I.J. Intelligent Systems and Applications, **2015, 11, 41-48**
Published Online October 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2015.11.06

An Efficient Algorithm for Mining Weighted Frequent Itemsets Using Adaptive Weights

Hung Long Nguyen
Faculty of Economic Information System, Vietnam Commercial University (VCU)
E-mail: ntthlong@gmail.com

Abstract—Weighted frequent itemset mining is more practical than traditional frequent itemset mining, because it can consider different semantic significance (weight) of items. Many models and algorithms for mining weighted frequent itemsets have been proposed. These models assume that each item has a fixed weight. But in real world scenarios, the weight (price or significance) of the items may vary with time. Therefore, reflecting these changes in item weight is necessary in several mining applications, such as retail market data analysis and web click stream analysis. Recently, Chowdhury F. A. et al. have introduced a novel concept of adaptive weight for each item and propose an algorithm AWFPM (Adaptive Weighted Frequent Pattern Mining). AWFPM can handle the situation where the weight (price or significance) of an item may vary with time. In this paper, we present an improved algorithm named AWFIMiner. Experimental computations show that our AWFIMiner is more efficient and scalable for mining weighted frequent itemsets using adaptive weights. Moreover, because it only requires one single database scan, the AWFIMiner is applicable for mining these itemsets on data streams.

Index Terms—Data mining, Knowledge discovery, Weighted frequent itemset mining, Adaptive weight, Pattern growth techninque.

I. Introduction

Data mining discovers hidden and potentially useful information from databases. Frequent itemset (or frequent pattern) mining is an important technique of data mining. Finding frequent itemsets is a powerfull tool in mining association rules, closed itemsets, functional dependencies, … [1,4,5,6,7].

In recent years, weighted frequent itemset mining has been studied by many people [2,3,9-15]. Weighted frequent itemset mining is different from traditional frequent itemset mining in which we are not only interested in the number of times items appearing in the database but also interested in the degree of different significance (weight) of items. In many pratical applications, items in a transaction can have different degree of importance. For example, in retail market analysis, even though expensive products do not appear in a large number of transactions, they contribute a larger portion of overall revenue. Therefore, weighted frequent itemset mining plays more pratical role in the real world scenarios than traditional frequent itemset mining [1, 5,6,7].

Even though, weighted frequent itemset mining consider different weights of each item during the mining process, it is not enough to reflect the real world enviroment where the weight of an item can vary with time. In our real world scenarios, the significance (weight) of an item might be widely affected by many factors. Customer's buying behaviors (or interests) are changing with time, so they affect the significances (weights) of products in retail markets. The weights of seasonal products may also vary when the season changes from summer to winter or winter to summer. Web click stream analysis can be another example of this matter. The significance of each website may change with time depending on the popularity, political issues, public events and so on.

Recently, in [3], Chowdhury F. A. et al**.** have introduced a new approach for weighted frequent itemset mining with an assumption that weights of items can vary with time and proposed the algorithm AWFPM (Adaptive Weighted Frequent Pattern Mining).

In this paper, we present an improved algorithm named AWFIMiner. Experimental computations show that our AWFIMiner is more efficient and scalable than AWFPM algorithm.

The remainder of this paper is organized as follows. In Section II, we describe some recent related works. In Section III, we state the problem of adaptive weighted frequent pattern mining and explain our proposed algorithm AWFIMiner. In Section IV, our experimental results are presented and analyzed. Finally, in Section V, conclusions are drawn.

II. Related Works

Weighted frequent itemset mining was first proposed and studied by Cai C. H. et al**.** in [2]. In this work, the authors have introduced a concept of weighted support and proposed the MINWAL algorithm. A weighted support of an itemset is defined as the resultant value of multiplying its support with the average weight of the member items. The main challenging problem of weighted frequent itemset mining relates to the downward closure property (also known as Apriori property). This property tells that if an itemset is infrequent then all of its supersets must be infrequent. This property is

broken if different weights are applied to the items. To maintain the downward closure property, MINWAL defined an upper bound, called k-support. Support of itemsets generated in level k must be greater or equal to the k-support bound. MINWAL is based on the Apriori algorithm in traditonal frequent itemset mining, but most of the candidates are infrequent. Moreover, according to [13, 14], it takes too long to use the k-support bound for satisfying the downward closure property. After MINWAL, some other models and algorithms have been proposed. Most of them are based on the Apriori [1] algorithm.

In [10], Tao F. has proposed the WARM algorithm. The problem of the breaking the downward closure property is solved by using a weighted support and developing a weighted downward closure property. However, the meaning of weighted support is different from that defined in MINWAL. Weighted support of itemset "ab" in WARM is the fraction of the weight of transactions containing both "a" and "b" to the weight of all transactions in the database. WARM is also algorithm based on the Apriori.

In [11], Wang W. et al. proposed an algorithm named WAR for mining weighted association rules. For mining these rules, WAR first generated frequent itemsets without considering weights and then does post-processing during the rule gemeration step. Thus WAR algorithm dose not concerned with mining weighted frequent itemsets and is post-processing approach. Moreover, WAR is also based on the Apriori algorithm.

The algorithms which are developed based on the Apriori algorithm use candidate generation-and-test paradigm. Obviously, these algorithms require multiple database scans and result in poor mining performance. The first FP-tree based weighted frequent itemset mining algorithm is WFIM which has been proposed by Yun U. and Leggett J. J. [12]. WFIM uses two database scans over a static database. It has used a minimum weight and a weight range. Items are given fixed weights randomly from the weight range. It has arranged the FP-tree in weight ascending order.

In [13], Yun U. presented a WIP (Weighted Interesting Pattern mining with a strong weight and/or support affinity) algorithm that integrates the strengths of the previous techniques and generates weighted interesting patterns according to user feedback. In WIP, a new measure, weight confidence, is defined to generate weighted hyperclique patterns with similar levels of weights. A weight range is used to decide weight boundaries and the h-confidence measure serves to identify strong support affinity patterns. WIP not only gives a balance between the two measures of weight and support, but also considers weight affinity and/or support affinity between items within patterns so more valuable patterns can be generated.

In [14], Yun U. re-examined two basic but interesting constraints, a weight constraint and a length decreasing support constraint and propose WLPMiner (weighted frequent pattern mining with length decreasing constraints). WLPMiner integrates these two measures to generate fewer and more meaningful patterns. For pruning techniques, the author has used the notion of WSVE (Weighted Smallest Valid Extension) to apply to both the length decreasing support constraints and weight constraints, and a weight range as a supplement to maintain the downward closure property. The key insights achieved in this approach are the high performance of the WSVE property and the use of a weight range in the weight constraint. It is shown that combining a weight constraint with a length decreasing support constraint improves performance in terms of the number of patterns and runtime. WLPMiner is also an algorithm using FP-tree structure.

In [15], Zhang S. et al. proposed a new strategy, called Weight, for maintaining the association rules in incremental databases by using the weighting technique to high-light new data. Any recently added transactions are assigned higher weights. Moreover, all transactions in a database are given the same weight. They did not use different weights for individual items or transactions. Their algorithm is based on the level-wise candidate generation-and-test methodology of the Apriori algorithm. Therefore, for a particular dataset, they gene-rate a large number of candidates and need to perform several database scans to get the final result.

Recently, stream data mining has become an important research area in computer science [8,9]. In [9], Pauray S. M. Tsai proposed a new procedure for stream data mining which is called Weighted sliding window model. This model allows users to fix the number of mining windows and their sizes. However, like above models, all items in the windows are assigned the same weight.

The weight varying with time problem has just considered recently in [3] by Chowdhury F. A. et al. In this work, the authors introduced a novel concept of adaptive weight for each item and proposed an algorithm AWFPM (Adaptive Weighted Frequent Pattern Mining). AWFPM can handle the situation where the weight of an item may be changed in any batch of transactions in the database. An pattern is called adaptive weighted frequent pattern if the adaptive weighted support of the pattern is greater or equal to the minimum threshold. AWFPM exploits a pattern growth mining technique to avoid the level-wise candidate generation-and-test problem. To maintain the downward closure property, AWFPM uses the global maximum weight and the local maximum weight. The global maximum weight is maximum weight of all the items in the global database and the local maximum weight is the the highest weight of all the items in a conditional database.

In this paper, we reconsider the model for mining weighted frequent itemsets using adaptive weights proposed by Chowdhury F. A. et al. in [3]. Our goal is to give an improved algorithm that can be more efficient in both memory space and runtime.

III. Adaptive Weighted Frequent Itemsets and AWFIMiner Algorithm

A. Preliminaries

Given a transaction database DT, let I be the set of all the items (attributes) in DT. Each transaction T in DT is a subset of I, which has a transaction identifier $\langle \text{TID} \rangle$. A subset of I which consists of k distinct items, is called k-itemsets or itemset of length k.

Assume that DT is divided into k batches; each item in each batch is assigned a distinct weight, which is a nonnegative real number.

Definition 1. [3] Adaptive weighted support of an itemset X, denoted as AWsupp(X), is defined by:

$$\text{AWsupport}(X) = \sum_{j=1}^{K} W(X, j) \times F(X, j) \qquad (1)$$

where W(X, j) is the weight of X in the j^{th} batch which is calculated by the average weight of the items in the batch belonging to X, F(X, j) is the support (or frequency) of X in the j^{th} batch.

Definition 2. [3] An itemset X is called adaptive weighted frequent itemset if the adaptive weighted support of X is greater or equal to the minimum threshold AWminsupp, that is:

$$\text{AWsupp}(X) \geq \text{AWminsupp} \qquad (2)$$

Example: Given the database shown in Table 1 consisting of 3 batches.

Table 1. An example of adaptive weighted database.

Batch	TID	Trans.	Weight				
1	T1	b c e	a	b	c	d	e
	T2	a b e	0.3	0.7	0.5	0.2	0.5
	T3	b c					
2	T4	a c	a	b	c	d	e
	T5	c d e	0.6	0.2	0.5	0.3	0.5
	T6	c e					
3	T7	a b e	a	b	c	d	e
	T8	c e	0.6	0.4	0.7	0.4	0.6
	T9	a c d					
	T10	c d e					

The adaptive weight support of the itemset "be" is

$$\text{AWsupp(be)} = \frac{0.7+0.5}{2} \times 2 + \frac{0.2+0.5}{2} \times 0 + \frac{0.4+0.6}{2} \times 1 = 1.7$$

If the minimum threshold is 1.4 then "be" is an adaptive weighted frequent itemset.

Given a transaction database DT, the weights of the items, our task is to find all adaptive weighted frequent itemsets in DT.

An adaptive weighted frequent itemset defined as above does not satisfy the downward closure property. For example, consider the database shown in Table 1. Using (1) we have

$$\text{AWsupport(d)} = 0.2 \times 0 + 0.3 \times 1 + 0.4 \times 2 = 1.1$$

$$\text{AWsupp(cd)} = \frac{0.5+0.2}{2} \times 0 + \frac{0.5+0.3}{2} \times 1 + \frac{0.7+0.4}{2} \times 2 = 1.5$$

If the minimum threshold is 1.4 then "cd" is an adaptive weighted frequent itemset but "d" is not.

In order to have the downward closure property, we introduce a notion of maximum adaptive weighted frequent itemset, defined as follows.

Definition 3. Given a transaction database DT consisting of K batches and an itemset X. Let MAXW(j) be the highest weight value of the items in the j^{th} batch, $j = 1, \ldots, K$. Then the measure

$$\text{MAXAWsupp(X)} = \sum_{j=1}^{K} \text{MAXW}(j) \times F(X, j) \qquad (3)$$

is called the maximum adaptive weighted support of X in DT.

Example: Consider the database presented Table 1, we have K = 3.

$$\text{MAXW}(1) = 0.7,\ \text{MAXW}(2) = 0.6,\ \text{MAXW}(3) = 0.7$$

the occurrence frequent of "be" in the first, second and third batch are 2, 0 and 1 respectively. Thus

$$\text{MAXAWsupp(be)} = 0.7 \times 2 + 0.6 \times 0 + 0.7 \times 1 = 2.1$$

Definition 4. Given a transaction database DT consisting of K batches and an itemset X. For a given threshold AWminsupp, X is called a maximum adaptive weighted frequent itemset if

$$\text{MAXAWsupp}(X) \geq \text{AWminsupp}$$

Proposition 1. Maximum adaptive weighted frequent itemset has the downward closure property, that is if X is a maximum adaptive weighted frequent itemset then all of its subsets are also maximum adaptive weighted frequent itemsets.

Proof. For every $Y \subseteq X$, we have $F(Y, j) \geq F(X, j)$, j=1,…,K. It follows that

$$\sum_{j=1}^{K} MAXW(j) \times F(Y, j) \geq \sum_{j=1}^{K} MAXW(j) \times F(X, j)$$

or

$$MAXAWsupp(Y) \geq MAXAWsupp(X)$$

Therefore, if

$$MAXAWsupp(X) \geq AWminsupp$$

then we also have

$$MAXAWsupp(Y) \geq AWminsupp \blacksquare$$

Proposition 2. Given a transaction database DT and an itemset X. If X is an adaptive weighted frequent itemset then X is also a maximum adaptive weighted frequent itemset.

Proof. For every itemset X, we always have

$$MAXW(j) \geq W(X, j), \forall\ j = 1, ..., K$$

Therefore, if

$$\sum_{j=1}^{K} W(X, j) \times F(X, j) \geq AWminsupp$$

then we also have

$$\sum_{j=1}^{K} MAXW(j) \times F(X, j) \geq AWminsupp \blacksquare$$

Proposition 1 and Proposition 2 show that maximum adaptive weighted frequent itemsets have the downward closure property and they are candidates for adaptive weighted frequent itemsets. Thus, in order to mine adaptive weighted frequent itemsets, we propose an algorithm AWFIMiner consisting of two steps, as following:

1) Find all maximum adaptive weighted frequent itemsets.
2) From the set of all maximum adaptive weighted frequent itemsets, by applying (1), determine the the set of all adaptive weighted frequent itemsets.

To find the maximum adaptive weighted frequent itemsets efficiently, we apply pattern growth techninque as in the FP-growth algorithm for mining traditional frequent itemsets [5,7]. First we construct a tree structure, called AWFI-tree for the database and then we mine conditional trees to find maximum adaptive weighted frequent itemsets.

B. Construction of AWFI-tree

Like FP-tree [5], the structure of AWFI-tree is combined of two parts: AWFI-tree and a header table. AWFI-tree consists of one root node referred to as "null" (signs as {}), a set of item-prefix sub-trees as children of the root. Header table is maintained to keep items in lexico-graphical order and the related information of items. In each entry of a header table, the first value is the item- identifier, after that the weight and frequency information of an item in batch by batch fashion, and a pointer pointing to the first node in the AWFI-tree carrying the item. Transactions in each batch of the database are inserted one by one into the tree in lexicographical order of items. Except the root, each node of the AWFI-tree contains item-identifier and its frequency information. To facilitate the tree traversals adjacent links are also maintained in AWFI-tree like FP-tree.

To avoid including a list of batch-based frequency information at every node in AWFI-tree (as AWFP-tree does), we introduce the concept of tail-node which can be defined as follows.

Definition 5. Let $T = \{i_1, i_2, ..., i_k\}$ be a transaction in a database with items sorted according to lexicographic order. If T is inserted into a SWFP-tree in this order, then the node of the tree that represents item i_k is defined as a tail-node for T. For example, if the lexicographically sorted transaction {a,c,d} is inserted into an AWFI-tree, then the node that represents item “d” is a tail-node in the tree for this transaction.

Hence, two types of nodes can be maintained in a AWFI-tree: tail-nodes and ordinary nodes. The latter are the types of nodes that are used in the FP-tree.

Remark: Every leaf node of the AWFI-tree must be tail-node.

By using the notion of tail-node and ordinary node, we organize nodes in the AWFI-tree as follows: At each ordinary node, we only keep the total frequency of the node in the path from the root, but at each tail-node, we maintain a list of batch-by-batch frequency information. Whenever a new tail-node is created in the tree by inserting a transaction from the j^{th} batch of the database consisting of K batches, a list consisting of K frequency values in K batches will be created with value 1 at the j^{th} position, value 0 at all remaining positions. For example, if a tail-node "b" first appears in the second batch of the database consisting of 3 batches then the structure of the node will be b:0,1,0.

Obviously, with the above organization of nodes, we can save memory space and still maintain all information needed for mining database.

Fig. 1 represents the AWFI-tree constructed for the database shown in Table 1 (the traversals adjacent links are not shown in the figure for simplicity). In this tree, there are 7 tail-nodes and 5 ordinary nodes. As batch-by-batch frequency information is kept separately in each tail-node we can easily discover that which transactions have been occurred in which batch. For example, we can easily detect that the the transaction {a,b,e} appears once

in the first batch and once in the third batch, the transaction {c,d,e} appears once in the second batch and once in the third batch, the occurence frequencies of the items in the database are a:4, b:4, c:8, d:3 and e:7 respectively.

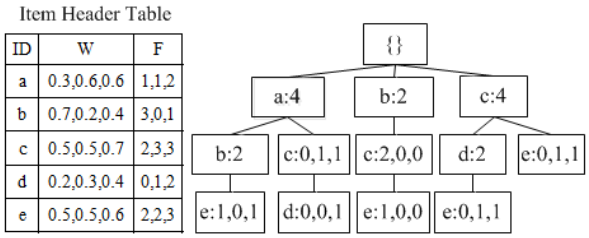


Fig.1. The AWFI-tree after inserting 3 batches of the database of Table 1.

We can consider the AWFI-tree constructed above as an extended FP-tree. It inherits the compress advantage of FP-tree. Moreover, it has its own properties to feed the need of mining adaptive weighted frequent itemsets.

Property 1. The AWFI-tree can be constructed in a single-pass of database.

Property 2. The total count of frequency values of any node in the AWFI-tree is greater than or equal to the sum of total counts of frequency values of its children.

Property 3. The frequency value in each batch of an ordinary node in the AWFI-tree equal to the sum of corresponding frequency values of the tail nodes which are its children.

For example, in the AWFI-tree shown in Table 1, the ordinary node "b" in the first branch on the left has one child tail-node e: 1,0,1, so the frequency distribution of node "b" in three batches is the same as the frequency distribution of node "e", that is, b:1,0,1. The ordinary node "a" in the first branch on the left has two child tail-nodes e:1,0,1 and c:0,1,1, so the frequency distribution of "a" in three batches is a:1+0,0+1,1+1, that is a:1,1,2.

Property 4. The frequency value in each batch of an item in the database equal the sum of corresponding frequency values of the nodes carring the same name.

For example, in the AWFI-tree of Table 1, the node "b" appears once in the first branch with the frequency distribution 1,0,1 and once in the third branch with the frequency distributions 2,0,0, so the frequency distribution of "b" in batches is b: 3,0,1.

Property 5. The frequency distribution in batches of a path in the AWFI-tree is the frequency distribution of the suffix node.

For example, in the AWFI-tree presented in Table 1, the path $\langle a:4; c:0,1,1\rangle$ has the suffix node "c", so its frequency distribution is 0,1,1 which is the frequency distribution of c.

C. AWFIMiner Algorithm

By using the pattern growth mining approach, the AWFIMiner algorithm mines adaptive weighted frequent itemsets from the AWFI-tree as follows.

Algorithm: AWFIMiner()

Input: A transaction database contains K batches, the weights of the items in each batch, the minimum support count threshold AWminsupp;

Output: The set of all adaptive weighted frequent itemsets;

1. Create AWFI-tree;
2. $L = \varnothing$;
3. Identify the set C_1 of candidate 1-itemsets, that are itemsets whose MAXAWsupp calculated by the formula (3) is no less than AWminsupp;
4. $L = L \cup C_1$;
5. Prune the AWFI-tree: erase every node in the AWFI-tree that does not represent for the candidate items in C_1, (these nodes can not associate with other nodes to generate maximum adaptive weighted frequent item-sets (according to Proposition 1));
6. For each item in the header table, in bottom-up order:

6.1. Construct its conditional tree;

6.2. Prune the conditional tree: erase every node which does not represent for the candidate items;

6.3. Mine pruned conditional tree to get maximum adaptive weighted frequent itemsets by pattern growth. Add the maximum adaptive weighted frequent itemsets into L;

6.4. From L, determine the adaptive weighted frequent itemsets, that are itemsets having AWsupp value no less than AWminsupp (calculated by (1));

Example: Consider the database shown in Table 1. We find the set of all adaptive weighted frequent itemsets by using the AWFIMiner algorithm with the threshold AWminsupp=1.4 as follows.

1. Construct AWFI-tree, we get the tree shown in Fig. 1.
2. From the header table, we have

$$MAXW(1) = 0.7,\ MAXW(2) = 0.6,\ MAXW(3) = 0.7$$

$$MAXAWsupp(a) = 0.7\times1 + 0.6\times1 + 0.7\times2 = 2.7$$

$$MAXAWsupp(b) = 2.8,\ MAXAWsupp(c) = 5.3$$

$$MAXAWsupp(d) = 2.0,\ MAXAWsupp(e) = 4.7$$

We observe that all the items in the original database have the values MAXAWsupp no less than AWminsupp =1.4, so they are not pruned in the AWFI-tree and they are single candidates. We have, $L = \{a, b, c, d, e\}$.

3. Construct and mine conditional trees of items in bottom-up order in the header table:

3.1. Construction and mining the conditional tree of "e". The conditional database of "e" contains prefix paths

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

From this conditional database we have the AWFI-tree presented in Fig. 2(a).

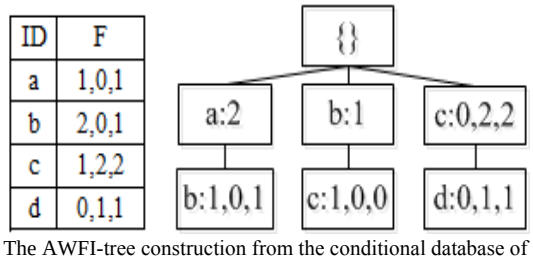
Because the conditional database of "e" contains all the items of the original database, $MAXW(1) = 0.7, MAXW(2) = 0.6, MAXW(3) = 0.7$ From the header table, the frequency values of items co-occurring with "e" in each batch are $\langle a : 1,0,1; b : 2,0,1; c : 1,2,2; d : 0,1,1\rangle$.

Using formula (3), we get the maximum adaptive weighted support of items as $\langle a : 1.4; b : 2.1; c : 3.3; d : 1.3\rangle$. Since AWminsupp equal to 1.4, node "d" is moved from the tree. After removing node "d", we obtain the conditional tree of "e" which is presented in Fig. 2(b).

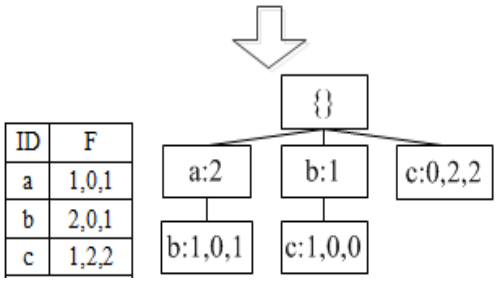
From this conditional tree, using (3), we get three candidate 2-itemsets "ce", "be" and "ae" with the maximum adaptive weights 3.3, 2.1 and 1.4 respectively. By adding these 2-itemsets into set L, we get $L = \{a, b, c, d, e, ae, be, ce\}$.

We continue to mine by developing the itemsets "ae", "be" and "ce". The conditional tree of "ae" is empty which give no candidate itemset. Mining the conditional tree of "be" gives a candidate itemset "abe" with the maximum adaptive weight 1.4. Mining the conditional tree of "ce" gives no candidate itemset because there is only one co-appearing 3-itemset "bce" which has the maximum adaptive weight 0.7 less than AWminsupp 1.4. So, we have

$$L = \{a, b, c, d, e, ae, be, ce, abe\}.$$



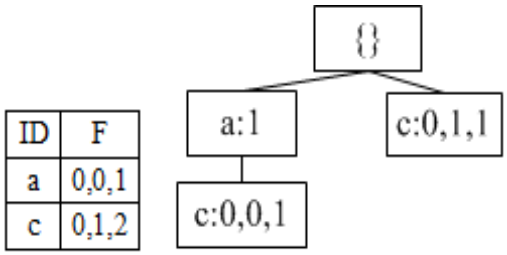
(a) The AWFI-tree construction from the conditional database of "e"



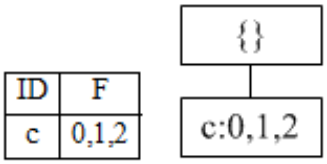
(b) The conditional tree of "e"

Fig.2. The AWFI-tree and conditional tree of "e"

3.2. Construction and mining the conditional tree of "d". The conditional database of "d" contains prefix paths $\{ac : 0,0,1; c : 0,1,1\}$. From this conditional database we have the AWFI-tree shown in Fig. 3(a).



(a) The AWFI-tree construction from the conditional database of "d"



(b) The conditional tree of "d"

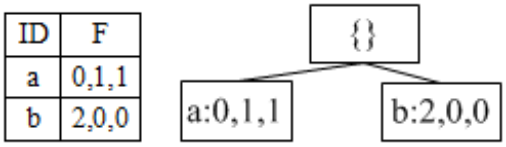
Fig.3. The AWFI-tree and conditional tree of "d"

Since the conditional database of "d" contains only three items "a", "c" and "d", we have $MAXW(1) = 0.5, MAXW(2) = 0.6, MAXW(3) = 0.7$ The frequency values of items occurring together with "d" in each batch are $\langle a : 0,0,1; c : 0,1,2\rangle$. Using (3), we get the maximum adaptive weighted supports of these items as $\langle a : 0.7; c : 2.0\rangle$. Since AWminsupp equal to 1.4, node "a" is removed from the tree. After removing node "a", we get the conditional tree of "d" as presented in Fig. 3(b). By mining this conditional tree, we get the candidate itemset "cd" with the maximum adaptive weight 2.0. So, we have

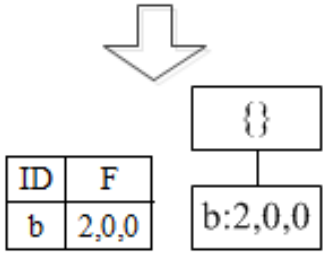
$$L = \{a, b, c, d, e, ae, be, cd, ce, abe\}.$$

3.3. Construction and mining the conditional tree of "c". The conditional database of "c" contains prefix paths $\{a:0,1,1; b:2,0,0\}$. From this conditional database, we have the AWFI-tree shown in Fig. 4(a). This database has three items "a", "b" and "c", so

$$MAXW(1) = 0.7, MAXW(2) = 0.6, MAXW(3) = 0.7$$



(a) The AWFI-tree construction from the conditional database of "c"



(b) The conditional tree of "c"

Fig.4. The AWFI-tree and conditional tree of "c"

The frequency values of items occurring together with "c" in each batch are $\langle a:0,1,1; b:2,0,0\rangle$. Using (3), we obtain the maximum adaptive weighted supports of items as $\langle a:1.3; b:1.4\rangle$. Since AWminsupp is 1.4, node "a" is remove from the tree. After removing node "a", we get the conditional tree of "c" presented in Fig. 4(b). By mining this conditional tree, we get the candidate itemset "bc" with the maximum adaptive weight 1.4. By adding "bc" into L, we have

$$L = \{a, b, c, d, e, ae, bc, be, cd, ce, abe\}.$$

3.4. Construction and mining the conditional tree of "b". The conditional database of "b" contains only one prefix path $\{a:1,0,1\}$, and its conditional tree is empty. We get no candidate.

3.5. The conditional tree of the last item "a". It is also an empty tree.

The set of all candidates that we finally obtain is:

$$L = \{a, b, c, d, e, ae, bc, be, cd, ce, abe\}.$$

4. By checking the adaptive weighted supports of the itemsets in L using (1), we obtain seven adaptive weighted frequent itemsets with the corresponding adaptive weighted supports:

$$a:2.1, b:2.5, c:4.6, e:3.8, be:1.7, ce:2.8, cd:1.5.$$

IV. Experimental Result

In order to evaluate the performance of our proposed algorithm, as in [3], we have performed several experriments on both synthenic dataset T10I4D100K, which is taken from IBM's database (http://www.cs.loyola.edu/~assoc_gen.html), and real-life dataset Mushroom from frequent itemset mining dataset repository (http://fimi.ua.ac.be/data/). These datasets do not provide the weight values of each item. To get these values of each item, we also have generated random numbers in [0.1,0.9]. We compare our algorithm AWFIMiner, with the algorithm AWFPM proposed by Chowhury F. A. et al [3]. Our programs were written in Microsoft Visual C++ 6.0 and run with the Windows 7 operating system on a Pentium dual core 2.70 GHz CPU with 2.00 GB main memory.

The Mushroom dataset contains 8124 transactions and 119 distinct items. Its mean transaction size is 23 items, and it is a dense dataset. We have divided this dataset into 3 and 6 batches. When K = 3, the first and second batches contains 3000 transactions and the third batch contains 2124 remaining transactions. For K= 6, the first batch contains 1124 transactions and all the other batches contain 1400 transactions.

T10I4D100K contains 100000 transactions, 870 distinct items. Its mean transaction size is 10.1, and it is a spare dataset. We have divided this dataset in two way, 3 and 5 batches. For K=3, the first and second batches contains 35000 transactions and the last batch contains 30000 remaining transactions. For K=5, each batch has the same number of transactions which is 20000.

For each batch in both Mushroom and T10I4D100K dataset, we have generated new weight value of each item. For the existing AWFPM algorithm proposed by Chowdhury F. A. and his colleagues, we have only used K=3.

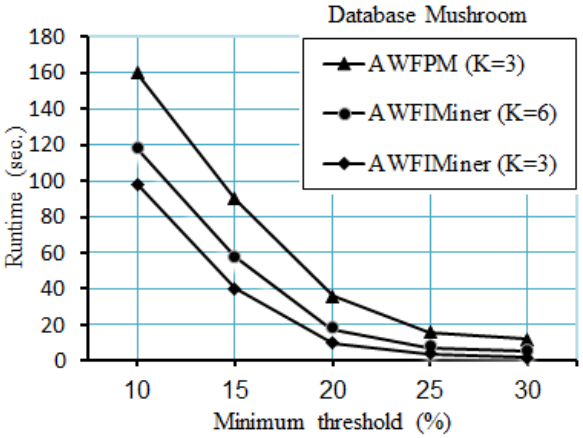


Fig.5. Runtime performance for Mushroom

Fig. 5 shows the runtime performance curves for Mushroom. The minimum support threshold range of 10% to 30% is used. Fig. 6 describe the runtime performance curves for T10I4D100K with the minimum support threshold range used of 1% to 5%.

Experimental results in Fig. 5 and Fig. 6 show that by using an more efficient tree structure, a single database scan and the pattern growth mining approach, the AWFIMiner algorithm performs better than the AWFPM algorithm proposed in [3]. In addition, by using two type of nodes in the tree, AWFIMiner allows us to save huge memory space while still keep all batch by batch transaction information of the database in the prefix tree in a compact format. It requires less memory compared to the existing AWFPM. For example, when the Mushroom dataset is divided into 3 batches, the AWFIMiner requires 0.94 MB while AWFPM requires 1.26 MB memory. When the T10I4D100K dataset is divided into 3 batches, the memory requirements by AWFIMiner and AWFPM are 14.72 MB and 17.61 MB memory respectively.

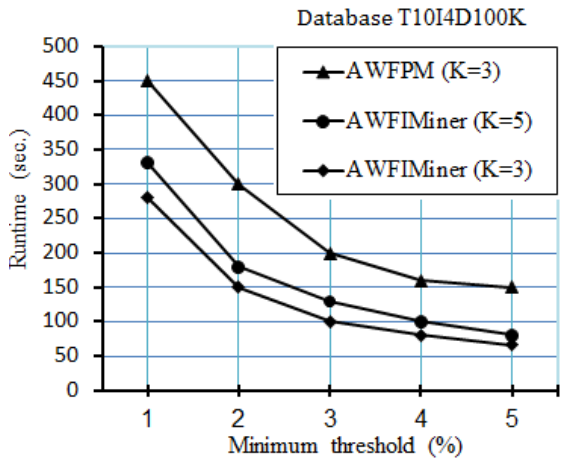


Fig.6. Runtime performance for T10I4D100K

V. Conclusions

Mining weighted frequent itemset plays an important role in many real data mining scenarios. In this paper, we considered the problem of mining weighted frequent itemsets using adaptive weights and proposed an improved algorithm AWFIMiner. The main constribution of this paper is to provide a novel, more compact tree structure AWFI-tree for adaptive weighted frequent itemset mining, and a new measure to maintain the downward closure property. This measure allows us to prune the prefix tree and conditional trees more efficiently when using pattern growth mining approach. Experimental results show that AWFIMiner is an efficient algorithm for mining adaptive weighted frequent itemsets in both dense and sparse datasets. AWFIMiner outperforms the existing AWFPM algorithm in [3]. Moreover, it can handle the whole database information using a single scan of database and therefore applicable for stream data mining.

References

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

[2] Cai, C.H., Fu, A.W.C., Cheng, C.H., Kwong, W.W., Mining association rules with weighted items. In *Proceedings of Intl. Database Engineering and Applications Symposium* (IDEAS 1998), Cardiff, Wales, UK, July 1998, 68–77.

[3] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, Young-Koo Lee, Mining Weighted Frequent Patterns Using Adaptive Weights. In: *Fyfe et al. (Eds.): IDEAL* 2008, LNCS 5326, 2008, 258–265.

[4] Darshan M. Tank, Improved Apriori Algorithm for Mining Association Rules, *Int. Jour. Information Technology and Computer Science*, 2014, 07, 15-23.

[5] 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, 2004, 53–87.

[6] Han, J., H., Xin, D., and Yan, X., Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, vol. 15, 2007, 55–86.

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

[8] Jiang, N., Gruenwald, L., Research Issues in Data Stream Association Rule Mining. *SIGMOD Record* 35(1), 2006, 14–19.

[9] Paray S.M. Tsai, Mining frequent itemsets in data streams using the weighted sliding window model. *Expert Systems with Applications* 36, 2009, 11617-11625.

[10] Tao, F., Weighted association rule mining using weighted support and significant framework. In: 9th ACM SIGKDD *Int. Conf. on Knowledge Discovery and Data Mining, USA*, 2003, 661–666.

[11] Wang, W., Yang, J., Yu, P.S., WAR: weighted association rules for item intensities. *Knowledge Information and Systems* 6, 2004, 203–229.

[12] Yun, U., Leggett, J.J., WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In: *Fourth SIAM Int. Conf. on Data Mining*, USA, 2005, 636–640.

[13] Yun, U., Efficient Mining of weighted interesting patterns with a strong weight and/or support affinity. *Information Sciences* 177, 2007, 3477–3499.

[14] Yun, U., An efficient mining of weighted frequent patterns with length decreasing support constraints. *Knowledge-Based Systems*, Volume 21 Issue 8, December 2008, 741-752.

[15] Zhang, S., Zhang, C.,Yan, X., Post-mining: maintenance of association rules by weighting. *Information Systems* 28, 2003, 691–707.

Authors' Profiles

Hung Long Nguyen is currently a lecturer at Faculty of Economic Information System, Vietnam Com-mercial University (VCU). He recei-ved his B.Sc. degree in Informatics from Hanoi University of Science in 1991, and his M.Sc. degree in Infor-mation Technology from Le Quy Don Technical University in 2002. His research interests include Data Mining, Knowledge Discovery in Databases, Information Systems, and Database.

