ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 4, pp: (102-112), Month: October - December 2014, Available at: www.researchpublish.com

Single-Pass Incremental and Interactive Mining for Weighted Frequent Pattern

[1]Dayanand Sunil Sonavane, [2]Dayanand Suresh Chilap, [3]Parag Jalindar Davkhar, [4]Mayur Gulab Varpe

[1,2,3,4]Department of Computer Engineering, Jaihind college of Engg. Kuran, Junnar, Pune,410511

(Savitribai Phule, Pune University)

Abstract: **Weighted frequent pattern (WFP) mining is more practical than frequent pattern mining because it can consider different semantic significance (weight) of the items. For this reason, WFP mining becomes an important research issue in data mining and knowledge discovery. However, existing algorithms cannot be applied for incremental and interactive WFP mining and also for stream data mining because they are based on a static database and require multiple database scans. In this paper, we present two novel tree structures IWFPTWA (Incremental WFP tree based on weight ascending order) and IWFPTFD (Incremental WFP tree based on frequency descending order), and two new algorithms IWFPWA and IWFPFD for incre- mental and interactive WFP mining using a single database scan. They are effective for incremental and interactive mining to utilize the current tree structure and to use the previous mining results when a database is updated or a minimum support threshold is changed. IWFPWA gets advantage in candidate pattern generation by obtaining the highest weighted item in the bottom of IWFPTWA. IWFPFD ensures that any non-candidate item cannot appear before candidate items in any branch of IWFPTFD and thus speeds up the prefix tree and conditional tree creation time during mining operation1.**

I. INTRODUCTION

In practice, the frequency of a pattern may not be a sufficient indicator for finding meaningful patterns from a large transaction database because it only reflects the number of transactions in the database which contain that pattern. In many cases, an item in a transaction can have different degree of importance (weight). For example, in retail applications an expensive product may con- tribute a large portion of overall revenue even though it does not appear in a large number of transactions. For this reason, weighted pattern was proposed to discover more important knowledge considering different weights of each item, which plays an important role in the real world scenarios. Weight-based pattern mining approach can be applied in many areas, such as market data analysis where the prices of products.

To discover knowledge or patterns from data streams, it is necessary to develop single-scan and on-line mining methods.

Motivated from these real world scenarios, we propose a pat- tern mining approach which can solve these problems at one time. In this paper, we propose two novel tree structures IWFPTWA (Incremental weighted frequent pattern tree based on weight ascending order) and IWFPTFD (Incremental weighted frequent pattern tree based on frequency descending order). Both of them can handle incremental data in a single-scan of database and they have the ‘‘build once mine many’’ property for interactive mining. Based on the above tree structures, we develop two new algorithms IWFPWA and IWFPFD. They exploit a pattern growth mining approach. Our proposed first tree structure IWFPTWA arranges the items in weight ascending order and can be constructed without any restructuring operation. IWFPWA gets advantage in candidate pattern generation by obtaining the highest weighted item in the bottom of the IWFPTWA. However, this weight ascending order does not guarantee that candidate items come before the non-candidate items in any branch of the tree. Hence, lots of non-candidate items may come in between the candidate items inside a tree.

The remainder of this paper is organized as follows. In Section 2, we describe background. In Section 3, we develop our proposed tree structures for incremental and interactive weighted frequent pattern mining and show how they can handle additions, deletions and modifications of transactions. In Section 4, we describe our proposed algorithms for incremental and interactive weighted frequent pattern mining and analyze their performances. In Section 5, our experimental results are presented and analyzed. In Section 6, we elaborately discuss the practical application areas of weighted frequent pattern mining. Finally, in Section 7, conclusions are presented.

2. BACKGROUND

In the following subsection we discuss about the background and related research works on frequent pattern mining, incremental and interactive pattern mining and weighted frequent pattern mining. Subsequently, we describe the main challenging problem in weighted frequent pattern mining.

2.1. Problem definitions and related Work

2.1.1. Frequent pattern mining

The support/frequency of a pattern is the number of transactions containing the pattern in the transaction database. The problem of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The downward closure property is used to prune the infrequent pat- terns. This property tells that if a pattern is infrequent then all of its super patterns must be infrequent. Apriori algorithm is the initial solution of frequent pattern mining problem and very useful in association rule mining But it suffers from the level-wise candidate generation-and-test problem and needs several database scans. FP-growth (frequent pattern growth) (Han, Pei, Yin, & Mao, 2004) solved this problem by using FP-tree-based solution without any candidate generation and using only two database scans. Many other research works have been done to devise new algorithms or improve the existing algorithms for finding frequent patterns. Moreover, pattern growth technique is also very useful for sequential pattern mining (Pei et al., 2004). However, this traditional frequent pattern mining considers equal weight (profit) for all items.

2.1.2. Incremental and interactive pattern mining

As for incremental mining, we mean a kind of mining techniques which can be well applied for the dynamic environment where a database grows and changes frequently. Interactive mining means that repeated mining with different minimum support threshold values can be possible by utilizing the same data 2009) improves Can Tree by restructuring the incremental prefix- tree according to the frequency descending order and thus, gets remarkable mining time improvement compared to Can Tree. These research works have shown that their incremental prefix-tree structures are quite possible and efficient using currently available memory size in gigabyte range. Some other single-pass mining algorithms have been developed to find out frequent patterns over a data stream in real time. Efficient dynamic database updating algorithm (EDUA) Zhang et al. (2007) is designed for mining databases when data deletion is carried out frequently in any subset of a database. Inc- WTP and Wss WTP algorithms (Lee & Yen, 2008) are designed for incremental and interactive mining of web traversal patterns.

2.1.3. Weighted frequent pattern mining

Let I = {i1, i2, ... , im} be a set of items and D be a transaction database {T1, T2, ... , Tn} where each transaction Ti e D is a subset of I. A pattern or item set is defined by the set X = {x1, x2, ... xk},

where X # I and k e [1, m]. However, an itemset is called k-item set

when it contains k distinct items. For example, ‘‘ab’’ is a 2-itemset and ‘‘abd’’ is a 3-itemset in Fig. 1.

A weight of an item is a non-negative real number assigned to reflect the importance of the item in the transaction database. The weight of a pattern, P{x1, x2, ... , xk} is given as follows:

A pattern is called a weighted frequent pattern if the weighted support of the pattern is greater than or equal to the minimum threshold. Consider the minimum threshold is 1.5 in Fig. 1 and then ad is a weighted frequent pattern.

Table 1 shows an example of a retail database in which the nor- malized weight values are assigned to items based on the price of each item. Normalization process is needed for adjusting the differ- ences among data of varying sources to create a common basis for comparison (Yun, 2007a; Yun & Leggett, 2005a, 2005b, 2006). According to the normalization process, the final weights of items can be determined within a specific weight range. For example, in Table 1 the weight values of items are given in the range from 0.05 to 0.8.

In the very beginning, some weighted frequent pattern mining algorithms MINWAL , WARM (Tao, 2003), WAR (Wang et al., 2004) have been developed based on the Apriori algorithm using the level-wise candidate generation-and-test para digm. Obviously, these algorithms require multiple database scans and result in poor mining performance. WFIM (Yun & Legg- ett, 2005a) is the first FP-tree based weighted frequent pattern mining algorithm using 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 using weight ascending order and maintained the down-ward closure property on that tree. WLPMINER & Leggett, 2005b) algorithm finds weighted frequent patterns using length decreasing support constraints. WCloset (Yun, For example, Weight(ad) = (0.6 + 0.35)/2 = 0.475 in the example database of Fig. 1.

A weighted support of a pattern is defined as the resultant value of multiplying the pattern's support with the weight of the pattern. So the weighted support of a pattern P is given as follows:

W supportðPÞ ¼ WeightðPÞ SupportðPÞ

2007c) is proposed to calculate the closed weighted frequent pat- terns. To extract the more interesting weighted frequent patterns, WIP (Yun, 2007a; Yun & Leggett, 2006) algorithm introduces a new measure of weight-confidence to measure strong weight affinity of a pattern. It has used another measure hyper clique confidence (Xiong, Tan, & Kumar, 2006) to measure strong support affinity of a pattern. Except that WFIM and WIP use the different pruning conditions to find out interesting patterns, overall mining procedures are almost same. It means that both of them require two database scans which are not suitable for either stream data mining or incremental/interactive mining.

2.2. Main challenging problem in weighted frequent pattern mining

WFIM (Yun & Leggett, 2005a) and WIP (Yun, 2007a; Yun & Legg- ett, 2006) pointed out the main challenging problem of weighted frequent pattern mining is, weighted frequency of a pattern (or an itemset) does not have the downward closure property. Consider the example database of Fig. 1. The item ‘‘a’’ has a weight of 0.6 and a frequency of 5, the item ‘‘d’’ has a weight of 0.35 and a frequency of 4, the pattern ‘‘ad’’ has a frequency of 4. According to Eq. (1), the weight of the pattern ‘‘ad’’ is 0.475 and according to Eq. (2) its weighted support is 1.9. Weighted support of ‘‘a’’ is 0.6 5 = 3.0 and ‘‘d’’ is 0.35 4 = 1.4. If the minimum threshold is 1.5, then pat- tern ‘‘d’’ is weighted infrequent but ‘‘ad’’ is weighted frequent, i.e., downward closure property is not satisfied here. WFIM and WIP maintain downward closure property by multiplying each pattern support by the global maximum weight. In the above example, item ‘‘a’’ has the maximum weight which is 0.6, then by multiplying it with the support of item ‘‘d’’, 2.4 can be obtained. So, pattern ‘‘d’’ is not pruned at the early stage and pattern ‘‘ad’’ will not be missed. At the final stage, this overestimated pattern ‘‘d’’ will be pruned finally by using its actual weighted support.

weighted frequent patterns containing the ‘‘build once mine many’’ property. However, this paper includes substantively novel and different contributions beyond the preliminary conference version (C.F. Ahmed et al., 2008) including new tree structure and algorithms which are more efficient with respect to runtime and memory, enhanced motivation with real-life applications, rigorous analysis of the mining as well as overall performance of the algorithms, elabo- rate descriptions to explain how to apply our tree structures and algorithms for incremental and interactive WFP mining with new figures and tables, a thorough presentation of the background, and comprehensive experimental results with discussions.

3. OUR PROPOSED TREE STRUCTURES

In this section, at first we describe the overall prefix tree structure for better understanding of our proposed tree structures. After that, we describe the construction process of our proposed tree structures and show that how additions, deletions and modifications are possible inside the tree structures. Similar to FP-tree (Han et al., 2004), a header table is maintained to keep an item order in both the tree structures. Each entry in a header table explicitly maintains item-id, frequency and weight information for each item. However, each node in a tree only maintains item-id and frequency information. To facilitate the tree traversals adjacent links are also maintained (not shown in the figures for simplicity) in our tree structures like FP-tree. We will use the term IWFPT to denote two tree structures together.

3.1. Overall prefix tree structure

A prefix tree is an ordered tree with any predefined item order such as lexicographic order, frequency ascending or descending order, and weight ascending and descending order. We can read transactions one by one from a transaction database and insert it into the tree according to any predefined order. In this way prefix tree can represent a transaction database in a very compressed form when different transactions have many items in common. This type of path overlapping is called prefix-sharing. The more the prefix-sharing the more compression we can achieve from the prefix tree structure. FP-growth (Han et al., 2004) algorithm introduced a prefix tree structure named FP-tree, which arranges the items in frequency descending order. They have shown that, by arranging the items according to that order huge prefix-sharing can be achieved. They have also shown that, in their mining operation when they create prefix and conditional trees for a particular item, the trees are also compact in size having huge prefix-sharing. As a consequence, they can achieve a lot of gain in overall mining time.

As discussed in Section 2.1.3, the previous prefix-tree-based algorithms for weighted frequent pattern mining (Yun, 2007a, 2007c, 2008b; Yun & Leggett, 2005a, 2005b, and 2006) are based on weight-ascending order of items. The main advantage of weight- ascending ordered prefix tree structures is to get the maximum weighted item in the bottom of a tree. Accordingly, when we go for bottom-up mining, the local maximum weight may reduce in each step (we will explain in Section 4.1). The previous works are based on a static database. In the first database scan they calculate the single-element candidates and in the second scan they mine the other candidates and finally select the actual weighted frequent patterns. Our first proposed incremental prefix tree structure IWFPTWA is based on the weight ascending order to perform incremental and interactive weighted frequent pattern mining. It has the advantage of the previous algorithms, but as it keeps the incremental data in weight ascending order, it has a poor prefix-sharing. Therefore, prefix-tree size becomes large and non- candidate nodes are present between the candidate nodes inside the tree. This enlarges the overhead of the prefix tree and conditional tree creation for a particular item/itemset during mining process. To solve these problems we propose our second tree structure IWFPTFD, which arranges the incremental prefix tree in frequency descending order by restructuring operation to get the advantages of FP-tree.

3.2. IWFPTWA

Our first proposal First of all, we describe the construction process of IWFPTWA (incremental weighted frequent pattern tree based on weight ascending order), then we show how it can handle insert and up- date operations.

Consider the example database in Fig. 1a. We scan the transactions one by one, sort the items in a transaction according to the weight ascending order and then insert into the tree. A header table is also maintained to keep all the items in the weight ascending order. The first transaction T1 has the items ‘‘a’’, ‘‘b’’, ‘‘c’’, ‘‘d’’, ‘‘g’’, and ‘‘h’’. After sorting, the new order will be ‘‘c’’, ‘‘d’’, ‘‘h’’, ‘‘g’’, ‘‘b’’, and ‘‘a’’. Fig. 2a shows IWFPTWA after inserting T1. Fig. 2b shows IWFPTWA after inserting T2. In the same way, all the transactions up to T6 are inserted into the tree. Fig. 2c shows the final IWFPTWA after inserting T6.

The term ‘‘db+’’ is used to denote a group of transactions to be added. Here ‘‘db’’ stands for database and ‘‘+’’ stands for addition of transactions. Similarly, ‘‘db ’’ and ‘‘dbmod’’ are used to denote a group of transactions to be deleted and modified, respectively. The original database presented in Fig. 1a is incremented by adding two groups of transactions dbþ and dbþ as shown in Fig. 3a. Fig. 3b Consider the example database in Fig. 1a. The initial IWFPTFD can be constructed in the same way of IWFPTWA (shown in Fig. 2). After adding all the six transactions of Fig. 1a, the IWFPTWA (shown in Fig. 2c) is sorted/restructured according to the frequency descending order by using a path adjusting method (Koh & Shieh, 2004) (also known as bubble sort method for tree restructuring) which is shown in Fig. 6a. According to the path adjusting method, any node may be split when it needs to be swapped with any child node having count smaller than that node. If the support counts of both nodes are equal, simple exchange operation be- tween them is performed. After each swapping operation, nodes having same items (if any due to the swapping) are merged to- gether. CP-tree (Tanbeer et al., 2009) shows that overall restructuring time is less if it is done slot-by-slot in the database compared to restructuring done on the full tree at the end. There- fore, we also perform this tree restructuring operation in a way of slot-by-slot fashion. It is remarkable that IWFPTFD (Fig. 6a) has only 15 nodes (without root) compared to 22 nodes in the IWFPT- WA

It has only 16 nodes compared to 27 nodes of IWFPTWA (shown in Fig. 5). The following properties are true for both IWFPTWA and IWFPTFD.

Property 2: The total count of frequency value of any node in IWFPT is greater than or equal to the sum of total counts of frequency values of its children.

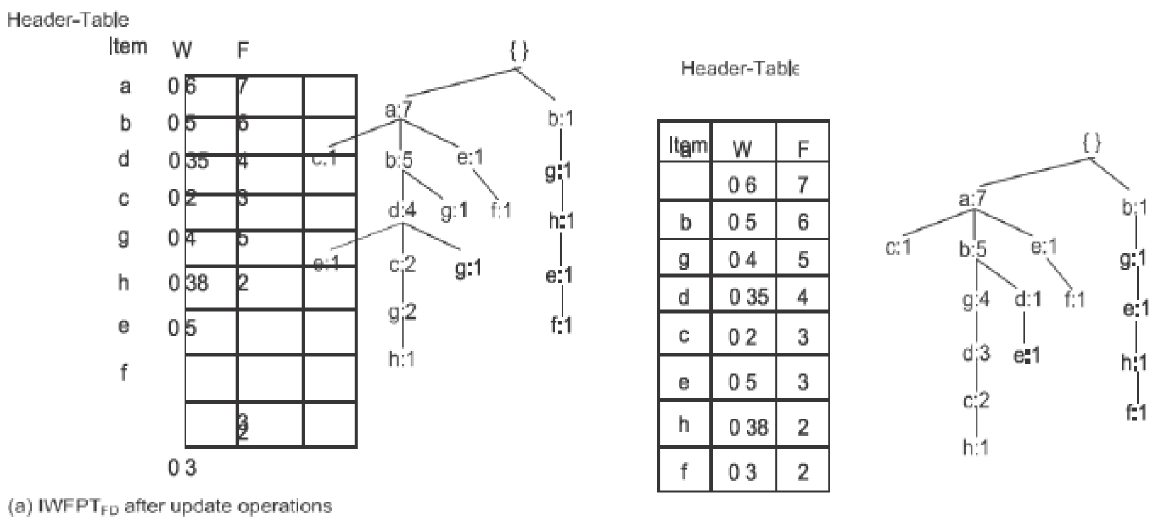
Proof. In the proposed algorithm, every transaction is inserted according to the weight ascending/frequency descending order of items. Consider X is the parent node of Y in path P1 of the tree and frequency value of X and Y are Fx and Fy respectively. The frequency value of Y is increased when a transaction contains items.

3.3. IWFPTFD

Our second proposal Here we first describe the construction process of IWFPTFD (incremental weighted frequent pattern tree based on frequency descending order). Afterwards, we show how it can handle insertions, deletions and modifications. one transaction contains aX, then only Fx is incremented and Fx becomes larger than Fy. Therefore, Fx P Fy. h

Property 3:IWFPT can be constructed in a single database scan.

Proof. In the proposed algorithm, one transaction is scanned from a database and is inserted into IWFPT in the weight ascending/frequency descending order. When a new group of transaction is added, the algorithm needs to traverse the incremented part only. Therefore, the algorithm needs to scan a transaction exactly once from a database and it can be said that an IWFPT can be constructed in a single scan of a database. H



Header-Table

Item	W	F
a	0 6	7
b	0 5	6
d	0 35	4
c	0 2	3
g	0 4	5
h	0 38	2
e	0 5	
f		
		3 2
	0 3	

(a) IWFPT$_{FD}$ after update operations

Header-Table

Item	W	F
	0 6	7
b	0 5	6
g	0 4	5
d	0 35	4
c	0 2	3
e	0 5	3
h	0 38	2
f	0 3	2

Fig. 7. Update operations in IWFPTFD.

3.4. Handling insertion of a new item with a different weight

Our proposed tree structures can also handle the situation efficiently when a new item comes with a different weight. Consider the current updated database (Fig. 3b) is incremented by .1. Mining process in IWFPWA In FP-growth mining algorithm (Han et al., 2004), when a prefix tree is created for a particular item, all the branches prefixing that item are taken with the frequency value of that item. After that, a adding dbþ which contains a new item ‘‘x’’ with a different conditional tree is created from the prefix tree by deleting the weight of 0.45. Figs. 8a and b show the modified database and weight table, respectively. Fig. 8c shows the resultant IWFPTWA with header table after inserting dbþ . The new item ‘‘x’’ is inserted into the header table according to its weight value. How- ever, the previous ordering of items is not affected with the insertion of this new value, i.e., if the previous order of two items ij and ik is ij < ik, then it still holds. Therefore, Property 1 holds for IWFPTWA in this situation also and it does not need any restructuring operation. We do not need to traverse any part of the database twice due to this type of insertion process. Hence, Property 3 holds for IWFPTWA. Fig. 8c shows that the first transaction of dbþ, ‘‘a b x’’, is arranged as ‘‘x b a’’ in weight ascending order and inserted as a new branch in IWFPTWA. On the other hand, the second transaction, ‘‘f x’’ gets a prefix-sharing with one existing child ‘‘f’’ node of the root and new item ‘‘x’’ is inserted as a new child of ‘‘f’’. Accordingly, this insertion process is also similar to the previous insertion process and does not change the frequency relationship between the parent and child nodes. As a consequence, Property 2 also holds for IWFPTWA in this situation.

Fig. 8d shows the resultant IWFPTFD after inserting dbþ and Fig. 8e shows the IWFPTFD after restructuring operation. These figures explain that although a new item with different weight is inserted into an IWFPTFD, frequency descending order of items can still be achieved with restructuring operation. However, this process does not need to traverse any part of the database twice. As a result, Property 3 holds for IWFPTFD. In a similar way of IWFPTWA, it can be shown from these figures that Property 2 also holds for IWFPTFD in this situation. Furthermore, IWFPTFD can still achieve a significant compression gain over IWFPTWA. For the current data- base of Fig. 8a, IWFPTFD has only 19 nodes compared to 31 nodes of IWFPTWA.

4. OUR PROPOSED ALGORITHMS

In this section, at first we describe the mining process of our proposed IWFPWA and IWFPFD algorithms using our proposed IWFPTWA and IWFPTFD tree structures, respectively. Then we analyze their mining performances and describe how they are effective in interactive mining. Finally, we formally present and describe our proposed algorithms nodes containing infrequent items. IWFPWA does the same type of mining operation. As discussed in Section 2.2, the main challenging problem in this area is that the weighted frequency of an item- set does not have the downward closure property and to utilize this property we have to use the global maximum weight. The global maximum weight, denoted by GMAXW, is the maximum weight of all the items in the whole database. For example, in Fig. 1b, the item ‘‘a’’ has the global maximum weight of 0.6. Local maxi- mum weight, denoted by LMAXW, is needed when we are doing the mining operation for a particular item. As IWFPTWA is sorted in weight ascending order, we get advantage here in the bottom-up mining operation. For example, after mining the weighted frequent patterns for the item ‘‘a’’, when we go for the item ‘‘b’’, then the item ‘‘a’’ will never come in its prefix and conditional trees. As a result, now we can easily assume that the item ‘‘b’’ has the maxi- mum weight. This type of maximum weight in mining process is known as LMAXW. As LMAXW is reducing from bottom to top, the probability of a pattern to be a candidate is also reduced.

Consider the current database presented at Fig. 8a, weight table of items at Fig. 8b, IWFPTWA constructed for that database at Fig. 8c, and minimum threshold = 2.2. Here GMAXW = 0.6. After multiplying the frequency of each item with GMAXW, the weighted frequency list is hc: 1.8, f: 1.8, d: 2.4, h: 1.2, g: 3.0, x: 1.2, e: 1.8, b: 4.2, a: 4.8i. As a result, the candidate items are ‘‘d’’, ‘‘g’’, ‘‘b’’, and ‘‘a’’. Now we construct the prefix and conditional trees for these items in a bottom-up fashion and mine the weighted frequent patterns. At first the prefix tree of the bottom-most item ‘‘a’’ is created by taking all the branches prefixing the item ‘‘a’’ as shown in Fig. 9a. To create the conditional tree for the item ‘‘a’’, we have to delete the nodes from its prefix tree which cannot form candidate pattern with item ‘‘a’’. Observe that the prefix tree of the item ‘‘a’’ (shown in Fig. 9a) contains global weighted infrequent items ‘‘c’’, ‘‘f’’, ‘‘h’’,

‘‘e’’, and ‘‘x’’. Without any calculation we can delete these nodes. After that we multiply the frequency of other items by LMAXW. As we are now dealing with the bottom-most item ‘‘a’’, LMAXW = GMAXW = 0.6. The weighted frequency list is hd: 2.4, g: 2.4, b: 3.6i. Accordingly, we cannot prune them now because any one of these three items could form weighted frequent pattern with the item ‘‘a’’. The conditional tree created for the item ‘‘a’’ is shown in Fig. 9b. Candidate patterns ‘‘ad’’, ‘‘ag’’, ‘‘ab’’, and ‘‘a’’ are generated here.

Candidate patterns with their actual weights and the weighted frequency using Eqs. (1) and (2) respectively and mine the actual weighted frequent patterns. Table 2 shows these calculations. The actual weighted frequent patterns.

4.2. Mining process in IWFPFD

To reduce the huge number of nodes in the prefix and conditional trees in IWFPWA, IWFPFD is designed based on IWFPTFD. IWFPFD uses same technique for creating prefix and conditional trees in mining operation. As it is sorted according to the frequency descending order, LMAXW could be anywhere for a particular item. Here we start the pattern growth mining operation from the top- most item. Obviously LMAXW here is the weight of the first item. After that for the second item, we compare the weight of the second item with previous LMAXW and choose the larger one as cur- rent LMAXW. For example, if the weight of second item is 0.7/0.6 and previous LMAXW (i.e., weight of the first item) is 0.6/0.7, LMAXW for second item is 0.7. By moving in this way we can save the LMAXW calculation for each item.

Here we have shown the worst case situation of our IWFPTFD in Fig. 8e. The top-most item ‘‘a’’ has the highest weight, i.e., LMAXW for all the candidate items below ‘‘a’’ (e.g., ‘‘b’’, ‘‘g’’, ‘‘d’’) is 0.6 which is also the GMAXW. This situation could generate some extra candidates. Starting from the top-most item, at first, the candidate pat- tern ‘‘a’’ is generated. After that, we go to the item ‘‘b’’. Fig. 10a shows the prefix and conditional tree for the item ‘‘b’’ that only contains the item ‘‘a’’. LMAXW = 0.6 for the item ‘‘b’’. Candidate pat- terns ‘‘ab’’ and ‘‘b’’ are generated here. Fig. 10b shows the prefix and conditional tree for the item ‘‘g’’. Again, LMAXW = 0.6 for the item ‘‘g’’. Candidate patterns ‘‘ag’’, ‘‘bg’’, and ‘‘g’’ are generated here. The prefix and conditional tree for the itemset ‘‘bg’’ is shown in Fig. 10c. Pattern ‘‘abg’’ is generated here. The prefix tree for the item ‘‘d’’ is shown in Fig. 10d. The conditional tree of item ‘‘d’’ is created by deleting the item ‘‘g’’ as it has lower weighted frequency with item ‘‘d’’ compared to the minimum threshold (shown in Fig. 10e). Candidate patterns ‘‘ad’’, ‘‘bd’’, ‘‘abd’’, and ‘‘d’’ are generated here. Table 3 shows the calculations of these candidate pat- terns. One extra pattern ‘‘bd’’ is generated here which is not generated by IWFPWA. Observe that no global weighted infrequent items like ‘‘c’’, ‘‘f’’, ‘‘h’’, ‘‘e’’, ‘‘x’’ appear in any prefix and conditional trees during mining in IWFPFD.

4.3. Analysis of mining performance

FP-growth (Han et al., 2004) showed that we can get a huge pre- fix-sharing among items and a very compressed tree by sorting the items in frequency descending order. They also showed that by arranging nodes in such order infrequent nodes cannot appear be- tween the frequent nodes. As IWFPTWA arranges items in weight ascending order, it has a poor prefix-sharing and that takes more memory. Moreover, non-candidate nodes frequently occur be- tween weighted frequent candidate nodes. Therefore, there are many global weighted infrequent nodes in the prefix trees of the candidate items in IWFPWA. It increases the prefix tree creation time. These non-candidate nodes have to be deleted while creating the conditional tree from the prefix tree. These deletions also in- crease the conditional tree creation time. As a consequence, the whole mining process is delayed remarkably. This situation can be explained in the mining process of IWFPWA in Section 4.1 and Fig. 9. Another remarkable thing is that, when we are creating the prefix trees for candidate items, we may have to traverse the whole tree and each node may appear inside the prefix tree. For example, to create prefix trees of candidate items in Fig. 9, we have to traverse almost all the nodes (30 nodes out of 31 nodes) of the IWFPTWA (Fig. 8c). Accordingly, these 30 nodes are also present at least in one prefix tree. This means almost all the nodes in the tree are participating during the mining operation. To get rid of these problems, IWFPTFD is designed in frequency descending order. As the candidates of weighted frequent patterns are taken by multi- plying GMAXW, the frequency descending order is also the maxi- mum weighted frequency descending order. Hence, by arranging the items in this order IWFPTFD keeps all the non-candidate items after the candidate items in each branch, i.e., no non-candidate item can appear in IWFPTFD between the candidate items. More- over, while prefix trees are created from the main tree, we do not have to traverse the non-candidate nodes. Only seven nodes of the tree (shown in Fig. 8e) have to be traversed to create the pre- fix trees as shown in the mining process of IWFPFD (in Fig. 10).

In spite of these drawbacks, IWFPWA has two advantages over IWFPFD. The first one is, in IWFPWA, when we go for mining opera- tion for the bottom-most item ‘‘a’’ (in Fig. 8c), then LMAXW is the weight of ‘‘a’’, after that for mining the next to bottom-most item ‘‘b’’ (in Fig. 8c), LMAXW is the weight of ‘‘b’’, and so on. The second advantage is, as the LMAXW is reducing from bottom to top, the probability of a pattern to be a candidate will also be reduced. For example, it is shown in Sections 4.2 and 4.3 that ‘‘bd’’ is a candidate pattern in IWFPFD but not in IWFPWA for minimum thresh- old 2.2. Table 4 shows the performance comparison between IWFPWA and IWFPFD using different criteria for minimum threshold 2.2 in the example database presented at Fig. 8a.

Lemma 1. The number of nodes participating during the mining operation in IWFPFD is always less than or equal to the number of nodes participating during the mining operation in IWFPWA.

Proof. All the nodes containing non-candidate items cannot appear any prefix tree of the candidate items because IWFPTFD is sorted in frequency descending order. But, for IWFPTWA, this item may appear anywhere in the tree. If it appears anywhere except the last node in any branch then it will appear in the prefix trees can be discovered in weight ascending/frequency descending order from an IWFPTWA/IWFPTFD.

Proof. Let X= {x1, x2, ... , xn} and Y= {y1, y2, ... , yn} be two transactions and their items are sorted in weight ascending order. When we want to insert X into an IWFPTWA, we need to check whether the root node contains any child node with item-id x1. If a match is found then the frequency value of x1 is added there, otherwise, a new node is created. The similar process is recursively applied for other items x2–xn in other levels of the tree. Since the root of any sub-tree does not have more than one child with the same item-id, items of Y shares the same path of X if they are identical, i.e., x1 = y1, x2 = y2, xn = yn. As a consequence, for any transaction, there is a unique path in an IWFPTWA starting from the root.

By considering the items of these two transactions are sorted in frequency descending order, it can be shown that IWFPTFD also has a unique path for each transaction starting from the root. h

IWFPTWA/IWFPTFD can exactly represent the updated database DB ± dbmod. h

Lemma 4 proves that when a database is updated by additions/ deletions/modifications of transactions, our proposed tree structures IWFPTWA and IWFPTFD are also updated properly and always represent the complete/correct set of current transactions in a database. Therefore, despite many additions/deletions/modifications of transactions, our algorithms always mine complete/correct set of weighted frequent patterns by using the tree structures IWFPTWA/IWFPTFD according to a user-given minimum threshold.

4.5. Analysis of space requirements

Even though IWFPTWA and IWFPTFD have a unique path for each transaction, the following observation shows that they have a lot of prefix-sharing (discussed in Section 3.1).

Observation 1. In an IWFPTWA/IWFPTFD, two transactions X= {x1, x2, ... , xn} and Y= {y1, y2, .. . , ym} share the first node if x1 = y1, first and second node if x1 = y1 and x2 = y2, and so on. Stated in other way, a transaction X = {x1, x2, ... , xn} is totally isolated in an IWFPTWA if no other transaction in DB starts with x1. For example, consider X= {a, b, c}, then X is totally isolated in an IWFPTWA/IWFPTFD if no other transaction starts with item a. The

existing research shows that probability of this event is very low and each path in a prefix tree achieves some prefix-sharing. It is also discussed in Section 3.1 that how much prefix-sharing will be achieved by a prefix tree is dependent on the ordering of items. Moreover, it is also shown in Section 3 that IWFPTFD achieves a larger prefix-sharing compared to IWFPTWA since it stores the items in frequency descending order.

However, according to Observation 1, an IWFPTWA/IWFPTFD usually achieves a lot of prefix-sharing due to the common prefix items inside transactions. Hence, the size of an IWFPTWA/IWFPTFD given minimum threshold. Even though FP-tree requires a less amount of memory, it must be constructed from the very beginning when the minimum support threshold is changed or the data- base is updated. As a consequence, it is not applicable for incremental or interactive mining. Moreover, it considers binary appearances of items inside transactions and not suitable for weighted frequent pattern mining.

4.6. Interactive mining performance

IWFPT has the ‘‘build once mine many’’ property that means after creating one tree; several mining operations can be performed using different minimum thresholds. For example, after the creation of IWFPT at first, we give the minimum threshold = 2.2, then after finding the result we can give another minimum threshold like 2.0 or 2.5. Same operation can be done several times. Therefore, after the first time, we do not have to create the tree again. If the current minimum threshold is larger than the previous one then we do not have to per- form mining operation as well. Because the resultant patterns of cur- rent mining threshold is a subset of the result we have got for the previous threshold. For example, after performing the mining operation for the minimum threshold = 2.2, if we go for mining operation with the minimum threshold = 2.5 then we do not need to perform mining operation again but if the new minimum threshold = 2.0 then we have to perform mining operation again.to denote the global maximum weight. In line 4, the root R of an IWFPTWA is created and initialized to NULL. In line 5, the next transaction Ti is scanned and new items inside it are inserted in H according to the weight ascending order. All the items of Ti are also sorted in this order in line 6. Procedures of additions/deletions/modifications of transactions are invoked from lines 7 to 13. In line 14, it checks whether the current transaction is the last transaction of the main DB or any db+/db /dbmod. If the above condition is true, then the current GMAXW is defined and minimum threshold (d) input is taken from the user in lines 15 and 16, respectively. The conditional statement in line 17 checks the cur- rent d with the previous one to take advantages of interactive mining stated in Section 4.6. IWFPWA algorithm scans the header table from the bottom to take the advantage of weight ascending order. In line 19, it checks whether the maximum weighted frequency of an item satisfies d and then it calls the Test Candidate procedure (shown in Fig. 13) in line 20. This procedure calculates the actual weighted frequency of a pattern and based on that result adds a pattern in the weighted frequent pattern list. In line 21, prefix tree with header table is created for a particular item and recursive Mining procedure is called in line 22.

The Insert procedure is presented in Fig. 12. It recursively inserts the elements of a transaction in an IWFPTWA/IWFPTFD. It receives one transaction and the current root of a sub-tree where the front element of the transaction has to be inserted as a child. Please note that the received transaction is already sorted in weight ascending/frequency descending order by the caller algorithm IWFPWA/IWFPFD.

The ''if condition'' in lines 2–4 tests whether or not the received transaction is empty. The procedure returns when the received transaction is empty. In line 5, the front element, x, of the received transaction is taken to insert as a child of the current root R and remaining elements are taken from the transaction for the next recursion. The ''if-else condition'' in lines 6–12 tests whether or not a match is found for x in the child nodes of R. If the item-name of x is matched with the item-name of any child node of R, then we need to increment the count value of the existing node by one. However, it creates a new child node of the current root if it fails to find any match. The new node is initialized with the item-name of x and count value of 1(lines 10 and 11). The count value of header table entry for the item is also incremented by one (line 13). In line 14, the procedure recursively calls itself with the remaining items of the transaction and current child node as the root.

The Delete procedure is also presented in Fig. 12. It recursively deletes the elements of a transaction in an IWFPTWA/IWFPTFD. It receives one transaction and the current root of a sub-tree where the front element of the transaction has to be deleted. The ''if condition'' in lines 2–4 tests whether or not the received transaction is empty. The procedure returns when the received transaction is empty. In line 5, the front element, x, of the received transaction is taken to delete from the current root R and remaining elements are taken from the transaction for the next recursion. In line 6, we search for a child node C of R containing the same item_id as x. The count value of C is then decremented by one along with its header table entry (lines 7 and 8). In line 9, the procedure recursively calls itself with the remaining items of the transaction and current child node as the root. After coming back from recursive call, it is checked that whether or not count value of C becomes zero in that node. If it becomes zero, then it is deleted from the child-list of R. Similar checking and deletion operations are performed for its header table entry.

As described above, the Insert and Delete procedures can per- form insertion and deletion of transactions in an IWFPTWA/ IWFPTFD. Consequently, any kind of modifications can be possible in our algorithm. For example, transaction T can be modified by adding/deleting many items inside it, and let the modified transaction be Tmod. To perform this modification we need to delete T and insert Tmod in an IWFPTWA/IWFPTFD. The Modify procedure (presented at the bottom of Fig. 12) receives two transactions T1 and T2. The T1 transaction represents the transaction to be modified and the T2 transaction represents the new modified transaction. This procedure at first deletes T1 and then inserts T2 to perform this modification.

The Mining procedure creates the conditional tree for a particular pattern a in lines 1–8 as shown in Fig. 13. Subsequently, in lines 9–13, this procedure creates a candidate pattern ab for each item b appears in the header table HC of the conditional tree CT, tests candidate ab by invoking the Test_Candidate procedure, creates the prefix tree with header table for ab and recursively calls itself with pattern ab.

The pseudo-code of IWFPFD algorithm is shown in Fig. 14. In line 2, it creates the global header table H to keep the items in frequency descending order. GMAXW and LMAXW variables are declared in lines 3 and 4, respectively. In line 5, the root R of an IWFPTFD is created and initialized to NULL. Procedures of additions/deletions/modifications of transactions are invoked from lines 6 to 14. In lines 15and 16, it performs the tree-restructuring operation in order to sort the tree nodes and header table items in the frequency descending order using a path adjusting method (known as bubble sort) (Koh & Shieh, 2004) if the next N% database is scanned or database is finished. In line 18, it calculates the current GMAXW. It has the same checking for interactive mining like IWFPWA in line 20. It scans the header table from top to bottom in line 21 and adjusts the LMAXW in lines 25 and 26 as specified in Section 4.2. For the candidate testing and mining operations, it uses the same functions of Fig. 13 as shown in lines 23 and 28, respectively.

5. EXPERIMENTAL RESULTS AND ANALYSIS

Our algorithms are the first incremental and interactive weighted frequent pattern mining algorithm so far we know. But, to show the power of ''build once mine many'' property of our tree structures in interactive mining, to show the effect of one database scan, we compare our algorithms with the existing recent WFIM algorithm using both dense and sparse datasets. We also show the effectiveness of our tree structures and algorithms when a database is increasing or shrinking. We use the term IWFP to de- note our two algorithms together.

5.1. Test Environment and Datasets

To evaluate the performance of our proposed tree structures and algorithms, we have performed several experiments on both IBM synthetic datasets (T10I4D100K, T40I10D100K) and real-life datasets (chess, mushroom, pumsb, pumsb , retail, connect, kosarak) from frequent itemset mining dataset repository and UCI Machine Learning Repository These datasets do not provide the weight values of each item. As like the performance evaluation of the previous weight based frequent pattern mining,2005a, 2005b, 2006), we have generated random numbers for the weight values of each item, ranging from 0.1 to 0.9. We obtained consistent results for all those datasets. We show the experimental results on IBM synthetic T10I4D100K dataset and real-life mush- room, chess, pumsb, and kosarak datasets. Our programs were written in Microsoft Visual C++ 6.0 and run on the Windows XP operating system with a Pentium dual core 2.13 GHz CPU.

5.2. Effectiveness of IWFP in interactive weighted frequent pattern mining

Existing algorithm WFIM needs two database scans and its data structure does not have the ‘‘build once mine many’’ property. For each given minimum threshold, it has to construct its data structure and scan database twice to mine the weighted frequent pat- terns. The datasets show that our algorithms are better than WFIM even if in the worst case. Fig. 15 shows the runtime comparison in mush- room dataset for the worst case of interactive mining in our algorithm. As we have stated in interactive mining in Section 4.6, our algorithms get benefit after the first mining threshold. We have ta- ken here the result in Fig. 15 for the worst case of our tree, i.e., we have given the threshold in descending order (at first 30% then 25% and so on). After the first threshold, our trees do not have to be constructed again. As the minimum threshold decreases,a new frequent patterns and interactive weighted frequent patterns. One synthetic dataset T10I4D100K and one real-life dataset mush- room are used for this comparison. We have used five different thresholds for each dataset as shown in Table 7.

5.3. Effectiveness of IWFP in incremental weighted frequent pattern mining

We have tested the effectiveness of IWFP in incremental mining with the kosarak dataset. It has almost 1 million transactions (990,002) and 41,270 distinct items. At first we have created the IWFPT for 0.2 million transactions of this dataset and then per- formed mining operation with a minimum threshold of 5%. An- other 0.2 million transactions were added in the tree and Fig. 19. Database is increasing by db+ = 0.2 million on the kosarak dataset.

Mining operation is needed. The best case occurs if we go for mining threshold in increasing order, i.e., at first 10% then 20% and so on. Then weighted frequent patterns of threshold 20% are a subset of the weighted frequent patterns of threshold 10%, so without mining, we can find the resultant weighted frequent patterns from the previous result. In that case, after the first mining threshold, the computation times for other thresholds are almost negligible compared to the first one. Figs. 16–18 show the runtime compare performed mining operation with the same minimum threshold. In the same way, all the transactions in the kosarak dataset were added and mining operation was performed at each stage with a minimum threshold of 5%. This result is shown in Fig. 19. After It is obvious in Fig. 19 that as the database is increasing, the tree construction and mining times are increasing. After adding all the db+, we have performed the deletion operation in that tree. Here db size is 0.1 million. At first 0.1 million transactions were deleted and mining operation was performed with a minimum threshold of 5%. Same operation was repeated for another 4 times. This result is shown in Fig. 20. After deleting each db we have restructured the IWFPTFD before mining. It is also obvious in Fig. 20 that as the database is decreasing, the tree construction and mining times are decreasing. Our IWFPT have efficiently handled 41,270 distinct items in the kosarak dataset. We can observe the different number of distinct items in each size of this dataset in the horizontal axis of Fig. 19. Constructed IWFPT has 28,780 distinct items for 0.2 million transactions, 34,062 distinct items for 0.4 million transactions, 37,030 distinct items for 0.6 million transactions, 39,562 distinct items for 0.8 million transactions and 41,270 distinct items for the full kosarak dataset. Hence, we can ob- serve the scalability of IWFP in Fig. 19 by handling 41,270 distinct items and around 1 million transactions in the kosarak dataset.

5.4. Memory usage

Prefix-tree-based frequent mining research showed that the memory requirements for prefix trees are low enough to use the gigabyte-range memory available recently. We have also handled our IWFPT very efficiently within this memory range. Table 8 shows the memory usage (MB) for both of our proposed IWFPT constructed for full mushroom, chess, T10I4D100K, pumsb and kosarak datasets. There- fore, our IWFPT prefix tree structures are efficient for weighted frequent pattern mining with the recently available gigabyte-range. The importance of different websites is also different in the real-life scenarios. Mining frequent web traversal patterns can find only the patterns occurred frequently. However, if different weights are assigned to different websites according to its real- life significances or user interests, then more crucial knowledge can be discovered by weighted web traversal pattern mining. WFP mining is also useful for biological gene data analysis, as different types of genes have different significance for a particular drug analysis. Global positioning system (GPS) of Telematics can be found another important application area of weighted frequent pattern mining. One possible application is the determination of a traffic pattern (a set of links) that considers speed and traffic volume using the weight and frequency information of each link (Yun, 2007a; Yun & Leggett, 2006). Candidate weighted frequent patterns (the combination of links) can be calculated according to a user's request to find a path between two locations.

The above discussion shows that WFP mining can mine more practical knowledge than the traditional frequent patterns. How- ever, as discussed in Section 1, the real-world databases are dynamic in nature, i.e, new transactions can be inserted and old transactions may be deleted or modified frequently. Moreover, interactive mining is essentially needed so that users can change their minimum thresholds dynamically according to their application interests. Hence, it would be more realistic to consider incremental and interactive WFP mining rather than only WFP mining in static databases.

7. CONCLUSIONS

In this paper, we propose two novel tree structures and two new single-pass weighted frequent pattern mining algorithms based on those tree structures for incremental databases. To the best of our knowledge, our approaches are the first effort to efficiently mine weighted frequent patterns with a single database scan and on incremental databases. Our tree structures have the ‘‘build once mine many’’ property and highly suitable for interactive mining. Between our two tree structures, because of relatively simpler construction process, IWFPTWA provides easier maintenance phase during incremental mining. On the other hand, a rather com- pact structure of IWFPTFD requires tree restructuring operation be- fore mining. However, we have shown that the tree restructuring cost of IWFPTFD is insignificant compared to the gain achieved in mining phase from the tree compactness that reduces the overall runtime.

REFERENCES

[1] Agrawal, R., Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In Proceedings of the 20th international conference on very large data bases, September (pp. 487–499).

[2] Agrawal, R., Imielin´ ski, T., Swami, A. (1993). Mining association rules between sets of items in large databases. In Proceedings of the 12th ACM SIGMOD international conference on management of data, May (pp. 207–216).

[3] Ahmed, C. F., Tanbeer, S. K., Jeong, B.-S., Lee, Y.-K. (2008). Mining weighted frequent patterns in incremental databases. In: Proceeding of the 10th Pacific Rim international conference on artificial intelligence (pp. 933–938). Hanoi, Vietnam, December 15–19.

[4] Cai, C. H., Fu, A. W., Cheng, C. H., Kwong, W. W. (1998). Mining association rules with weighted items. In Proceedings of international database engineering and applications symposium, IDEAS 98 (pp. 68–77). Cardiff, Wales, UK.

[5] Chang, J. H., & Lee, W. S. (2005). Estwin: Online data stream mining of recent frequent itemsets by sliding window method. Journal of information science,31(2), 76–90.

[6] Chang, L., Wang, T., Yang, D., Luan, H., & Tang, S. (2009). Efficient algorithms for incremental maintenance of closed sequential patterns in large databases. Data and Knowledge Engineering, 68, 68–106.

[7] Cheung, W., & Zaïane, O. R. (2003). Incremental mining of frequent patterns without candidate generation or support constraint. In Proceedings of the Seventh International Database Engineering and Applications, Symposium (IDEAS'03) (pp.111–116).

[8] Dong, J., & Han, M. (2007). BitTableFI: An efficient mining frequent itemsets algorithm. Knowledge-Based Systems, 20, 329–335.

[9] Grahne, G., & Zhu, J. (2005). Fast algorithms for frequent itemset mining using FP- Trees. IEEE Transactions on Knowledge and Data.