cb*}. Mining the condition tree of "cb", we have null tree. Therefore, *L*={*a*,*b*,*c*,*d*,*e*,*ca*,*ba*,*bca*,*cb*}.

Remove IMFP-tree(b) and its conditional tree after mining.

- *Building and mining IMFP-tree(c).*

Samples in building IMFP-tree(c) is 6 (as seen in Table 3, row 3) as:

{*ce*:0,0,1;*cde*:0,0,1;*ce*:0,1,0;*cd*:0,1,0;*cde*:1,0,0;*ce*:1,0,0}.

From these samples we can build IMFP-tree(c) in Fig. 3(a).

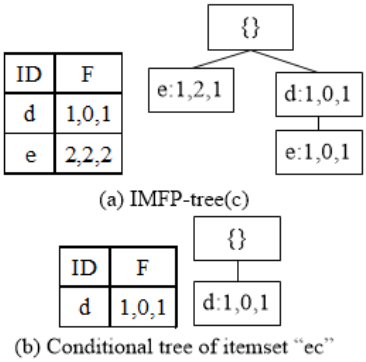

Fig.3. IMFP-tree(c) and its conditional tree

From index table, there are 2-itemset together with "c" as <*ec*:2,2,2;*dc*:1,0,1> and the support with weight respectively as <*ec*:2.0;*dc*:0.7>. These itemsets are satisfied γ. Update "ec" and "dc" into *L*, we have L={*a*,*b*,*c*,*d*,*e*,*ca*,*ba*,*bca*,*cb*,*ec*,*dc*}.

Mining conditional tree of "dc", we have a null tree. Then, by mining conditional tree of "ec", we have a conditional tree in Fig. 3(b).

There is a 3-itemset candidate of {*dec*:1,0,1} and support with weight as "dec" là <*dec*:0.7> satisfied γ. By mining conditional tree of itemset "dec" we have an null tree. Update itemset "dec" into *L* we have:

L={*a*,*b*,*c*,*d*,*e*,*ca*,*ba*,*bca*,*cb*,*ec*,*dc*,*dec*}.

Remove IMFP-tree(c) and its conditional tree after mining.

- *Building and Mining IMFP-tree(d).*

Number of samples in building IMFP-tree(d) is 2 (see Table 3, row 4) as: {*de*:0,0,1;*de*:1,0,0}. The tree can be seen in Fig. 4.

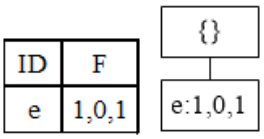

Fig.4. IMFP-tree(d)

There is a 2-itemset together with "d" is <*ed*:1,0,1> and the support with weight is <*ed*:0.7> satisfied γ. Therefore,

L={*a*,*b*,*c*,*d*,*e*,*ca*,*ba*,*bca*,*cb*,*ec*,*dc*,*dec*,*ed*}.

Remove IMFP-tree(d) after mining.

Result of frequent itemsets (with weights) over data stream at time T_1 is:

$$L=\left\{\begin{matrix} a:1.3;b:1.5;c:3.0;d:0.7;e:2.3;ca:1.0;ba:1.0; \\ bca:0.7;cb:1.2;ec:2.0;dc:0.7;dec:0.7;ed:0.7 \end{matrix}\right\}$$

C. Update inverted matrix

Algorithm

The transferring data from data stream to the inverted matrix is quite easy when we update data (eliminate respective transactions in the old batches and update the new ones). This work can be done in the inverted matrix without updating all data over data stream.

At time T_{i+1}, to clear old fashioned transactions in the inverted matrix we only need to remove the old batch of (B_{iN}), update inverted matrix index, change all the column index of cells which have not null addresses referenced to other cells in the inverted matrix, update the new batch of ($B_{i+1,1}$), change all batch's transactions into the inverted matrix.

Algorithm 3: Update inverted matrix.
Input: At time T_i.
Inverted matrix IM_{T_i};
Newest batch of $B_{i+1,1}$;
Output: $IM_{T_{i+1}}$ is inverted matrix at time T_{i+1};
Method:
1. At time T_{i+1}:
2. $CT=Card(B_{iN})$;
3. Remove current batch of B_{iN};

4. Update inverted matrix indexes for the rest batches;
   // Keep the rows' indexes and only change the columns' indexes where their addresses are not null.
5. **For** i:=1 **to** M **do** // M is item's position in the inverted matrix
6. **For** j:=1 **to** *LCI* **do** // *LCI* is the last column's index in the inverted matrix
7. **If** $(a_{ij} \neq \varnothing) and (c_{next} \neq \varnothing)$ **then** $c_{next}=c_{next}-CT$;
   // a_{ij} is a cell in inverted matrix.
8. Update the newest batch of $B_{i+1,1}$, and then update the inverted matrix indexes;
9. Update batches' weights;
10. **Return** $IM_{T_{i+1}}$;
11. **End**. // End of algorithm 3

Example of updating inverted matrix

The main tasks for updating in time T_2 are: Remove the old batch of (B_{13}), update the new of (B_{21}) and update the weights (see in Table 4).

Table 4. Data stream at time T_2

Tid	Trans.	Batch	Weight
T4	a c e	B_{23}	0.2
T5	b c		
T6	c e		
T7	a b e		
T8	a b c	B_{22}	0.3
T9	c e		
T10	c d e		
T11	a b c d	B_{21}	0.5
T12	b c d		
T13	d e		
T14	a b c e		
T15	b c e		

W_{T_2}

At time T_2, after removing the old batches B_{13}, batch B_{12} to B_{23}, batch B_{11} to B_{22}, the number of inverted matrix indexes in batches B_{23} , B_{22} will reduce to 3 (number of transactions in batch B_{13}).

We will have Table 5. Update the new batch B_{21}, and update the matrix indexes we will have Table 6.

Table 5. Inverted matrix after removing batch B_{13} and updating index number

Item	Loc	Transactional Array										
		B_{13}			B_{23}				B_{22}			
		~~1~~	~~2~~	~~3~~	1	2	3	4	5	6	7	
a	1	~~[1] (2,2)~~			[1] (3,1)	[2] (2,2)			[1] (2,5)			
b	2	~~[1] (3,1)~~	~~[2] (3,2)~~		[1] (3,2)	[2] (5,3)			[1] (3,5)			
c	3	~~[1] (5,1)~~	~~[2] (Ø,Ø)~~	~~[3] (4,1)~~	[1] (5,1)	[2] (Ø,Ø)	[3] (5,2)		[1] (Ø,Ø)	[2] (5,5)	[3] (4,5)	
d	4	~~[1] (5,2)~~							[1] (5,6)			
e	5	~~[1] (Ø,Ø)~~	~~[2] (Ø,Ø)~~		[1] (Ø,Ø)	[2] (Ø,Ø)	[3] (Ø,Ø)		[1] (Ø,Ø)	[2] (Ø,Ø)		

Table 6. Inverted matrix at time T_2 after updating batch of B_{21}

Item	Loc	Transactional Array											
		B_{23}				B_{22}			B_{21}				
		1	2	3	4	5	6	7	8	9	10	11	12
a	1	[1] (3,1)	[2] (2,2)			[1] (2,5)			[1] (2,8)	[2] (2,10)			
b	2	[1] (3,2)	[2] (5,3)			[1] (3,5)			[1] (3,8)	[2] (3,9)	[3] (3,10)	[4] (3,11)	
c	3	[1] (5,1)	[2] (Ø,Ø)	[3] (5,2)		[1] (Ø,Ø)	[2] (5,5)	[3] (4,5)	[1] (4,8)	[2] (4,8)	[3] (Ø,Ø)	[4] (5,10)	
d	4					[1] (5,6)			[1] (Ø,Ø)	[2] (Ø,Ø)	[3] (5,8)		
e	5	[1] (Ø,Ø)	[2] (Ø,Ø)	[3] (Ø,Ø)		[1] (Ø,Ø)	[2] (Ø,Ø)		[1] (Ø,Ø)	[2] (Ø,Ø)	[3] (Ø,Ø)		

V. Conclusions and Further Works

The paper has proposed MFIWDSIM algorithm for mining frequent itemsets with weights over data stream. This is based on the idea of transferring data from static database to inverted matrix derived from [10]. The improvement of this algorithm is that there are two components at each inverted matrix cell. The first shows a frequency of item in batch, and the second shows the referenced addresses for the next cell in the same transaction. Therefore, the frequency of each item can be

taken from last cell on the right at its position on the matrix without moving the data stream to the FP-tree as previous algorithms [9] done.

The proposed algorithm does not need to build FP-trees as in [9] for all data stream. This means all data stream is moved to inverted matrix for mining. From this, the samples are taken in order to build trees (FP-growth) for each item. As the items were increased in alphabet sorting list, therefore, building and mining the next items did not include the previous alphabet ordering items. This means the number of itemset candidates has been reduced significantly (the pruning tree is to make reducing of number candidates as well).

Number of scanning over data stream is twice as: Firstly, we need to sort items in alphabet order at each transaction, and create the index values for single items respectively. The second is to transfer data from stream (after sorting) to the inverted matrix.

Moreover, when data is transferred to inverted matrix, the updating information, or processing of itemsets seems to be easier doing at alternative times. It is because that we can do them all on the matrix instead of on the FP-trees, in where the data stream must be updated before doing each tasks.

With the conclusions given above, it seems that algorithms of MFIWDSIM will be the effective algorithms for interactional mining frequent itemset with weight over data stream using inverted matrix.

The trend for this paper is researching on the compact of inverted matrix as we can save it to computer memory. Other way might be an archive of calculating results at continuous times for the interaction mining.

REFERENCES

[1] Aggarwal C. In C. Aggarwal (Ed.), *Data Streams: Models and algorithms. Springer*, (2007).

[2] Agrawal R., Srikant, R., Fast Algorithms for Mining Association Rules. In: *20th Int'l. Conf. on Very Large Data Bases (VLDB)*, pp. 487–499, (1994).

[3] Aneri P., Chaudhari M. B., Frequent pattern mining of continuous data over data streams, *Int'l. Jour. for Technology Research Engineering*, Vol. 1, Issue 9, pp. 935-940, (2014).

[4] Cai, C.H., Fu, A.W.-C., Cheng, C.H., Kwong, W.W. (1998), Mining Association Rules with Weighted Items. In *Proceedings of Int'l. Database Engineering and Applications Symposium (IDEAS 1998)*, Cardiff, Wales, UK, July 1998, pp. 68–77, (1998) .

[5] Giannella C., Han, J., Pei, J., Yan, X., & Yu, P. S., Mining frequent patterns in data streams at multiple time granularities. In *H. Kargupta, A. Joshi, K.Sivakumar, & Y. Yesha (Eds.), Next generation data mining*, pp. 191–210, (2003).

[6] Han J., and Kamber M., *Data Mining: Concepts and Techniques*, Morgan Kanufmann, (2000).

[7] Han J., Pei, J., Yin, Y., Mao, R., Mining frequent patterns without candidate generation: a frequent-pattern tree approach, *Data Mining and Knowledge Discovery 8*, pp. 53–87, (2004).

[8] Hung Long Nguyen, An Efficient Algorithm for Mining Weighted Frequent Itemsets Using Adaptive Weights, *Int'l Jour. of Intelligent Systems and Applications*, Vol. 7, No. 11, pp. 41-48, (2015).

[9] Long Nguyen Hung, Thuy Nguyen Thi Thu, Giap Cu Nguyen, An Efficient Algorithm in Mining Frequent Itemsets with Weights over Data Stream Using Tree Data Structure, *Int'l Jour. of Intelligent Systems and Applications*, Vol. 7, No. 12, pp. 23-31, (2015).

[10] Mohammad El-Hajj, Osmar R. Zaïane, Inverted Matrix: Efficient Discovery of Frequent Items in Large Datasets in the Context of Interactive Mining, In: *Proc. 2003 Int'l. Conf. on Data Mining and Knowledge Discovery (ACM SIGMOD)*, pp. 109-118, August 24-27, 2003, (2003).

[11] Manku G., Motwani R. Approximate frequency counts over data streams. In: *Proceedings of the VLDB conference*, pp. 346–357, (2002).

[12] Tsai P. S. M., Mining frequent itemsets in data streams using the weighted sliding window model. *Expert Systems with Applications*, pp. 11617-11625, (2009).

[13] Vijayarani S., Sathya P., A survey on frequent pattern mining over data streams, *Int'l. Jour. of Computer Science and Information Tech. & Sec. (IJCSITS)*, Vol. 2., No. 5, pp. 1046-1050, (2012).

[14] Vikas K., Sangita., A review on algorithm for mining frequent itemset over data stream, *Int'l. Jour. of Data Advanced Research in Comp. Sci. and Software Engineering*, Vol 3., Issue 4, pp. 917-919, (2013).

[15] Wang J., Zeng Y., SWFP-Miner: An efficient algorithm for mining weight frequent pattern over data streams, *High Technology Letters*, Vol. 3, No. 3, pp. 289-294, (2012).

[16] Younghee K., Wonyoung K., Ungmo K., Mining frequent itemsets with normalized weight in continuous data streams, *Journal of Information Processing Systems*, Vol. 6, No. 1, pp. 79-90, (2010).

Authors' Profiles

Long Nguyen Hung is currently a lecturer at Faculty of Economic Information System, Vietnam Commercial University (VCU). He received his B.Sc. degree in Informatics from Hanoi University of Science in 1991, and his M.Sc. degree in Information Technology from Le Quy Don Technical University in 2002. His research interests include: Data Mining, Knowledge Discovery in Databases, Information Systems, and Database. Many his publications also are concentrated to these areas.

Thuy T. T. Nguyen was born in 1973 in Bacgiang, Vietnam. She graduated university in 1993 in Math. In 1999, she received MSc degree in Information Technology in Hanoi National University. She received PhD in Computer Science at The University of Hull, UK in 2011 respectively. From 2001 afterward, she joined to Vietnam University of Commerce, as a lecturer. Her interested research includes data mining, neural network, supervised/unsupervised learning techniques, and management information systems. Many her publications also are concentrated to these areas.