I.J. Intelligent Systems and Applications, **2015, 12, 23-31**
Published Online November 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2015.12.02

An Efficient Algorithm in Mining Frequent Itemsets with Weights over Data Stream Using Tree Data Structure

Long Nguyen Hung[*]
Email: ntthlong@gmail.com

Thuy Nguyen Thi Thu[*]
Email: NguyenTthuthuy@gmail.com

Giap Cu Nguyen[*]
*Informatics Department, Vietnam University of Commerce, Hanoi, Vietnam
Email: cunguyengiap@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 [20], a framework for mining frequent itemsets over a data stream is proposed by the use of weighted slide window model. Two algorithms of single pass (WSW) and the WSW-Imp (improving one) using weighted sliding model were proposed in there to solve the data stream problems. The disadvantage of these algorithms is that they have to seek all data stream many times and generate a large set of candidates. In this paper, we have proposed a process of mining frequent itemsets with weights over a data stream. Based on the downward closure property and FP-Growth method [8, 9] an alternative algorithm called WSWFP-stream has been proposed. This algorithm is proved working more efficiently regarding to computing time and memory aspects.

Index Terms—Data mining, frequent itemsets, data stream, weighted sliding window, weighted supports, tree data structure.

I. INTRODUCTION

Recent years, the data mining in particular to the mining over data stream has been concerned by many researchers [1,3-7,10-25], and it has been applied successfully in many real areas. As the processing data is no longer static, it becomes to dynamic, and sometimes has been extended continuously into undefined upper/lower boundary [1,3,4,5,6,8]. All these 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. The continuous data streams mentioned above are main causes of challenges in data mining [1] as: Large seeking sphere for frequent itemsets (in power function representation); explosion of data while the used memory has been limited; and the performed time (as fast as possible to extract the results). Take all into account, new efficient data mining algorithm over data streams is necessary.

Mining in frequent itemsets is a fundamental data mining task over a data stream. Almost traditional algorithms were used for the data mining frequent itemsets in static database. This allows retrieving each item more than one time in a database. However, the mining algorithms over data stream cannot access data stream more than one time as its continuousness property and the limitation of memory. Therefore, a necessary algorithm for mining over a data streams should be applied. Moreover, the algorithm efficient factors should be considered in applying data stream problems in aspect of the performed time and the memory.

In [20], Tsai P. S. M. has proposed a new approach for mining frequent itemsets over a data stream based on the weighted slide window model via two algorithms WSW and WSW-Imp (an improved one). Both of these algorithm are based the Apriori algorithm [2]. The algorithms based Apriori have *the downward closure property* of frequent itemsets. This means as any subsets of a frequent itemset is also a frequent itemset. The disadvantage of these kinds of algorithms as the need to generate and test too many candidate sub-itemsets and scan database many times. Therefore, the mining performance is seems to be low. The WSW-Imp algorithm (based on WSW) improves its performance by applying the following property: if two itemsets X_p and X_q are combined to form a new candidate itemset c by downward closure property, then the weighted support of c is not higher than the weighted supports of X_p and X_q.

In this paper, we have proposed a new efficient algorithm called WSWFP-stream (Window Sliding Weights Frequent Pattern over stream) for mining frequent itemsets over data streams. This is based on Tsai's model [20], but it is applied the FP-growth (Frequent Pattern) approach [8, 9]. The theoretical analysis and experiment show the better performance of WSWFP-stream than the WSW and WSW-Imp algorithms, regarding to saving in computing time and memory.

This paper will focus on the following sections. The related works can be seen in section II. In section III, we present an overview of the data mining frequent itemsets over a data stream using the weighted sliding window model. Section IV introduces a new proposed algorithm, WSWFP-stream, which is based on the weighted sliding window model. Section V shows the experiments and the discussion its results. Last section gives the conclusion and further works.

II. Related Works

There are many researches on frequent itemsets over data stream [1]. The popular mining researches can be seen as: Landmark model, titled-time window model, and sliding window model.

The landmark model [17] considers all data in one window. It involves all transactions from a fixed past time point to present, and treats them all similarly. In [17] Manku G. et al have proposed a model that counts the frequency of elements over the threshold defined by users. Their model differs from models based on Apriori by removing the candidate in generation phase. The proposal model, therefore, requires small main memory footprints. Base on Landmark model, Li Su et al in [13] has also introduced a model that deal with classification of association rules on a data stream by applying Lossy Counting algorithm. In [13], data structure was divided into three modules of BUFFER, TRIE, and SETGEN. Then, subsets of these transactions were enumerated along with their frequencies.

The tilted-time window model [7] is a modification of the landmark model. It considers the data at the time point of a system started up to present. The processing time is divided into alternative time slots, and the data is split into different batches by time. A batch, that closer to present, has been assigned a higher weight (to be a fine granularity [7]).

Differ from landmark model, the sliding model [5] focuses on new data backward to the fixed past time point. Moreover, the window size might be taken from given number of transactions. By using a compact data structure, the closed enumeration tree (CET), which monitors closed frequent itemsets as well as itemsets that form the boundary between the closed frequent itemsets and the rest of the itemsets, will be maintained a dynamically selected set of itemsets over a sliding-window. The cost of mining closed frequent itemsets over a sliding window in [5] was dramatically reduced to that of mining transactions, and they can be the possible causes boundary movements in the CET.

The other method of doing on frequent itemsets over data stream can be seen in [18]. In there, the authors have compared of use of Hoeffding trees and Hoeffding Option tree. The Hoeffding Option tree has shown the better under various memory limits as it added splitting options in the internal nodes to improve accuracy, stability and reduce ambiguity.

The new approach for mining frequent itemsets over a data stream based on the weighted slide window can be seen in [20]. Tsai P. S. M. has proposed two algorithms WSW and WSW-Imp used the downward closure property of frequent itemsets of Apriori [2] during the process.

Continue to the weighted sliding window model, in [24], Yong C. et al have proposed two algorithms of SWSS and SWSS-Imp to mine frequent sequential patterns. By building W-Tree to store frequent sequences, each node in W-Tree will has a W-List to maintain bitmaps of current sequence. As each sequence could be presented in each sliding window, the users can define the size of the sliding windows.

To inherit the advance in Tsai' approach [20], in this paper, we have proposed an improved algorithm called WSWFP-stream for mining frequent itemsets over data streams. This is based on Tsai's model [20], but it is applied the FP-growth (Frequent Pattern) approach [8, 9].

III. Mining Frequent Itemsets in Data Streams Using Tree Data Structure

Given *I* is a set of items, $I = \{i_1, i_2, ..., i_k\}$. A subset $X \subseteq I$ including *k* different items is called *k-itemset* or an itemset has length *k*. For simplification, an itemset $\{i_1,i_2,...,i_q\}$would be written as $i_1i_2...i_q$; for example, the itemset $\{a,b,c,d,e\}$ is replaced by *abcde* in short. A transaction is a tuple *t=(TID,X)* where *TID* is an identification index, and *X* is an itemset.

A stream of transactions *DS* is an infinite range of transactions, DS=$\{t_{i1},t_{i2},...,t_{im},...\}$ meanwhile t_{ij}, *i*=1,2,…; *j*=1,2,… is a transaction at time *i*. A window *W* in a transaction stream is a set of transactions between time points *i* and *j* (*j* > *i*).

In [20], Tsai P. S. M. has proposed a model for mining frequent itemsets over a data stream using weighted sliding window as follow.

Assume that at the time point T_i $(i = 1, 2, ...)$, a sliding window *W* is split into *N* batches W_{ij} and each batch W_{ij} is assigned by a different number α_{ij}, $0<\alpha_{ij}\leq 1$, (*i*=1,2…; *j*=1,2,…,*N*). A weight of a data batch is the number assigned to all itemsets of any transaction in this batch.

Definition 1: The support with weight of an itemset *X* is *WSWsupp(X)*, estimated by following equation:

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

Definition 2: The minimal support with weight of a data stream *DS* at time point T_i defined as:

$$\gamma = minsupp \times \sum_{j=1}^{N} \left| W_{ij} \right| \times \alpha_{ij} \quad (2)$$

where W_{ij} is the number of transaction in batch *j* at time point T_i and *minsupp* is the minimal support set by user for a data stream *DS*.

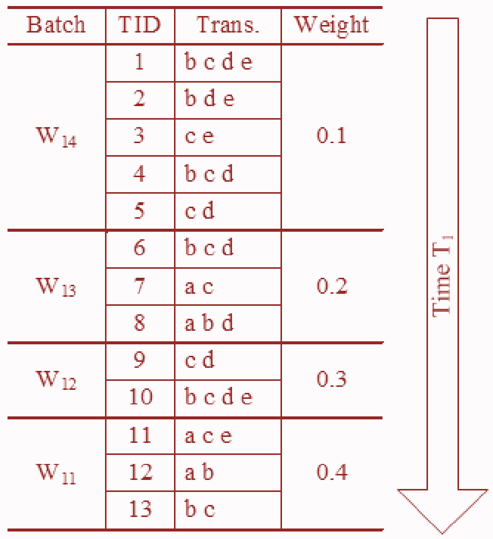
Definition 3: Given *k-itemset* $X \subseteq I$ and a minimal support with weight γ, *X* is called a frequent itemset with weight over a data stream *DS* bases on sliding window model if:

$$WSWsupp(X) \geq \gamma \qquad (3)$$

If that we can say the itemset *X* is satisfied by γ, otherwise, we can say the itemset *X* is not satisfied by γ.

For example: Given a data stream depicted in Table 1, the sliding window at time point T_1 has 13 transactions and is split into 4 batches as W_{11}, W_{12}, W_{13}, W_{14} with corresponding weights α_{11} =0.4, α_{12} =0.3, α_{13} =0.2, α_{14} =0.1, the minimal support with weights of *minsupp* is 20%.

Table 1. An Example of Data Stream At Time-Point T_1.



Batch	TID	Trans.	Weight
W_{14}	1	b c d e	0.1
	2	b d e	
	3	c e	
	4	b c d	
	5	c d	
W_{13}	6	b c d	0.2
	7	a c	
	8	a b d	
W_{12}	9	c d	0.3
	10	b c d e	
W_{11}	11	a c e	0.4
	12	a b	
	13	b c	

We have the weighted support of the itemset "*cd*" is:

$$WSWsupp("cd") = 0\times0.4 + 2\times0.3 + 1\times0.2 + 3\times0.1 = 1.1$$

The minimal weighted support at the time point T_1:

$$\gamma = minsupp \times \sum_{j=1}^{N} |W_{ij}| \times \alpha_{ij}$$
$$= 20\% \times (3\times0.4 + 2\times0.3 + 3\times0.2 + 5\times0.1) = 0.58$$

The $WSWsupp("cd") = 1.1 > \gamma = 0.58$, means "*cd*" is a weighted frequent itemset over the given data stream. In the other words, we say that the itemset "*cd*" is satisfied by γ.

Definition 4: Mining weighted frequent itemsets over a data stream *DS* using the sliding window model is finding a set *WSW* including all weighted frequent itemsets:

$$WSW = \{X / X \subseteq I, WSWsupp(X) \geq \gamma\}$$

Lemma 1: If *X* is a weighted frequent itemset then all subset of *X* are weighted frequent itemsets.

Proof .

We have

$DS_X = \{T \in DS / X \subseteq T\}$ is a transaction set including *X*.

$DS_Y = \{T \in DS / Y \subseteq T\}$ is a transaction set including *Y*.

Assume that *X* is a weighted frequent itemset. This means $WSWsupp(X) \geq \gamma$.

For all subsets $Y \subseteq X, X \neq \emptyset$, we have $\forall T \in DS_X \Rightarrow T \supseteq X \supseteq Y \Rightarrow T \in DS_Y$.

Therefore,

$$DS_X \subseteq DS_Y \Rightarrow |DS_X| \leq |DS_Y|$$

$$\Rightarrow F_{ij}(Y) = \frac{|DS_Y|}{|DS|} \geq \frac{|DS_X|}{|DS|} \geq F_{ij}(X)$$

$$WSWsupp(Y) = \sum_{j=1}^{N} F_{ij}(Y) \times \alpha_{ij}$$
$$\geq WSWsupp(X) = \sum_{j=1}^{N} F_{ij}(X) \times \alpha_{ij}$$

Therefore, it is proved that all subsets *Y* is weighted frequent itemsets.

IV. Process of Mining Weighted Frequent Itemsets over Data Stream Using Tree Data Structure

Process is given as follows:

(i) Building WSWFP-tree (A procedure of building data tree structure);
(ii) Mining WSWFP-tree (A Procedure SWFP-miner);
(iii) Updating WSWFP-tree by updating window procedure (A Procedure Update Window);
(iv) Re-mining WSWFP-tree (Algorithm of mining WSWFP-stream).

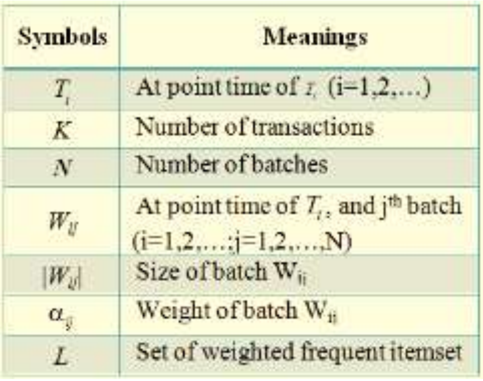
A. *WSWFP-Tree Construction:*

As a FP-tree [8,9], a WSWFP-tree has a structural tree and an item table. However, in order to construct a WSWFP-tree, our proposed algorithm has to access entire database one time only. The item table stores all items in the alphabetical order, their weights, the frequence of each item in each batch, and the pointer points to the first node on the WSWFP-tree that has the same name as the first one . The WSWFP-tree involves a root node called a null node (signs as {}) and a set of precedent trees that are subtrees of root node. The transactions of each batch in database are going to insert into WSWFP-tree by their items in alpabetical order. Except root node, each node on WSWFP-tree stores the name of item that the node represents, the frequence of each node in each batch on the branch from root node and pointers point to parents node, children nodes and the node with the same name on the tree.

When a new node is added into a WSWFP-tree by inserting a transaction from batch *k* of current window including *N* batches, a list of *N* frequent values in *N* batches are filled with the seting of 1 at the position *k* and 0 at other positions. For instance, if the current window has 4 batches and "*b*" is a node appeares at the first time in the tree by inserting a transaction of batch 2, then the structure of node "*b*" is: 0,1,0,0.

Here is some symbols definition using in each procedure and algorithm.

Table 2. Symbols Definition.



Symbols	Meanings
T_i	At point time of T_i (i=1,2,...)
K	Number of transactions
N	Number of batches
W_{ij}	At point time of T_i, and jth batch (i=1,2,...;j=1,2,...,N)
$\|W_{ij}\|$	Size of batch W_{ij}
α_{ij}	Weight of batch W_{ij}
L	Set of weighted frequent itemset

Procedure ConstructWSWFP-tree.
Input: $T_i, K, N, W_{ij} (i = 1,2,...; j = 1,2,..., N)$
Output: WSWFP-tree;
Method

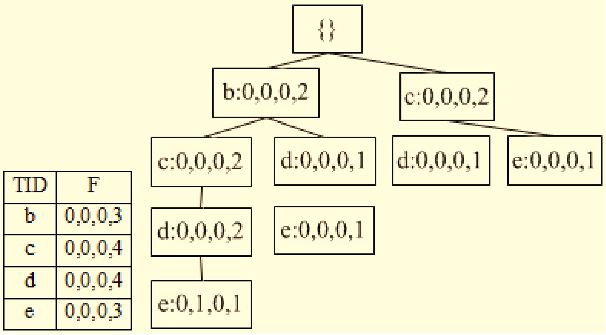
1. **If** (point of *T₁*) **then** create a Root, assign a label of Null ({})
2. **Scan for** (each transaction in current window)
3. **Begin**
4. **With** (each transaction τ in the j^{th} window) **do** revise the performing transaction to insert a node to the tree. The method can be seen as follows:

- The transactions having the same prefix will go in the same way in the prefix in the WSWFP-tree
- Information written in the node will be *NameNode*(w_1,w_2,..,w_N).

If the index in transaction is displayed at the appropriate batch, the w_j=1 otherwise, w_j=0. If the index is appeared many times, the appropriate w_j will be added with 1. Note that, *NameNode* is a name of tree node, and it also is name of the index over stream.

5. Build a index table with information as: TID, index, frequent distribution of index in *N* windows, count of index over stream, pointer pointing to the node that having the same index name in the WSWFP-tree.
6. **End;**
7. **Return** WSWFP-tree

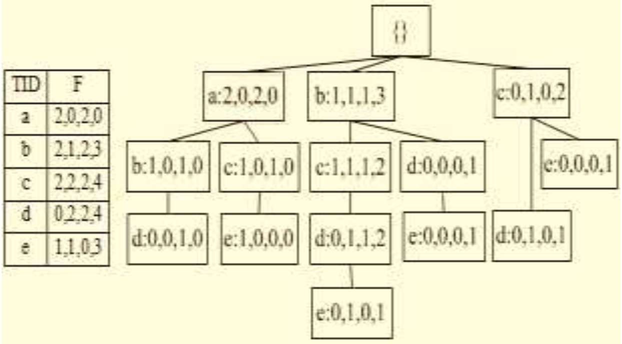
For example, an index table and WSWFP-tree are created after inserting the batch W_{14} (see Fig. 1).



TID	F
b	0,0,0,3
c	0,0,0,4
d	0,0,0,4
e	0,0,0,3

Fig.1. ID Table and WSWFP-tree after Inserting Batch W_{14}.

After inserting 4 batches with 13 transactions in Table 1, the resulted ID table and WSWFP-tree can be seen in Fig. 2.



TID	F
a	2,0,2,0
b	2,1,2,3
c	2,2,2,4
d	0,2,2,4
e	1,1,0,3

Fig.2. ID Table and WSWFP-tree for all Transaction in Table 1.

Obviously, the structure of the nodes in the WSWFP-tree store all required information for mining frequent itemsets in a current window of transaction stream. Moreover, a transaction of each batch is traced and updated easily by tracing and updating WSWFP-tree (delete oldfashion transations, add transactions of a new batch) when switching into mining new window.

B. Mining A WSWFP-Tree

A WSWFP-tree has important properties, which will be used in the proposal algorithm based on FP-growth approach [8, 9], as follow:

Property 1: The high of WSWFP-tree equals to the length of the longest transaction.

Property 2: The sum of frequent values in any node on tree is greater of equal to sum frequent values of any children nodes.

Property 3: The appeared frequency of an item in each batch equals to summary of corresponding frequent values in all nodes have the same name.

Property 4: The distribution of frequency in a batch of a branch on the tree is the distribution of frequency of an antecedent node

By applying the FP–growth [8,9], the proposed algorithm called SWFP-miner for mining frequent itemsets with weight over a data stream can be seen as:

Procedure SWFP-miner.

Input: $T_i, K, N, W_{ij}, \alpha_{ij} (i = 1, 2, ...; j = 1, 2, ..., N), minsupp$
Output: *L*.
Method:

```
1. Calculate the minimal support follows $\gamma$ by (3);
2. From table of items, determine the set $C_1$ is set of
   1-itemsets satisfy $\gamma$.
3. $L = C_1$;
4. For each (for each item $\sigma$ in the items‘ table, in
   reverse alphabetical order) do
5. Begin

   5.1. Create the conditional-tree for corresponding item
        $\sigma$;
   5.2. Construct the candidate itemsets;
   5.3. Eiminate the candidate itemsets have the supports
        do not satisfy $\gamma$;
   5.4. Insert the satisfied itemset $\gamma$ into L;
   5.5. Delete all processed node $\sigma$ on conditional-tree;

6. End;
7. Return L
```

For example: Given the data in Table 1, at the point time T_1 including 13 transactions and 4 batches as W_{11}, W_{12}, W_{13}, W_{14}. The appropriate weights are α_{11} =0.4, α_{12} =0.3, α_{13} =0.2, α_{14} =0.1 and the minimal support *minsupp* is 20%.

Calculate γ by (2), there is:

$$\gamma = minsupp \times \sum_{j=1}^{N} \left| W_{ij} \right| \times \alpha_{ij} = 0.58$$

From the item table, count the supports of 1-itemsets are: $\langle a:1.2, b:1.8, c:2.2, d:1.4, e:1.0 \rangle$.

The 1-itemsets satisfy γ=0.58. Therefore *L=C₁ ={a,b,c,d,e}*.

Construct and extract the condition trees for items in reverse alphabetical order from the bottom of the item table:

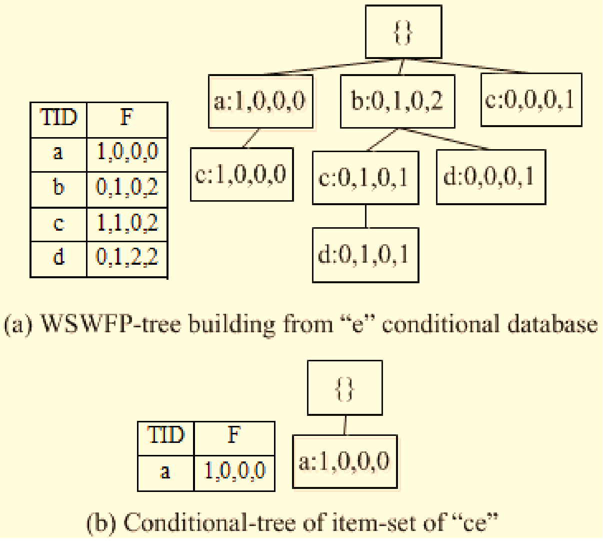
Construct and extract the condition tree of item "e"

Database of condition of the item "*e*" has precedent branch $\{ac:1,0,0,0; bcd:0,1,0,1; bd:0,0,0,1; c:0,0,0,1\}$

From the conditional database, the sub-WSWFP-tree for item "*e*" can be built (see Fig. 3(a)).

From the item table, count the appeared frequencies of two candidates 2-itemsets in the same batch with "*e*" are $\{ae:1,0,0,0; be:0,1,0,2; ce:1,1,0,2, de:0,1,0,2\}$ and the supports with weight of the corresponding itemsets are: $\langle ae:0.4, be:0.5; ce:0.9, de:0.5 \rangle$. There is an itemsets of "*ce*" satisfy γ. Insert the 2- itemsets of "*ce*" into the set *L*, we have: $L = \{a, b, c, d, e, ce\}$.

Continuously extract the 2-itemsets of "*ce*". Extract the conditional-tree of itemsets "*ce*", Fig. 3(b), there are one candidate 3-itemsets $\{ace:1,0,0,0\}$ and their weighted supports $\langle ace:0.4 \rangle$ do not satisfy γ. Continue to extract the conditional-tree of the itemsets of "*ace*" we have a result of null tree. Therefore, $L = \{a, b, c, d, e, ce\}$.



TID	F
a	1,0,0,0
b	0,1,0,2
c	1,1,0,2
d	0,1,2,2

(a) WSWFP-tree building from "e" conditional database

TID	F
a	1,0,0,0

(b) Conditional-tree of item-set of "ce"

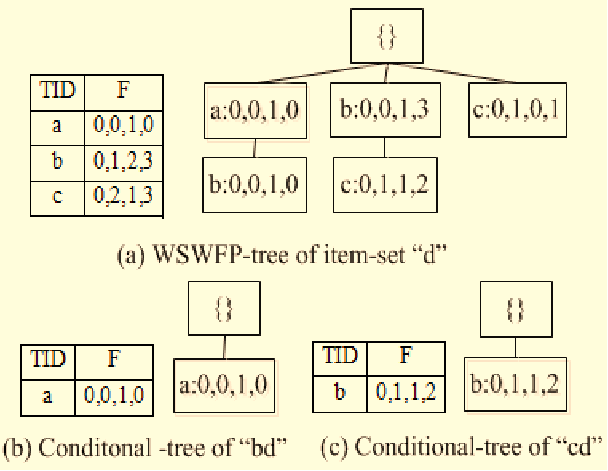
Fig.3. The WSWFP- tree of item "e" and the Conditional-Tree of Itemsets "ce".

Construct and extract the condition tree of item "d"

The database of condition of the item "*d*" has some precedent branches as

$$\{ab:0,0,1,0; bc:0,1,1,2; b:0,0,0,1; c:0,1,0,1\}$$

The database of condition construct WSWFP-tree of the item "*d*" depicted in Fig. 4(a).



TID	F
a	0,0,1,0
b	0,1,2,3
c	0,2,1,3

(a) WSWFP-tree of item-set "d"

TID	F
a	0,0,1,0

TID	F
b	0,1,1,2

(b) Conditonal -tree of "bd" (c) Conditional-tree of "cd"

Fig.4. The WSWFP-tree of item "d" and Conditional-Tree of two Itemsets "bd" and "cd".

From the item table with the frequency of item "*d*", there are candidates 2-itemsets of "*bd*" and "*cd*" satisfy γ, according to the windows of $\{ad:0,0,1,0; bd:0,1,2,3; cd:0,2,1,3\}$ and the appropriate support with weights as $\langle ad:0.2, bd:1.0; cd:1.1\rangle$. Therefore, we have $L=\{a,b,c,d,e,ce,bd,cd\}$.

Mining conditional-tree of "*bd*" (Fig. 4(b)) we have a candidate 3-itemsets of $\{abd:0,0,1,0\}$, and the support with weight of $\langle abd:0.2\rangle$ do not satisfy γ.

Mining conditional-tree of "*cd*" (Fig. 4(c)) we have a candidate 3-itemsets of $\{bcd:0,1,1,2\}$;and the support with weight of $\langle bcd:0.7\rangle$ satisfy γ. Inserting "*bcd*" into L, we have $L=\{a,b,c,d,e,ce,bd,cd,bcd\}$.

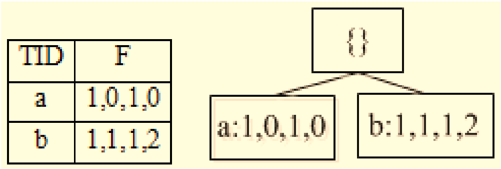
Continue to mine the conditional-tree of "*bcd*" we have a null tree.

Therefore, $L=\{a,b,c,d,e,ce,bd,cd,bcd\}$.

Construct and extract the condition tree of item "c"

The database of condition of the item "*c*" has precedent branches $\{a:1,0,1,0; b:1,1,1,\}$.

The database of condition construct WSWFP-tree of the item "*c*" depicted in Fig. 5.



TID	F
a	1,0,1,0
b	1,1,1,2

Fig.5. WSWFP-tree Building from Database of "c".

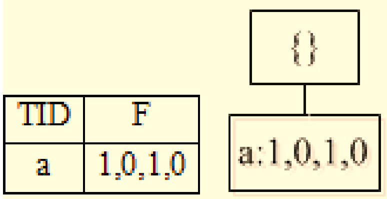
Do the same method as constructing and extract of other items above we have:

$$L=\{a,b,c,d,e,ce,bd,cd,bcd,ac;bc\}.$$

Construct and extract the condition tree of item "b"

The database of condition of the item "*b*" has one precedent branch $\{a:1,0,1,0\}$.

The database of condition construct WSWFP-tree of the item "*b*" depicted in Fig. 6.



TID	F
a	1,0,1,0

Fig.6. The WSWFP-tree of item "b".

Do the same method as constructing and extract of other items above we have:

$$L=\{a,b,c,d,e,ce,bd,cd,bcd,ac;bc;ab\}.$$

Construct and extract the condition tree of item "a"

The result is a null tree.

At the time point T_1, the set of weighted frequent itemsets and their supports are

$$L=\begin{Bmatrix} a:1.2, b:1.8, c:2.2, d:1.4, e:1.0, ce:0.9, bd:1.0, \\ cd:1.1, bcd:0.7, ac:0.6; bc:1.1; ab:0.6 \end{Bmatrix}$$

C. Update WSWFP-Tree by the Use of Update-Window Procedure

To delete the information from oldfashtion batches on a tree, the following tasks have to be done as:

- In the list of frequent values of each node on tree, replace the value at first position by 0, and the value at position of *j* (1<*j*≤N) is replace by the value of position of *j*-1.
- Prune any node which has its all frequent values as 0.

After deleting the oldfashion batches, a new batch is inserted in to the tree as normal (call Procedure *ConstructWSWFP-tree).*

For example: Fig. 7 represents a WSWFP-tree at the point time T_1 after deleting the batch W_{14}.

Asume that at the point time T_2, the window of data stream can be seen in Table 3.

After deleting the old fashion batches, the transactions of new batch are inserted into tree and a new update tree at the point time T_2 can be seen in Fig. 8.

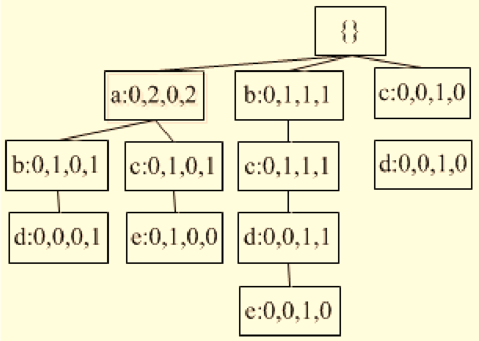
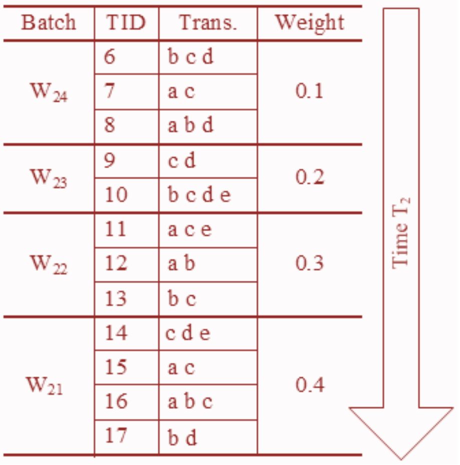


Fig.7. WSWFP-tree after Deleting Batch W14 with Database in Table 1.

Table 3. An Example of Data Stream at Time-Point $T_{2.}$



Batch	TID	Trans.	Weight
W_{24}	6	b c d	0.1
	7	a c	
	8	a b d	
W_{23}	9	c d	0.2
	10	b c d e	
W_{22}	11	a c e	0.3
	12	a b	
	13	b c	
W_{21}	14	c d e	0.4
	15	a c	
	16	a b c	
	17	b d	

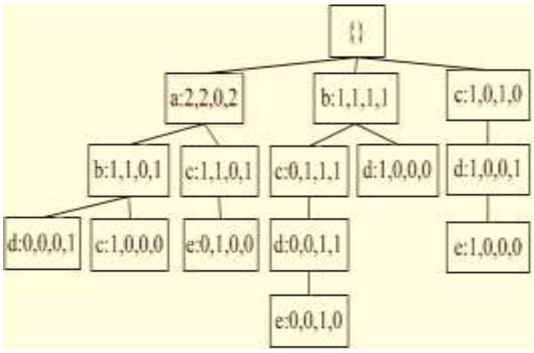


Fig.8. WSWFP-tree after Updating Batch of W21 at Time-Point T2.

Procedure UpdateWindow().

```
1.  For each (route in WSWFP-tree) do
2.      Begin
3.  With (current node) then change Its values as:
        $NameNode(w_1, w_2, \ldots, w_N)$ is replaced by
        $NameNode(0, w_1, \ldots, w_{N-1})$;
4.  If (for all $w_j = 0$) then delete the nodes in the route;
5.     End;
6.  Scan for (all transactions in the new batch ($W_{i1}$))
7.  Begin
8.  Call ConstructWSWFP-tree; // Only perform over transactions in the newest batch;
9.  End;
10. Return WSWFP-tree
```

D. *Mining Frequent Data-Set with Weights over Data Stream by Using Tree*

The algorithm WSWFP-stream is built basing on mining many time the data stream in WSWFP-tree (delete the old fashion window and update the new one in the 3rd step).

Algorithm WSWFP-stream.
Input: $T_i, K, N, W_{ij}, \alpha_{ij} (i = 1, 2, ...; j = 1, 2, ..., N), minsupp;$
Output: *L*;
Method:

```
1.  If (point of $T_1$) then
2.  Begin
3.  Call ConstructWSWFP-tree;  // Building a tree with all transactions over data stream
4.  Call SWFP-miner; // Mining WSWFP-tree for the first time
5.  End
6.  Else
7.  Begin
8.      cont=”Y”;
9.  While (upper(cont)=”Y”) do    // Process will be repeated with variable of cont
10. Begin
11. Call ReleaseWindowOld; //Delete the oldfashion windows
12. Call UpdateWindow;
13. Call SWFP-miner;     // Re-mining WSWFP-tree (after updating)
14. Accept “Do you want to mine data stream (Y/N)?” to cont;
15. If (upper(cont)≠”Y”) then exit;
16. End;
17. End;
18. Return L
19. End.
```

V. Experiment

In order to evaluate the performance of WSWFP-stream and compare to the Tsai's two algorithms (WSW and WSW-Imp), we have run several tests on PC Pentium dual core 2.13 GHz CPU with 1GB memory of the use operating system Window 7. The program is coded by Microsoft Visual C++ version 6.0, and method to generate transactions has been done the same as one used in Apriori [2]. The given parameters for the experiment as: the number of items N = 1000, and the maximum number of frequent itemsets |L| = 2000.

To simulate a data stream as a weighted sliding window model, each transaction is processed continuously. We also assume that the number of transactions in each batch is the same. In our experiment, the number of transactions in each batch is 4, and the weights of each batch W_{i1}, W_{i2}, W_{i3} and W_{i4} are 0.4, 0.3, 0.2 and 0.1, while the weight w_{i1} is closest window of present time point *i*.

Fig. 9 shows the processing times of three algorithms WSWFP-stream, WSW and WSW-Imp with different minimal supports. In this test, the total number of transactions is 100K and the number of each batch is 10K. Because the number of batches in each window is 4 then the number of time points is 7. The processing time is summary of times spent for all time points.

In overall, it shows that the WSWFP-stream running time is better in alternative minimal supports. For example, WSWFP-stream running time is taken in about 27 seconds whereas WSW and WSW-Imp running time are over 32 seconds.

The figure shows that as the minimal support reduces the competitive advantage of WSW-Imp increases.

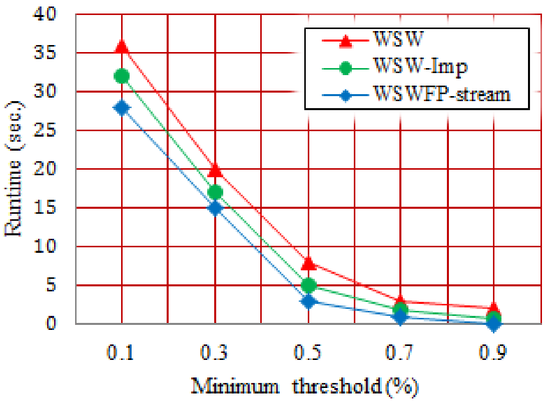


Fig.9. The Processing Times of Three Algorithms Vs the Minimal Support.

Fig. 10 depicts the processing times of three algorithms WSWFP-stream, WSW and WSW-Imp with different number of transactions in a sliding window at minimal support value 0.1. The processing times of three algorithms are increase as the number of transactions increase. The WSWFP-stream running time is nearly the same as two others at the small number of transaction. However, its running is quite different to the others when the number of transactions are increased. For example, with 250K of transaction the WSWFP-stream running time was taken in 90 seconds whereas the others running times were more than 100 seconds. In overall, the performance of WSWFP-stream is out standing from the two algorithms, WSW and WSW-Imp, in all situations.

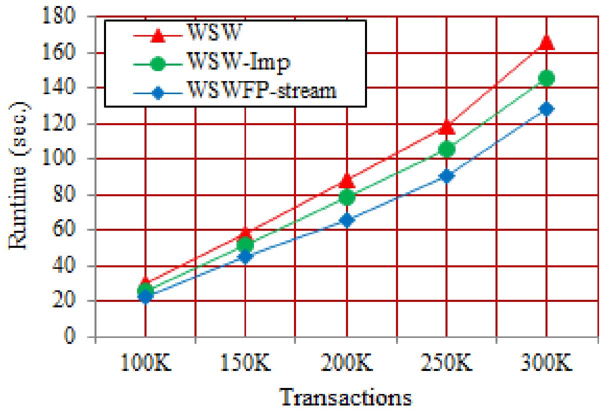


Fig.10. The Processing Time Vs the Number of Transactions in a Sliding Window.

To compare the results of three algorithms, we have been experimented them to alternative size of batch.

The Fig. 11 shows the effect of size of batch, from 1K to 20K of transactions, at the minimal support 0.5. We can see that as the size of a batch increases the different between the processing times of three algorithms WSWFP-stream, WSW and WSW-Imp decreases.

For example, for 1K size of batch, the WSWFP-stream running time is taking in advance but its time is going to the same as WSW and WSW-Imp running times at the large size (over 20K).

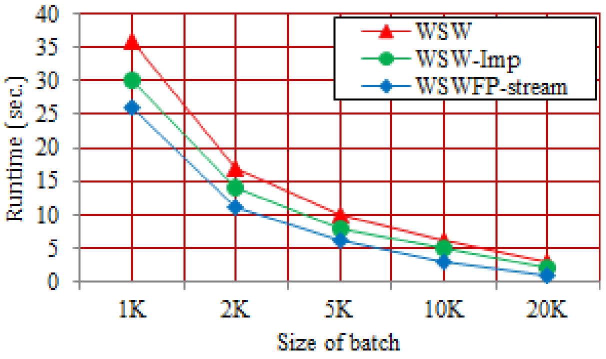


Fig.11. The Processing Time Vs The Size of Batch.

When the size of batch is small the number of frequent itemsets is small also that cause the probability of infrequent itemsets in a batch is higher. In general, the performance of WSWFP-stream algorithm is better than the performances of two algorithms WSW and WSW-Imp.

VI. Conclusion

The mining frequent itemsets has an important role in data mining to mine the data n the real world. In this paper, basing on the weighted sliding window model derived from Tsai P. S. M. [20], we have proposed a new algorithm so-called WSWFP-stream for mining weighted frequent itemsets over a data stream. This algorithm is based on the FP-growth's approach [8,9]. It means that the algorithm's process only needs to access the database one time, stores all information in a tree, which has an easily updating structure and does not generate large number of candidates during the extraction. Therefore, the proposed algorithm WSWFP-stream is theoretically more efficient than Tsai's algorithms (WSW and WSW-Imp) in processing time and memory aspects. The experiment also proved this point of view.

References

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

[2] Agrawal R., Srikant, R., Fast Algorithms for Mining Association Rules. In: *20th Int. 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. Jour. for Technology Research Engineering*, Vol. 1, Issue 9, pp. 935-940, (2014).

[4] Chang J.H., Lee W.S.: estWin, Online data stream mining of recent frequent itemsets by sliding window method. *Journal of Information Sciences*, Vol. 3, No. 2, pp. 76-90, (2005).

[5] Chi Y., Wang H., Yu P. S., Muntz R. R., Catch the moment: Maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems*, Vol. 10, No. 3, pp. 265–294, (2006).

[6] Fan W., Huang Y., Wang H., Yu, P. S. Active mining of data streams. In: *Proceedings of the Fourth SIAM Int. Conf. on Data Mining*, pp. 457-461, (2004).

[7] 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).

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

[9] 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).

[10] Jothimani K., Thanamani A. S., An overview of mining frequent itemsets over data streams using sliding window model, *Int. Jour. Of Emerging Trend & Technology in Computer Science (IJETTCS)*, Vol. 1, Issue 1, pp. 86-89, (2012).

[11] Keming T., Caiyan D., Ling C., A novel strategy for mining frequent closed itemsets in data streams, *Journal of Computer*, Vol. 7, No. 7, pp. 1564-1573, (2012).

[12] Kuen-Fang J., Chao-Wei L., A sliding-window based adaptive approximating method to discover recent

frequent itemsets from data streams, *Proc. of the Int. Multiconference of Engineering and Computer Scientists (IMECS 2010)*, Vol. I, March 17-19, Hong Kong, (2010).

[13] Li Su, Hong-yan Liu, A new classfication algorithm for data stream, *Int. Jour. Modern Education and Computer Science,* Vol. 3, No. 4, pp, 32-39, (2011).

[14] Lin C.H., Chiu D.Y., Wu Y.H., Chen A.L.P., Mining frequent itemsets from data stream with a time-sensitive sliding window. In: *5th SIAM Int. Conf. on Data Mining*, pp. 68-79, (2005).

[15] Mahmood D., Mohammad H. S., An efficient algorithm for mining frequent itemsets within large windows over data streams, *Int. Jour. of Data Engineering*, Vol. 2, Issue 3, pp. 119-125, (2011).

[16] Mahmood D., Mohammad H. S., Mehran T., An efficient sliding window based algorithm for adaptive frequent itemset mining over data streams, *Journal of Information Science and Engineering 29*, pp. 1001-1020, (2013).

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

[18] Reshma Yusuf B., Chenna Reddy B., Mining data stream using option trees, *Int. Jour. Network and Information Security*, Vol. 4, No. 8, pp. 49-54, (2012)

[19] Shaik H., Murthy J. V. R., Anuradha Y., Chandra M., Mining frequent patterns from data streams using dynamic DP-tree ¸*Int. Jour. of Computer Applications*, Vol. 52, No. 19, pp. 23-27, (2012).

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

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

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

[23] 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).

[24] Yong C., Rong F. B., Chuan X., A new approach for maximal frequent sequential patterns mining over data streams, *Int. Jour. of Digital Content Technology and its Applications*, Vol. 5, No. 6, pp. 104-112, (2011)

[25] 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.

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, information systems especially to management information systems. Many her publications also are concentrated to these areas.

Giap N. Cu was born in 1984 in Phutho province, Vietnam. He received BSc. degree of Information Technology at Hanoi University of Technology in 2007. In 2012, he received MSc. degree of Computer Science in Vrije Universiteit Brussels. Now, He is a lecturer in Faculty of Economic Information System in Vietnam Commercial University. His research interests include: Parallel & genetic Algorithm; Neutral network; Expert & Prediction system.

How to cite this paper: 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", International Journal of Intelligent Systems and Applications (IJISA), vol.7, no.12, pp.23-31, 2015. DOI: 10.5815/ijisa.2015.12.02