Expert Systems with Applications 39 (2012) 7976–7994

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Single-pass incremental and interactive mining for weighted frequent patterns

Chowdhury Farhan Ahmed[a], Syed Khairuzzaman Tanbeer[a], Byeong-Soo Jeong[a,*], Young-Koo Lee[a], Ho-Jin Choi[b]

[a] Department of Computer Engineering, Kyung Hee University, 1 Seochun-dong, Kihung-gu, Youngin-si, Kyunggi-do, 446-701, Republic of Korea
[b] Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 335 Gwahak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea

ARTICLE INFO

Keywords:
Data mining
Knowledge discovery
Weighted frequent pattern mining
Incremental mining
Interactive mining

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 IWFPT$_{WA}$ (Incremental WFP tree based on weight ascending order) and IWFPT$_{FD}$ (Incremental WFP tree based on frequency descending order), and two new algorithms IWFP$_{WA}$ and IWFP$_{FD}$ for incremental 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. IWFP$_{WA}$ gets advantage in candidate pattern generation by obtaining the highest weighted item in the bottom of IWFPT$_{WA}$. IWFP$_{FD}$ ensures that any non-candidate item cannot appear before candidate items in any branch of IWFPT$_{FD}$ and thus speeds up the prefix tree and conditional tree creation time during mining operation. IWFPT$_{FD}$ also achieves the highly compact incremental tree to save memory space. To our knowledge, this is the first research work to perform single-pass incremental and interactive mining for weighted frequent patterns. Extensive performance analyses show that our tree structures and algorithms are very efficient and scalable for single-pass incremental and interactive WFP mining.

1. 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 contribute a large portion of overall revenue even though it does not appear in a large number of transactions. For this reason, weighted pattern mining (Ahmed, Tanbeer, Jeong, & Lee, 2008; Cai, Fu, Cheng, & Kwong, 1998; Tao, 2003; Wang, Yang, & Yu, 2004; Yun, 2007a, 2007b, 2007c; Yun, 2008a, 2008b; Yun & Leggett, 2005a, 2005b; Yun & Leggett, 2006) 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 are important factors, web traversal pattern mining where each web page has different strength of impact, and biomedical data analysis where most diseases are not caused by a single gene but by a combination of genes.

Along with considering the different weights of items in mining meaningful patterns, developing incremental and interactive mining algorithm is also quite important as a database grows very rapidly and changes frequently in real world applications. When a database is changed by inserting and deleting several transactions, it is very time consuming to re-process the whole mining operations for getting new mining results. Furthermore, finding an appropriate minimum support threshold may require repeated mining processes to adjust threshold value. Therefore, it is important to avoid unnecessary computations by utilizing the previous data structures or mining results. Some research works (Chang, Wang, Yang, Luan, & Tang, 2009; Cheung & Zaïane, 2003; Koh & Shieh, 2004; Lee & Yen, 2008; Leung, Khan, Li, & Hoque, 2007; Li, Deng, & Tang, 2006; Lin, Hong, & Lu, 2009; Tanbeer, Ahmed, Jeong, & Lee, 2009; Zhang, Zhang, & Zhang, 2007) have been done to handle such incremental databases. They have shown that incremental prefix tree can be an effective data structure for maintaining whole information of a changing transaction database.

* Corresponding author. Tel.: +82 31 201 2932; fax: +82 31 202 1723.
E-mail addresses: jeong@khu.ac.kr, jeong_khu@yahoo.com (B.-S. Jeong).

0957-4174/$ - see front matter Crown Copyright © 2012 Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.01.117

There have been a lot of research works dealing with these issues in the area of data mining (Cai et al., 1998; Cheung & Zaïane, 2003; Koh & Shieh, 2004; Leung et al., 2007; Li et al., 2006; Tanbeer et al., 2009; Tao, 2003; Wang et al., 2004; Yun, 2007a, 2007b, 2007c; Yun, 2008a; Yun & Leggett, 2005a, 2005b, 2006; Zhang et al., 2007). However, to the best of our knowledge there was no attempt to solve these problems together when newly developing a pattern mining algorithm. Existing weighted frequent pattern mining algorithms (Cai et al., 1998; Tao, 2003; Wang et al., 2004; Yun, 2007a, 2007c; Yun & Leggett, 2005a, 2005b, 2006) assumed that databases were static and did not consider the scenarios where one or more transactions could be deleted and inserted in the database. Moreover, the data structures presented in the previous works do not have the "*build once mine many*" property which is very necessary for interactive mining. Hence, the data structure was designed for a particular minimum threshold. If the minimum threshold is 30%, for example, then they can calculate the result for this specified threshold only. If the users again want to know the result for minimum threshold 20%, then they have to build their data structures again and start their calculations from the very beginning. In our real world scenarios, users need to extract knowledge using different minimum thresholds according to their application interests from the database. Accordingly, the "*build once mine many*" property of a data structure is necessary to discover this type of knowledge without building the data structure again and also within a real time. On the other hand, most incremental and interactive mining approaches were developed for finding frequent patterns without considering the different weights of items. Furthermore, in recent years many applications (i.e., sensor networks, network monitoring, stock trading, etc.) produce data in the shape of data streams (Chang & Lee, 2005; Jiang & Gruenwald, 2006; Metwally, Agrawal, & Abbadi, 2006; Raissi, Poncelet, & Teisseire, 2007; Yu, Chong, Lu, Zhang, & Zhou, 2006). 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 pattern mining approach which can solve these problems at one time. In this paper, we propose two novel tree structures IWFPT_{WA} (Incremental weighted frequent pattern tree based on weight ascending order) and IWFPT_{FD} (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 IWFP_{WA} and IWFP_{FD}. They exploit a pattern growth mining approach. Our proposed first tree structure IWFPT_{WA} arranges the items in weight ascending order and can be constructed without any restructuring operation. IWFP_{WA} gets advantage in candidate pattern generation by obtaining the highest weighted item in the bottom of the IWFPT_{WA}. 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. This situation increases the prefix tree and conditional tree creation time during the mining operation. Our proposed second tree structure IWFPT_{FD} arranges the items in frequency descending order and ensures non-candidate items cannot appear before candidate items in any branch of a tree. By using IWFPT_{FD}, our proposed second algorithm IWFP_{FD} speeds up the prefix tree and conditional tree creation time during the mining operation and achieves overall runtime gain over IWFP_{WA}. Moreover, by sorting the items in frequency descending order, IWFPT_{FD} also achieves a highly compact incremental tree structure to save memory space.

In summary, the main contributions of this paper are: (1) Devising two novel tree structures that are very efficient for finding weighted frequent patterns, (2) development of two new single-scan mining algorithms based on the above tree structures, which can be applied for finding weighted frequent patterns over a data stream, (3) description of how to apply our algorithms for incremental and interactive mining, and (4) extensive performance study to compare the performance of our algorithms with the existing recent WFIM algorithm and show the effectiveness of one database scan and incremental database.

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 (Agrawal & Srikant, 1994; Agrawal, Imieliński, & Swami, 1993) is used to prune the infrequent patterns. This property tells that if a pattern is infrequent then all of its super patterns must be infrequent. *Apriori* (Agrawal & Srikant, 1994; Agrawal et al., 1993) algorithm is the initial solution of frequent pattern mining problem and very useful in association rule mining (Agrawal & Srikant, 1994; Agrawal et al., 1993; Jiang & Gruenwald, 2006; Koh & Shieh, 2004; Li et al., 2006; Lim & Lee, 2010; Verma & Vyas, 2005). 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 (Chang & Lee, 2005; Cheung & Zaïane, 2003; Dong & Han, 2007; Grahne & Zhu, 2005; Han, Cheng, Xin, & Yan, 2007; Hu, Sung, Xiong, & Fu, 2008; Jiang & Gruenwald, 2006; Koh & Shieh, 2004; Lee, Tsao, Chen, Lin, & Yang, 2010; Leung et al., 2007; Li et al., 2006; Metwally et al., 2006; Pei & Han, 2000; Raissi et al., 2007; Song, Yang, & Xu, 2008; Tanbeer et al., 2009; Verma & Vyas, 2005; Wang, Han, & Pei, 2003; Ye, Wang, & Shao, 2005; Yu et al., 2006; Zhang et al., 2007) 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 structure or previous mining results. Some research works (Cheung & Zaïane, 2003; Koh & Shieh, 2004; Leung et al., 2007; Li et al., 2006; Lin et al., 2009; Tanbeer et al., 2009; Zhang et al., 2007) have developed single-pass incremental and interactive mining algorithms based on traditional frequent pattern mining. AFPIM (Koh & Shieh, 2004) readjusts an FP-tree structure using path adjusting (bubble sort) mechanism in order to improve incremental mining performance. CanTree (Leung et al., 2007) captures the transactions in canonical order and maintains the whole database information in a tree while the database is growing or shrinking. It has the "*build once mine many*" property. CP-tree (Tanbeer et al., 2009) improves CanTree by restructuring the incremental prefix-tree according to the frequency descending order and thus, gets remarkable mining time improvement compared to CanTree. 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 (Chang & Lee, 2005; Jiang & Gruenwald, 2006; Li, 2009; Metwally et al., 2006; Raissi et al., 2007; Yu et al., 2006) 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. IncWTP and WssWTP algorithms (Lee & Yen, 2008) are designed for incremental and interactive mining of web traversal patterns.

2.1.3. Weighted frequent pattern mining

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \ldots, T_n\}$ where each transaction $T_i \in D$ is a subset of I. A pattern or itemset is defined by the set $X = \{x_1, x_2, \ldots, x_k\}$, where $X \subseteq I$ and $k \in [1, m]$. However, an itemset is called k-itemset 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, $P\{x_1, x_2, \ldots, x_k\}$ is given as follows:

$$\text{Weight}(P) = \frac{\sum_{q=1}^{\text{length}(P)} \text{Weight}(x_q)}{\text{length}(P)} \qquad (1)$$

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:

$$W\text{support}(P) = \text{Weight}(P) \times \text{Support}(P) \qquad (2)$$

For example, Wsupport(ad)=0.475 × 4 = 1.9 in Fig. 1.

TID	Transactions
T_1	a,b,c,d,g,h
T_2	a,e,f
T_3	b,e,f,g,h
T_4	a,b,c,d
T_5	a,b,d,h
T_6	a,b,d,e

(a) Transaction database

Items	W
a	0.6
b	0.5
c	0.2
d	0.35
e	0.5
f	0.3
g	0.4
h	0.38

(b) Weight Table

Fig. 1. An example of retail database.

Table 1
An example transaction database and weight table.

Bar code	Item	Price ($)	Support (frequency)	Normalized Weight
1	Personal computer	800	500	0.8
2	Laser printer	450	320	0.45
3	Bubble jet printer	250	450	0.25
4	Digital camera	600	700	0.6
5	Memory stick	200	825	0.2
6	Hard disk	130	350	0.13
7	DVD drive	100	450	0.1
8	CD drive	50	250	0.05

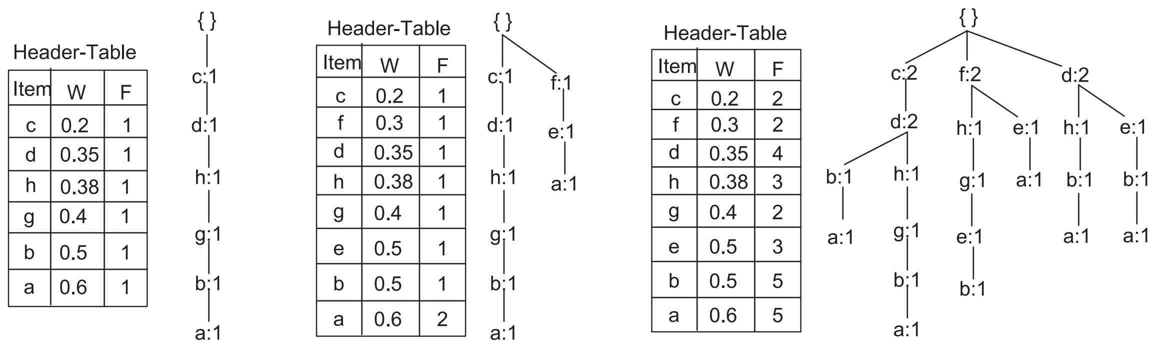
A pattern is called a weighted frequent pattern if the weighted support of the pattern is greater than or equal to the minimum threshold. Consider the minimum threshold is 1.5 in Fig. 1 and then *ad* is a weighted frequent pattern.

Table 1 shows an example of a retail database in which the normalized weight values are assigned to items based on the price of each item. Normalization process is needed for adjusting the differences 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 (Cai et al., 1998), WARM (Tao, 2003), WAR (Wang et al., 2004) have been developed based on the *Apriori* algorithm using the level-wise candidate generation-and-test paradigm. Obviously, these algorithms require multiple database scans and result in poor mining performance. WFIM (Yun & Leggett, 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 *downward closure* property on that tree. WLPMINER (Yun, 2008b; Yun & Leggett, 2005b) algorithm finds weighted frequent patterns using length decreasing support constraints. WCloset (Yun, 2007c) is proposed to calculate the closed weighted frequent patterns. 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 hyperclique-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 & Leggett, 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



Header-Table

Item	W	F
c	0.2	1
d	0.35	1
h	0.38	1
g	0.4	1
b	0.5	1
a	0.6	1

(a) After Inserting T_1

Header-Table

Item	W	F
c	0.2	1
f	0.3	1
d	0.35	1
h	0.38	1
g	0.4	1
e	0.5	1
b	0.5	1
a	0.6	2

(b) After inserting T_2

Header-Table

Item	W	F
c	0.2	2
f	0.3	2
d	0.35	4
h	0.38	3
g	0.4	2
e	0.5	3
b	0.5	5
a	0.6	5

(c) After inserting transactions up to T_6

Fig. 2. $IWFPT_{WA}$ construction.

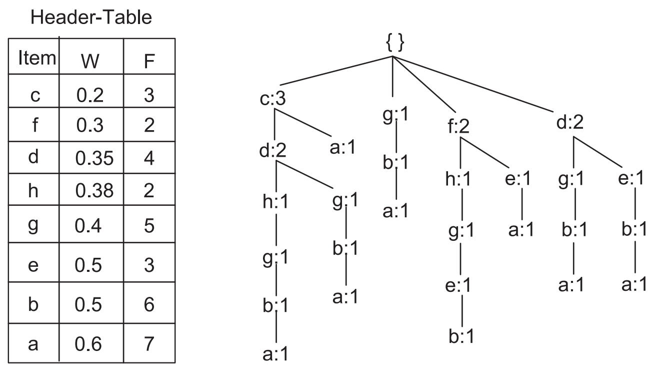
	TID	Transactions
Original DB	T_1	a,b,c,d,g,h
	T_2	a,e,f
	T_3	b,e,f,g,h
	T_4	a,b,c,d
	T_5	a,b,d,h
	T_6	a,b,d,e
db_1^+	T_7	a,b,c
	T_8	a,b,c,d,g
db_2^+	T_9	a,b,g
	T_{10}	a,c

(a) Incremented database

TID	Transactions
T_1	a,b,c,d,g,h
T_2	a,e,f
T_3	b,e,f,g,h
T_5	a,b,d,g
T_6	a,b,d,e
T_8	a,b,c,d,g
T_9	a,b,g
T_{10}	a,c

(b) Updated database

Fig. 3. Incremented and updated database.



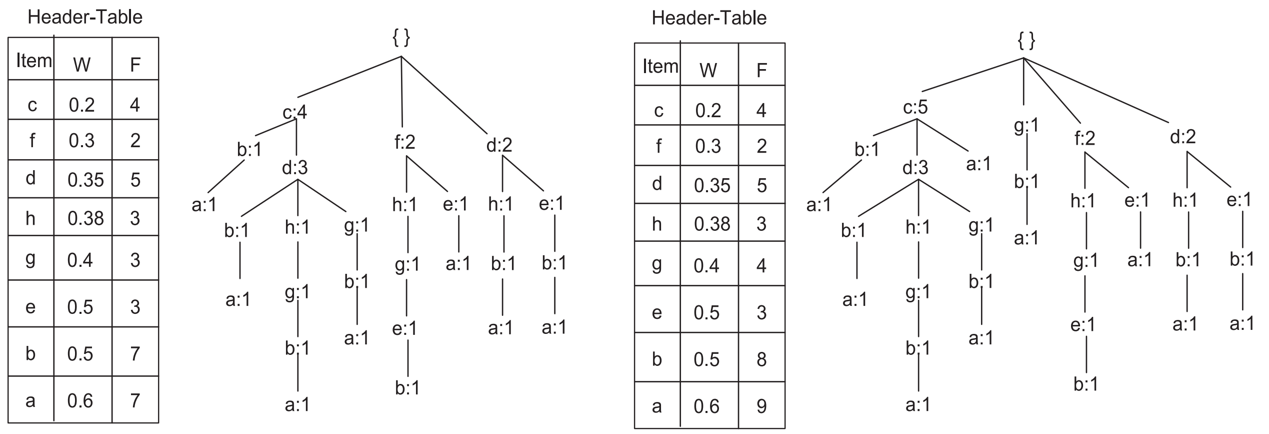
Header-Table

Item	W	F
c	0.2	3
f	0.3	2
d	0.35	4
h	0.38	2
g	0.4	5
e	0.5	3
b	0.5	6
a	0.6	7

Fig. 5. Update operations in $IWFPT_{WA}$.

and “*d*” is 0.35 × 4 = 1.4. If the minimum threshold is 1.5, then pattern “*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's 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.

To our knowledge, none of existing weighted frequent pattern mining methods proposed any solution for incremental mining, where lots of transactions can be added and deleted. Moreover, they need at least two database scans, not suitable for stream data mining and do not have the “*build once mine many*” property for interactive mining. For this reason, we propose novel tree structures and algorithms for single-pass incremental and interactive mining of



Header-Table

Item	W	F
c	0.2	4
f	0.3	2
d	0.35	5
h	0.38	3
g	0.4	3
e	0.5	3
b	0.5	7
a	0.6	7

(a) After Inserting db_1^+

Header-Table

Item	W	F
c	0.2	4
f	0.3	2
d	0.35	5
h	0.38	3
g	0.4	4
e	0.5	3
b	0.5	8
a	0.6	9

(b) After Inserting db_2^+

Fig. 4. Incremental operations in $IWFPT_{WA}$.

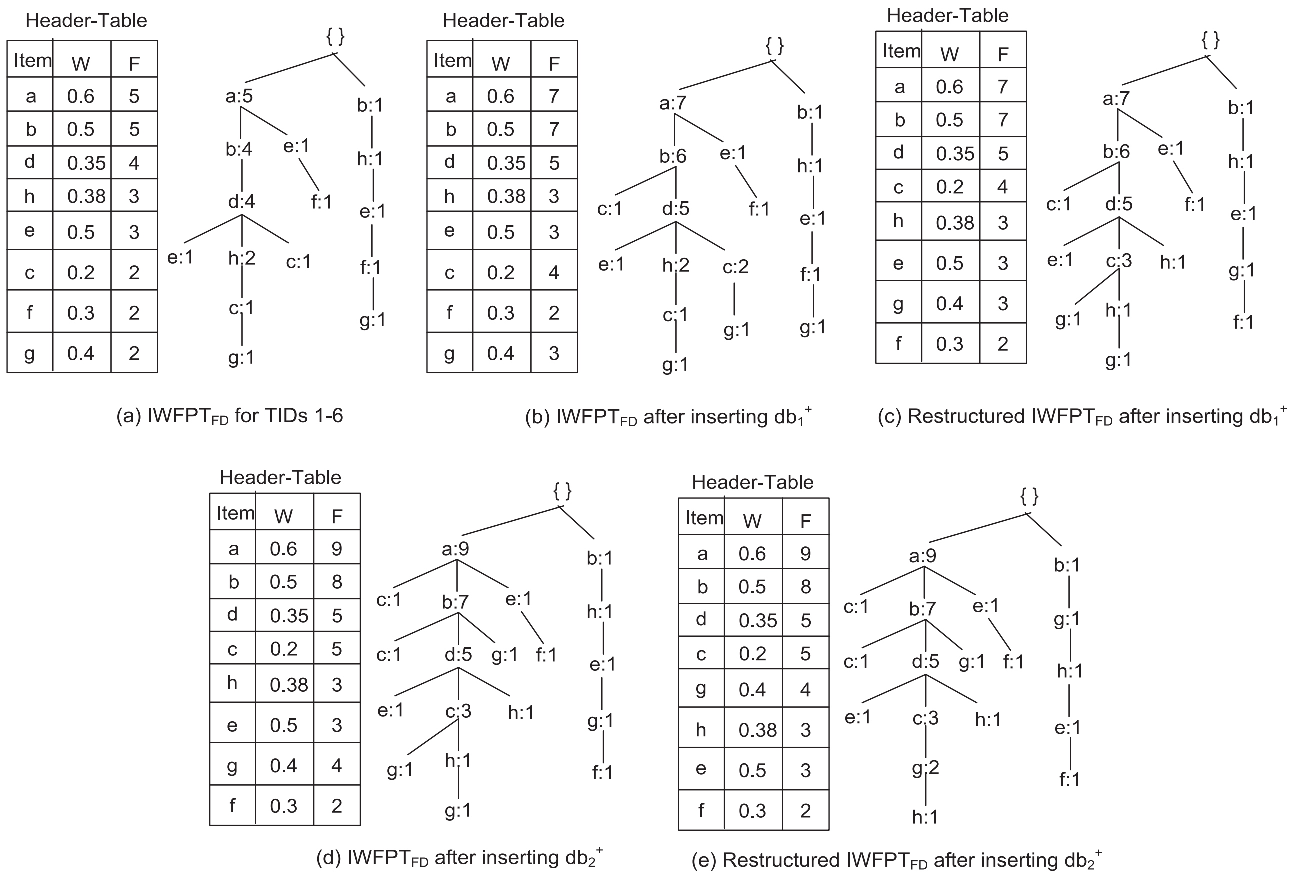

(a) IWFPT$_{FD}$ for TIDs 1-6
(b) IWFPT$_{FD}$ after inserting db$_1^+$
(c) Restructured IWFPT$_{FD}$ after inserting db$_1^+$
(d) IWFPT$_{FD}$ after inserting db$_2^+$
(e) Restructured IWFPT$_{FD}$ after inserting db$_2^+$

Fig. 6. Construction of IWFPT$_{FD}$ and performing inserting and restructuring operations.

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, elaborate 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, 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 IWFPT$_{WA}$ 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 IWFPT_{FD}, which arranges the incremental prefix tree in frequency descending order by restructuring operation to get the advantages of FP-tree.

3.2. IWFPT_{WA}: our first proposal

First of all, we describe the construction process of IWFPT_{WA} (incremental weighted frequent pattern tree based on weight ascending order), then we show how it can handle insert and update 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 T_1 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 IWFPT_{WA} after inserting T_1. Fig. 2b shows IWFPT_{WA} after inserting T_2. In the same way, all the transactions up to T_6 are inserted into the tree. Fig. 2c shows the final IWFPT_{WA} after inserting T_6.

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 "db_{mod}" 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 db_1^{+} and db_2^{+} as shown in Fig. 3a. Fig. 3b shows the database is updated by deleting T_4 and T_7, and by modifying T_5 (item "*h*" is replaced by "*g*"). Figs. 4a and b show that IWFPT_{WA} can easily be incremented by inserting db_1^{+} and db_2^{+}, respectively. The insertion process is same as transactions are inserted during the construction process.

Now we delete db^{-} (T_4 and T_7) from the current IWFPT_{WA} (shown in Fig. 4b). T_4 is present in the tree as the path "*c d b a*". To remove this transaction we have to reduce the frequency value of all the nodes in that path by one. After reducing, the frequency values of nodes "*a*" and "*b*" become zero at that path. Therefore, we have to delete these two nodes (shown in Fig. 5). Fig. 5 also shows that T_7 has been removed in the same way and T_5 is modified with replacing item "*h*" with item "*g*" in the path "*d h b a*".

Property 1.. *The ordering of items in IWFPT_{WA} is not affected in spite of changing the frequency of the items by additions, deletions and modifications.*

Proof. Let X and Y be two items with weight W_x and W_y respectively. Consider $W_x < W_y$ and their frequency values are F_x and F_y, respectively. When database changes due to incremental updating, the values of F_x and F_y may vary based on the occurrences of the items in the incremented database. But it cannot create any effect on W_x and W_y. Since the items in the tree are ordered in their weight ascending order, X always comes before Y in any branch of the tree. □

3.3. IWFPT_{FD}: our second proposal

Here we first describe the construction process of IWFPT_{FD} (incremental weighted frequent pattern tree based on frequency descending order). Afterwards, we show how it can handle insertions, deletions and modifications.

Consider the example database in Fig. 1a. The initial IWFPT_{FD} can be constructed in the same way of IWFPT_{WA} (shown in Fig. 2). After adding all the six transactions of Fig. 1a, the IWFPT_{WA} (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 between them is performed. After each swapping operation, nodes having same items (if any due to the swapping) are merged together. 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. Therefore, we also perform this tree restructuring operation in a way of slot-by-slot fashion. It is remarkable that IWFPT_{FD} (Fig. 6a) has only 15 nodes (without root) compared to 22 nodes in the IWFPT_{WA} (shown in Fig. 2c).

The new frequency descending sort order in the header table is $\langle a, b, d, h, e, c, f, g\rangle$ as shown in Fig. 6a. Transactions in db_1^{+} are inserted into IWFPT_{FD} according to that order. After inserting T_7 and T_8 in IWFPT_{FD} (Fig. 6a) the resultant tree is shown in Fig. 6b. It is obvious from the header table that items are not in the frequency descending order now. So we need to sort the header table and tree according to the frequency descending order. The sorted tree is shown in Fig. 6c. It has only 17 nodes compared to 27 nodes of IWFPT_{WA} (shown in Fig. 4a). The new sort order is now $\langle a, b, d, c, h, e, g, f\rangle$. Transactions in db_2^{+} are inserted into IWFPT_{FD} according to that order. After inserting T_9 and T_{10} in IWFPT_{FD} (Fig. 6c) the resultant tree is shown in Fig. 6d. The sorted tree is shown in Fig. 6e. It has only 18 nodes compared to 31 nodes of IWFPT_{WA} (shown in Fig. 4b). After deleting T_4, T_7, and modifying T_5 from IWFPT_{FD} using the same way described in Section 3.2 the resultant tree is shown in Fig. 7a. After sorting the resultant tree is shown in Fig. 7b. It has only 16 nodes compared to 27 nodes of IWFPT_{WA} (shown in Fig. 5).

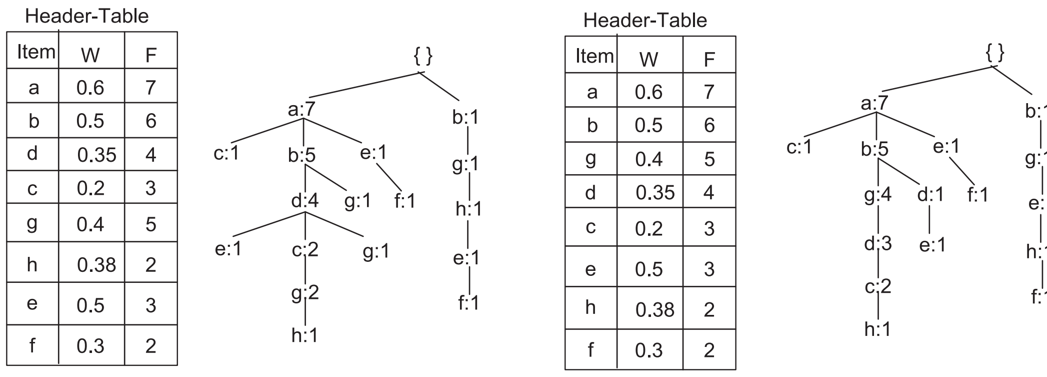
The following properties are true for both IWFPT_{WA} and IWFPT_{FD}.

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 P_1 of the tree and frequency value of X and Y are F_x and F_y respectively. The frequency value of Y is increased when a transaction contains items $\alpha XY\beta$. Here α and β are the set of other items before X and after Y respectively in P_1 (β can be *NULL*). Hence, F_x is incremented before F_y, and F_y cannot be greater than F_x. On the other hand, if at least one transaction contains αX, then only F_x is incremented and F_x becomes larger than F_y. Therefore, $F_x \geqslant F_y$. □

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. □



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

Fig. 7. Update operations in IWFPT$_{FD}$.

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 adding db_3^+ which contains a new item "*x*" with a different weight of 0.45. Figs. 8a and b show the modified database and weight table, respectively. Fig. 8c shows the resultant IWFPT$_{WA}$ with header table after inserting db_3^+. The new item "*x*" is inserted into the header table according to its weight value. However, the previous ordering of items is not affected with the insertion of this new value, i.e., if the previous order of two items i_j and i_k is $i_j < i_k$, then it still holds. Therefore, Property 1 holds for IWFPT$_{WA}$ 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 IWFPT$_{WA}$. Fig. 8c shows that the first transaction of db_3^+, "*a b x*", is arranged as "*x b a*" in weight ascending order and inserted as a new branch in IWFPT$_{WA}$. 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 IWFPT$_{WA}$ in this situation.

Fig. 8d shows the resultant IWFPT$_{FD}$ after inserting db_3^+ and Fig. 8e shows the IWFPT$_{FD}$ after restructuring operation. These figures explain that although a new item with different weight is inserted into an IWFPT$_{FD}$, 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 IWFPT$_{FD}$. In a similar way of IWFPT$_{WA}$, it can be shown from these figures that Property 2 also holds for IWFPT$_{FD}$ in this situation. Furthermore, IWFPT$_{FD}$ can still achieve a significant compression gain over IWFPT$_{WA}$. For the current database of Fig. 8a, IWFPT$_{FD}$ has only 19 nodes compared to 31 nodes of IWFPT$_{WA}$.

4. Our proposed algorithms

In this section, at first we describe the mining process of our proposed IWFP$_{WA}$ and IWFP$_{FD}$ algorithms using our proposed IWFPT$_{WA}$ and IWFPT$_{FD}$ 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.

4.1. Mining process in IWFP$_{WA}$

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 conditional tree is created from the prefix tree by deleting the nodes containing infrequent items. IWFP$_{WA}$ 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 itemset 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 maximum weight, denoted by *LMAXW,* is needed when we are doing the mining operation for a particular item. As IWFPT$_{WA}$ 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 maximum 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, IWFPT$_{WA}$ 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 ⟨*c*:1.8, *f*:1.8, *d*:2.4, *h*:1.2, *g*:3.0, *x*:1.2, *e*:1.8, *b*:4.2, *a*:4.8⟩. 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 ⟨*d*:2.4, *g*:2.4, *b*:3.6⟩. 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.

	TID	Transactions
	T_1	a,b,c,d,g,h
	T_2	a,e,f
	T_3	b,e,f,g,h
	T_5	a,b,d,g
	T_6	a,b,d,e
	T_8	a,b,c,d,g
	T_9	a,b,g
	T_{10}	a,c
db_3^+	T_{11}	a,b,x
	T_{12}	h,x

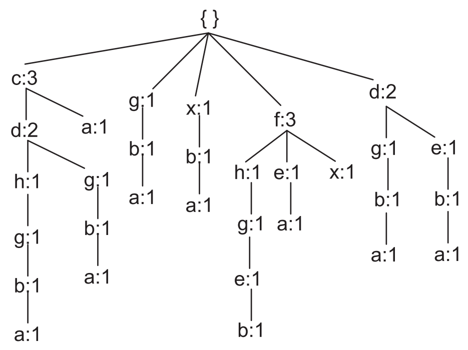
(a) Incremented database containing a new item with a different weight

Items	W
a	0.6
b	0.5
c	0.2
d	0.35
e	0.5
f	0.3
g	0.4
h	0.38
x	0.45

(b) Weight Table

Header-Table

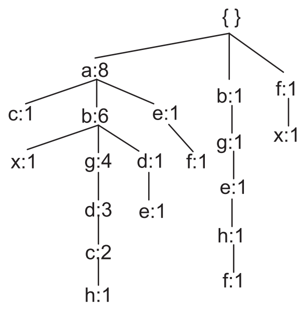
Item	W	F
c	0.2	3
f	0.3	3
d	0.35	4
h	0.38	2
g	0.4	5
x	0.45	2
e	0.5	3
b	0.5	7
a	0.6	8



(c) $IWFPT_{WA}$ after inserting db_3^+

Header-Table

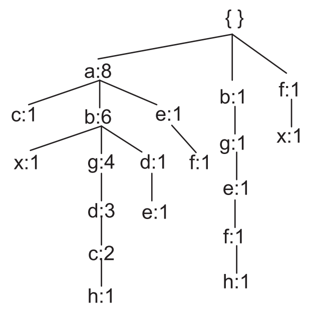
Item	W	F
a	0.6	8
b	0.5	7
g	0.4	5
d	0.35	4
c	0.2	3
e	0.5	3
h	0.38	2
f	0.3	3
x	0.45	2



(d) $IWFPT_{FD}$ after inserting db_3^+

Header-Table

Item	W	F
a	0.6	8
b	0.5	7
g	0.4	5
d	0.35	4
c	0.2	3
e	0.5	3
f	0.3	3
h	0.38	2
x	0.45	2



(e) Restructured $IWFPT_{FD}$ after inserting db_3^+

Fig. 8. Handling insertion of a new item with a different weight.

The prefix tree of pattern "*ab*" is created in Fig. 9c. Two items "*d*" and "*g*" are candidates with pattern "*ab*". As a result, this is also the conditional tree for the pattern "*ab*". Candidate patterns "*abd*" and "*abg*" are generated. The prefix trees of patterns "*abg*" and "*ag*" are shown in Fig. 9d and e, respectively. Now prefix tree for the item "*b*" is created in Fig. 9f. For item "*b*", the *LMAXW* = 0.5 as the item "*a*" will not come out here. Observe that the global weighted infrequent items "*c*", "*f*", "*h*", "*e*", and "*x*" are present in this tree. We have to delete them. The weighted frequency list is ⟨*d*:2.0, *g*:2.5⟩. The key point is that, the weighted frequency of "*d*" is $4 \times 0.5 = 2.0$, as *LMAXW* reduces from 0.6 to 0.5. Now without further calculation we can prune "*d*". But if *LMAXW* is 0.6 at this point, then the weighted frequency of "*d*" is $4 \times 0.6 = 2.4$ and hence it becomes a candidate. This is the main advantage of $IWFP_{WA}$.

The conditional tree of item "*b*" contains only one item "*g*" (shown in Fig. 9g) and the candidate pattern "*bg*" is generated. The prefix tree of the item "*g*" is shown in Fig. 9h and *LMAXW* = 0.4 for this tree. Global infrequent items "*c*", "*f*", and "*h*" are present but we do not need any calculation for them. The weighted frequency list is ⟨*d*:1.2⟩. As a consequence, we do not have to create any conditional tree for the item "*g*". We have to test all the

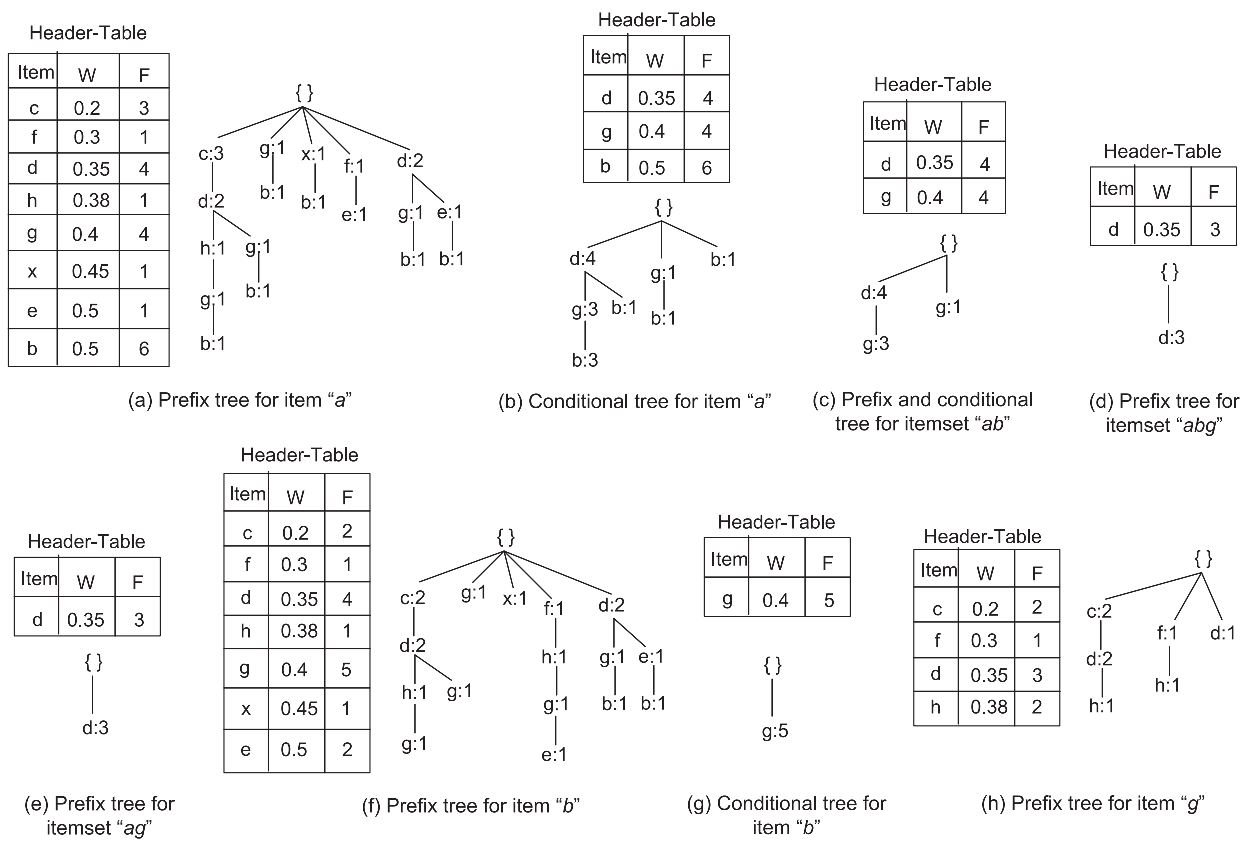

Header-Table
Item W F
c 0.2 3
f 0.3 1
d 0.35 4
h 0.38 1
g 0.4 4
x 0.45 1
e 0.5 1
b 0.5 6
{ }
c:3 d:2 h:1 g:1 g:1 b:1 b:1 b:1 x:1 b:1 f:1 e:1 d:2 g:1 e:1 b:1 b:1
(a) Prefix tree for item "a"
Header-Table
Item W F
d 0.35 4
g 0.4 4
b 0.5 6
d:4 g:3 b:3 b:1 g:1 b:1 b:1
(b) Conditional tree for item "a"
Header-Table
Item W F
d 0.35 4
g 0.4 4
d:4 g:3 g:1
(c) Prefix and conditional tree for itemset "ab"
Header-Table
Item W F
d 0.35 3
d:3
(d) Prefix tree for itemset "abg"
Header-Table
Item W F
d 0.35 3
d:3
(e) Prefix tree for itemset "ag"
Header-Table
Item W F
c 0.2 2
f 0.3 1
d 0.35 4
h 0.38 1
g 0.4 5
x 0.45 1
e 0.5 2
c:2 d:2 h:1 g:1 g:1 g:1 x:1 f:1 h:1 g:1 e:1 d:2 g:1 e:1 b:1 b:1
(f) Prefix tree for item "b"
Header-Table
Item W F
g 0.4 5
g:5
(g) Conditional tree for item "b"
Header-Table
Item W F
c 0.2 2
f 0.3 1
d 0.35 3
h 0.38 2
c:2 d:2 h:1 f:1 h:1 d:1
(h) Prefix tree for item "g"

Fig. 9. Mining process in IWFP$_{WA}$.

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 are ⟨*a*:4.8, *b*:3.5, *ab*:3.3, *bg*:2.25⟩.

4.2. Mining process in IWFP$_{FD}$

To reduce the huge number of nodes in the prefix and conditional trees in IWFP$_{WA}$, IWFP$_{FD}$ is designed based on IWFPT$_{FD}$. IWFP$_{FD}$ 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 current *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 IWFPT$_{FD}$ 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 pattern "*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 patterns "*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 patterns. One extra pattern "*bd*" is generated here which is not generated by IWFP$_{WA}$. Observe that no global weighted infrequent items like "*c*", "*f*", "*h*", "*e*", "*x*" appear in any prefix and conditional trees during mining in IWFP$_{FD}$.

4.3. Analysis of mining performance

FP-growth (Han et al., 2004) showed that we can get a huge prefix-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 between the frequent nodes. As IWFPT$_{WA}$ arranges items in weight ascending order, it has a poor prefix-sharing and that takes more memory. Moreover, non-candidate nodes frequently occur between weighted frequent candidate nodes. Therefore, there are many global weighted infrequent nodes in the prefix trees of the candidate items in IWFP$_{WA}$. 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 increase 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 IWFP$_{WA}$ 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

Table 2
Weighted frequent patterns calculations from candidate set in $IWFP_{WA}$.

No.	Candidate patterns	Weight calculation	Weighted support calculation	Result
1	ad:4	(0.6 + 0.35)/2 = 0.475	0.475 × 4 = 1.9	Pruned
2	ag:4	(0.6 + 0.4)/2 = 0.5	0.5 × 4 = 2.0	Pruned
3	ab:6	(0.6 + 0.5)/2 = 0.55	0.55 × 6 = 3.3	Passed
4	a:8	0.6	0.6 × 8 = 4.8	Passed
5	abd:4	(0.6 + 0.5 + 0.35)/3 = 0.483	0.483 × 4 = 1.93	Pruned
6	abg:4	(0.6 + 0.5 + 0.4)/3 = 0.5	0.5 × 4 = 2.0	Pruned
7	bg:5	(0.5 + 0.4)/2 = 0.45	0.45 × 5 = 2.25	Passed
8	b:7	0.5	0.5 × 7 = 3.5	Passed
9	g:5	0.4	0.4 × 5 = 2.0	Pruned
10	d:4	0.35	0.35 × 4 = 1.4	Pruned

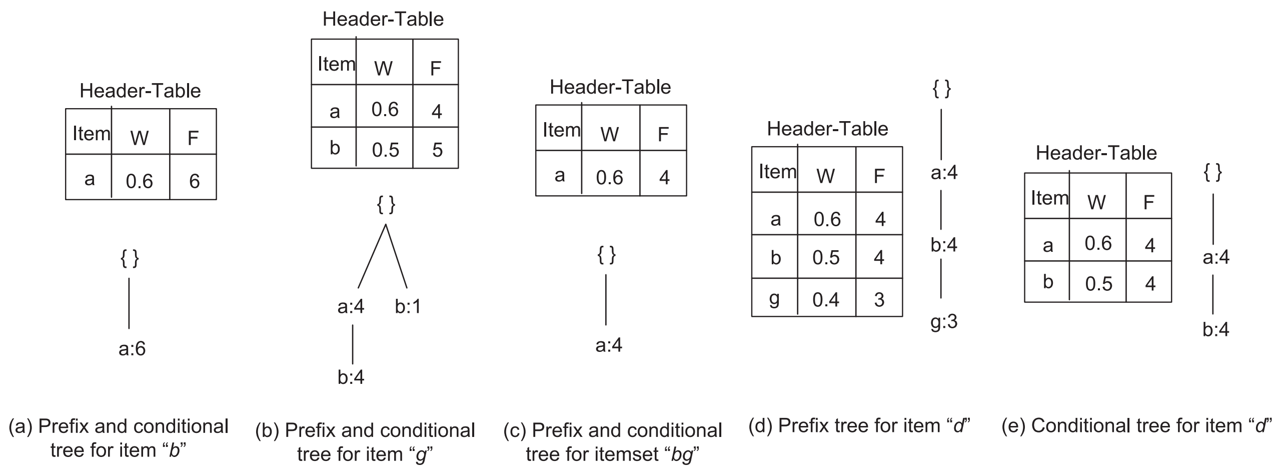


Fig. 10. Mining process in $IWFP_{FD}$.

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 $IWFPT_{WA}$ (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, $IWFPT_{FD}$ is designed in frequency descending order. As the candidates of weighted frequent patterns are taken by multiplying *GMAXW*, the frequency descending order is also the maximum weighted frequency descending order. Hence, by arranging the items in this order $IWFPT_{FD}$ keeps all the non-candidate items after the candidate items in each branch, i.e., no non-candidate item can appear in $IWFPT_{FD}$ between the candidate items. Moreover, 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 prefix trees as shown in the mining process of $IWFP_{FD}$ (in Fig. 10).

In spite of these drawbacks, $IWFP_{WA}$ has two advantages over $IWFP_{FD}$. The first one is, in $IWFP_{WA}$, when we go for mining operation 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 $IWFP_{FD}$ but not in $IWFP_{WA}$ for minimum threshold 2.2. Table 4 shows the performance comparison between $IWFP_{WA}$ and $IWFP_{FD}$ 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 $IWFP_{FD}$ is always less than or equal to the number of nodes participating during the mining operation in $IWFP_{WA}$.*

Table 3
Weighted frequent patterns calculations from candidate set in $IWFP_{FD}$.

No.	Candidate patterns	Weight calculation	Weighted support calculation	Result
1	a:8	0.6	0.6 × 8 = 4.8	Passed
2	ab:6	(0.6 + 0.5)/2 = 0.55	0.55 × 6 = 3.3	Passed
3	b:7	0.5	0.5 × 7 = 3.5	Passed
4	ag:4	(0.6 + 0.4)/2 = 0.5	0.5 × 4 = 2.0	Pruned
5	bg:5	(0.5 + 0.4)/2 = 0.45	0.45 × 5 = 2.25	Passed
6	g:5	0.4	0.4 × 5 = 2.0	Pruned
7	abg:4	(0.6 + 0.5 + 0.4)/3 = 0.5	0.5 × 4 = 2.0	Pruned
8	ad:4	(0.6 + 0.35)/2 = 0.475	0.475 × 4 = 1.9	Pruned
9	bd:4	(0.5 + 0.35)/2 = 0.425	0.425 × 4 = 1.7	Pruned
10	abd:4	(0.6 + 0.5 + 0.35)/3 = 0.483	0.483 × 4 = 1.93	Pruned
11	d:4	0.35	0.35 × 4 = 1.4	Pruned

Proof. All the nodes containing non-candidate items cannot appear any prefix tree of the candidate items because $IWFPT_{FD}$ is sorted in frequency descending order. But, for $IWFPT_{WA}$, 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 of candidate items. Hence, the number of nodes participating during the mining operation in $IWFP_{FD}$ cannot be greater than the number of nodes participating in $IWFP_{WA}$. □

Lemma 2. *The number of nodes N participating in the mining operation in $IWFP_{FD}$ reduces when the number of non-candidate items increases for increased minimum threshold value.*

Proof. N is equal to the total number of nodes in $IWFP_{FD}$ when the minimum threshold value is zero. If the minimum threshold increases then some non-candidate nodes from the bottom of the tree cannot appear in any prefix tree of the candidate items. Suppose for minimum threshold x, the number nodes not participating in mining is y. If x increases by d_x amount then y has to increase d_y amount and $d_y \geqslant 0$. So, $y \leqslant y + d_y$. □

4.4. Analysis of completeness/correctness

In this section, we show the completeness/correctness of our approaches in spite of lots of additions/deletions/modifications of transactions.

Lemma 3. For any transaction in a DB, there exists a unique path in an $IWFPT_{WA}/IWFPT_{FD}$ starting from the root.

Proof. Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ be two transactions and their items are sorted in weight ascending order. When we want to insert X into an $IWFPT_{WA}$, we need to check whether the root node contains any child node with *item-id* x_1. If a match is found then the frequency value of x_1 is added there, otherwise, a new node is created. The similar process is recursively applied for other items x_2–x_n 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., $x_1 = y_1$, $x_2 = y_2, \ldots, x_n = y_n$. As a consequence, for any transaction, there is a unique path in an $IWFPT_{WA}$ starting from the root. By considering the items of these two transactions are sorted in frequency descending order, it can be shown that $IWFPT_{FD}$ also has a unique path for each transaction starting from the root. □

Lemma 4. *All the current transactions of*

(i) *an initial database DB*
(ii) *an updated database $DB + db^{+}$ after insertions*
(iii) *an updated database $DB - db^{-}$ after deletions*
(iv) *an updated database $DB \pm db_{mod}$ after modifications*

can be discovered in weight ascending/frequency descending order from an $IWFPT_{WA}/IWFPT_{FD}$.

Proof.

(i) Let T be an $IWFPT_{WA}$ for *DB*. According to Lemma 3, each transaction X has a unique path staring from root in T and they are stored in weight ascending order. Hence, all the transactions can be discovered from T in the weight ascending order. Similarly, it can be shown that all the transactions can be discovered from an $IWFPT_{FD}$ in frequency descending order.
(ii) Consider the original database *DB* is incremented by inserting a group of transactions db^{+}. Similar to the construction process of an $IWFPT_{WA}$, any new transaction inside db^{+} is inserted into the $IWFPT_{WA}$ and is stored in a unique path (according to Lemma 3). Let T' be the modified $IWFPT_{WA}$ for $DB + db^{+}$. According to Lemma 4(i), all the current transactions of $DB + db^{+}$ can be discovered from T' in weight ascending order. Similar result can be shown for an $IWFPT_{FD}$.
(iii) Consider the original database *DB* is decremented by deleting a group of transactions db^{-}. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a transaction in db^{-}. According to Lemma 3, it is stored in a unique path in the $IWFPT_{WA.}$ Therefore, it can be traversed through a unique path in the $IWFPT_{WA}$ in weight ascending order. While traversing the path, frequency value can be reduced by one for each item and thus deletion of X can be completed. The nodes containing zero count after this reduction can be deleted. Now the $IWFPT_{WA}$ contains all the transactions except X. Similarly, other transactions of db^{-} can be removed and the modified $IWFPT_{WA}$ exactly represents the updated database $DB - db^{-}$. Similar result can be shown for an $IWFPT_{FD}$.
(iv) Lemma 4(ii) and 4(iii) show that the $IWFPT_{WA}/IWFPT_{FD}$ can be properly updated for insertion and deletion of a full transaction. It trivially implies that $IWFPT_{WA}/IWFPT_{FD}$ can also be properly updated for modifications of a transaction (insertions/deletions/replacements of items inside a transaction). We can delete the transaction from the $IWFPT_{WA}/IWFPT_{FD}$, and insert the final modified transaction. As a consequence, the modified $IWFPT_{WA}/IWFPT_{FD}$ can exactly represent the updated database $DB \pm db_{mod}$. □

Lemma 4 proves that when a database is updated by additions/deletions/modifications of transactions, our proposed tree structures $IWFPT_{WA}$ and $IWFPT_{FD}$ 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 $IWFPT_{WA}/IWFPT_{FD}$ according to a user-given minimum threshold.

4.5. Analysis of space requirements

Even though $IWFPT_{WA}$ and $IWFPT_{FD}$ 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 $IWFPT_{WA}/IWFPT_{FD}$, two transactions $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$ share the first node if $x_1 = y_1$, first and second node if $x_1 = y_1$ and $x_2 = y_2$, and so on. Stated in other way, a transaction $X = \{x_1, x_2, \ldots, x_n\}$ is totally isolated in an $IWFPT_{WA}$ if no other transaction in *DB* starts with x_1. For example, consider $X = \{a, b, c\}$, then X is totally isolated in an $IWFPT_{WA}/IWFPT_{FD}$ if no other transaction starts with item *a*. The

Table 4
Performance comparison between $IWFP_{WA}$ and $IWFP_{FD}$.

Performance evaluation criteria	$IWFP_{WA}$	$IWFP_{FD}$
No. of nodes in the global tree	31	19
No. of nodes participating in mining operation	30	7
No. of prefix and conditional trees	8	5
No. of nodes in prefix and conditional trees	51	10
No. of global infrequent node deletions in prefix trees	17	0
No. of candidate patterns	10	11

existing research (Cheung & Zaïane, 2003; Grahne & Zhu, 2005; Han et al., 2004, 2007; Koh & Shieh, 2004; Leung et al., 2007; Tanbeer et al., 2009; Yun, 2007a, 2007c, 2008a, 2008b; Yun & Leggett, 2005a) 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 $\mathrm{IWFPT_{FD}}$ achieves a larger prefix-sharing compared to $\mathrm{IWFPT_{WA}}$ since it stores the items in frequency descending order.

The following lemma shows the worst-case memory requirement of $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$.

Lemma 5. *Given a database size |DB|, the size of an $IWFPT_{WA}/IWFPT_{FD}$ (without considering the root) is bounded by $\sum_{X\in DB}|size(X)|$, where X is a transaction in DB.*

Proof. According to Lemma 3, a transaction X contributes at best one path in an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$. Therefore, its maximum size in an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ is $|size(X)|$. Consider the worst-case, where an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ does not get any prefix-sharing in any node. At this situation, the maximum size of an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ is $\sum_{X\in DB}|size(X)|$. □

However, according to Observation 1, an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ usually achieves a lot of prefix-sharing due to the common prefix items inside transactions. Hence, the size of an $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ is normally much smaller than $\sum_{X\in DB}|size(X)|$.

However, the memory requirement of $\mathrm{IWFPT_{WA}}/\mathrm{IWFPT_{FD}}$ is larger than the FP-tree memory requirement since an FP-tree captures only frequent items from transactions according to a user-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 database 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 perform mining operation as well. Because the resultant patterns of current 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.

4.7. Description of algorithms

The pseudo-code of $\mathrm{IWFP_{WA}}$ algorithm is shown in Fig. 11. In line 2, it creates the global header table H to keep the items in weight ascending order. In line 3, it declares a variable *GMAXW*

```
Input: DB, Weight table, group of $db^+$, $db^-$ and $db_{mod}$, minimum threshold ($\delta$)

Output: Weighted frequent patterns

1.  Begin
2.  Create the global header table $H$ to keep the items according to the weight ascending order
3.  Let $GMAXW$ be the maximum weight among all the items
4.  Let $R$ be the root of $IWFPT_{WA}$ and initialize it to $NULL$
5.  Scan the next transaction $T_i$ and insert the new items of $T_i$ in $H$ according to the weight ascending order
6.  Sort the items inside $T_i$ according to the weight ascending order
7.  If Database is original $DB$ or $db^+$ then
8.        Call Insert($T_i$, $R$)
9.  Else If Database is $db^-$ then
10.       Call Delete($T_i$, $R$)
11.  Else
12.       Call Modify($T_i$, $T_j$, $R$)
13.  End If
14.   If $T_i$ is the last transaction of $DB$ or any $db^+/db^-/db_{mod}$ then
15.         $GMAXW$ = weight of the bottom-most item in $H$
16.         Input $\delta$ from the user and
17.              If this is the first $\delta$ value for this $IWFPT_{WA}$, or previous $\delta$ > current $\delta$ then
18.                   For each item $\alpha$ from the bottom of $H$
19.                        If $frequency(\alpha) \times GMAXW \geq \delta$ then
20.                             Call Test_Candidate ($\alpha$, $frequency(\alpha)$)
21.                             Create Prefix tree $PT_\alpha$ with its header table $HT_\alpha$ for item $\alpha$
22.                             Call Mining ($PT_\alpha$, $HT_\alpha$, $\alpha$, $Weight(\alpha)$)
23.                        End If
24.                   End For
25.              Else Find the weighted frequent itemsets from previous result without mining
26.              End If
27.          For any other mining request go to line 16, otherwise go to line 30
28.  Else go to line 5
29.  End If
30.  If no more $db^+/db^-/db_{mod}$ remains then
31.       go to line 34
32.  Else go to line 5
33.  End If
34.  End
```

Fig. 11. The $\mathrm{IWFP_{WA}}$ Algorithm.

Procedure Insert (*T, R*)

```
1.  Begin
2.  if T is NULL
3.      return
4.  End If
5.  Let divide T as [x|X] where x is the first element and X is the remaining list
6.  If R has a child C such that C.item_name = x.item_name then
7.          C.count = C.count +1
8.  Else
9.          Create a new node C as child of R
10.         C.item_name = x.item_name
11.         C.count = 1
12. End If
13. Increase the count of Header table H's entry by one for C.item_name
14. call Insert(X, C)
15. End
```

Procedure Delete (*T, R*)

```
1.  Begin
2.  if T is NULL
3.      return
4.  End If
5.  Let divide T as [x|X] where x is the first element and X is the remaining list
6.  Find a child C of R such that C.item_name = x.item_name
7.  C.count = C.count -1
8.  Decrease the count of Header table H's entry by one for C.item_name
9.  call Delete(X, C)
10. If C.count = 0 then
11.   Delete C from R
12. End If
13. If Header table H's entry for C.item_name becomes zero then
14.   Delete the item H
15. End If
16. End
```

Procedure Modify (*T_1, T_2, R*)

```
1.  Begin
2.  call Delete($T_1$, R)
3.  call Insert($T_2$, R)
4.  End
```

Fig. 12. Insert, delete and modify procedures.

to denote the global maximum weight. In line 4, the root *R* of an $IWFPT_{WA}$ is created and initialized to *NULL*. In line 5, the next transaction T_i is scanned and new items inside it are inserted in *H* according to the weight ascending order. All the items of T_i 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 $db^{+}/db^{-}/db_{mod}$. If the above condition is true, then the current *GMAXW* is defined and minimum threshold (δ) input is taken from the user in lines 15 and 16, respectively. The conditional statement in line 17 checks the current δ with the previous one to take advantages of interactive mining stated in Section 4.6. $IWFP_{WA}$ 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 δ 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 $IWFPT_{WA}/IWFPT_{FD}$. 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 $IWFP_{WA}/IWFP_{FD}$.

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 $IWFPT_{WA}/IWFPT_{FD}$. 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

Procedure Mining (*T, H, α, LMAXW*)

```
1.  Begin
2.  For each item $\beta$ of $H$   // Conditional tree and its header table construction
3.     If frequency($\beta$) $\times$LMAX W $< \delta$
4.        Delete $\beta$ from $H$ and $T$
5.     End If
6.  End For
7.  Let $CT$ be the Conditional tree of $\alpha$  //created from $T$
8.  Let $HC$ be the Header table of Conditional tree $CT$  //created from $H$
9.  For each item $\beta$ in $HC$
10.      Call Test_Candidate($\alpha\beta$, frequency($\alpha\beta$))
11.      Create Prefix tree $PT_{\alpha\beta}$ with its Header table $HP_{\alpha\beta}$ for pattern $\alpha\beta$
12.      Call Mining ($PT_{\alpha\beta}$, $HP_{\alpha\beta}$, $\alpha\beta$, LMAXW)
13. End For
14. End
```

Procedure Test_Candidate (*X, F*)

```
1.  Begin
2.  Let the actual weight of pattern $X$ be $W_X$
3.  Set $W_X = 0$
4.  For each item $x_i$ in the pattern $X$
5.     $W_X = W_X + Weight(x_i)$
6.  End For
7.  $W_X = W_X$ / Length(X)
8.  If $(F \times W_X) \geq \delta$ then
9.     Add $X$ in the weighted frequent pattern list
10. End If
11. End
```

Fig. 13. Mining and Test_Candidate procedures.

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 perform insertion and deletion of transactions in an IWFPT$_{WA}$/IWFPT$_{FD}$. 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 T_{mod}. To perform this modification we need to delete *T* and insert T_{mod} in an IWFPT$_{WA}$/IWFPT$_{FD}$. The Modify procedure (presented at the bottom of Fig. 12) receives two transactions T_1 and T_2. The T_1 transaction represents the transaction to be modified and the T_2 transaction represents the new modified transaction. This procedure at first deletes T_1 and then inserts T_2 to perform this modification.

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

The pseudo-code of IWFP$_{FD}$ 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 IWFPT$_{FD}$ is created and initialized to *NULL*. Procedures of additions/deletions/modifications of transactions are invoked from lines 6 to 14. In lines 15 and 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 IWFP$_{WA}$ 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 denote 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 (http://www.fimi.cs.helsinki.fi/data/) and UCI Machine Learning Repository (http://www.kdd.ics.uci.edu/). These datasets do not provide the weight values of each item. As like the performance evaluation of the previous weight based frequent pattern mining (Tao, 2003; Wang et al., 2004; Yun, 2007a, 2007c, 2008b; Yun & Leggett,

```
Input: DB, Weight table, group of $db^+$, $db^-$ and $db_{mod}$, minimum threshold ($\delta$), N (percentage of $DB/db^+/db^-/db_{mod}$,
       after which restructuring operation should be done each time)

Output: Weighted frequent patterns

1.  Begin
2.  Create the global header table H to keep the items according to the frequency descending order
3.  Let GMAXW be the maximum weight among all the items
4.  Let LMAXW be the maximum local weight
5.  Let R be the root of $IWFPT_{FD}$ and initialize it to NULL
6.  Scan the next transaction $T_i$ and insert the new items of $T_i$ in the bottom of H
7.  Sort the items inside $T_i$ according to the current frequency descending sort order
8.  If Database is original DB or $db^+$ then
9.          Call Insert($T_i$, R)
10. Else If Database is $db^-$ then
11.         Call Delete($T_i$, R)
12. Else
13.         Call Modify($T_i$, $T_j$, R)
14. End If
15. If next N% of database is scanned or end of database found then
16.         Perform bubble sort [25] operation to restructure the tree and header table H.
17. If $T_i$ is the last transaction of DB or any $db^+/db^-/db_{mod}$ then
18.         Calculate the current GMAXW and Set LMAXW = 0
19.         Input $\delta$ from the user
20.             If this is the first $\delta$ value for this $IWFPT_{FD}$, or previous $\delta$ > current $\delta$ then
21.                 For each item $\alpha$ from the top of H,
22.                     If (frequency($\alpha$) × GMAX W) ≥ $\delta$ then
23.                         Call Test_Candidate ($\alpha$, frequency($\alpha$))
24.                         Create Prefix tree $PT_\alpha$ with its header table $HT_\alpha$ for item $\alpha$
25.                         If Weight ($\alpha$) > LMAXW then
26.                             LMAXW = Weight ($\alpha$)
27.                         End If
28.                         Call Mining ( $PT_\alpha$, $HT_\alpha$, $\alpha$, LMAXW)
29.                     End If
30.                 End For
31.             Else Find the weighted frequent itemsets from previous result without mining
32.             End If
33.         For any other mining request go to line 19, otherwise go to line 36
34. Else go to line 6
35. End If
36. If no more $db^+/db^-/db_{mod}$ remains then
37.         go to line 40
38. Else go to line 6
39. End If
40. End
```

Fig. 14. The $IWFP_{FD}$ algorithm.

Table 5
Dataset characteristics.

Datasets	Size (MB)	No. of transactions	No. of distinct items (D)	Max. transaction Length	Min. transaction length	Avg. transaction length (A)	Dense/sparse characteristics ratio $R = (A/D) * 100$ (%)
mushroom	0.56	8124	119	23	23	23	19.327
chess	0.34	3196	75	37	37	37	49.33
T10I4D100K	3.83	100,000	870	29	1	10.1	1.16
pumsb∗	10.7	49,046	2088	63	49	50.48	2.42
kosarak	30.5	990,002	41,270	2498	1	8.1	0.0196

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 *mushroom*, *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 and 2 GB main memory.

Table 5 shows different important characteristics of the synthetic and real-life datasets. Dense and sparse natures of datasets are very useful properties (Verma & Vyas, 2005; Ye et al., 2005). Many pattern mining research works (Chang & Lee, 2005; Cheung & Zaïane, 2003; Dong & Han, 2007; Grahne & Zhu, 2005; Han et al., 2004; Hu et al., 2008; Jiang & Gruenwald, 2006; Koh & Shieh, 2004; Leung et al., 2007; Li et al., 2006; Metwally et al., 2006; Pei & Han, 2000; Raissi et al., 2007; Tanbeer et al., 2009; Verma & Vyas, 2005; Wang et al., 2003; Ye et al., 2005; Yu et al., 2006; Yun, 2007a, 2007c; Yun & Leggett, 2005a, 2005b, 2006; Zhang et al., 2007) evaluate the performances of the algorithms using these characteristics of datasets. A dataset becomes dense when it contains many items per transaction and its number of distinct items is small. Consider the *chess* dataset in Table 5. It has a total of 75 distinct items and its average transaction length is 37. So, 49.33% items are present in every transaction. If we take the measure of R shown in Table 5, it tells the probability of an item to appear in a transaction. We can tell that a dataset is dense if $R > 10\%$ and sparse if $R \leqslant 10\%$.

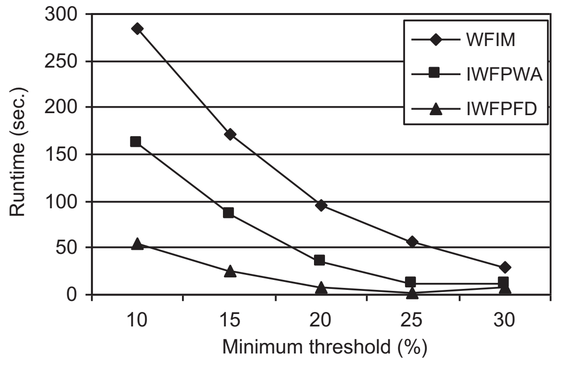


Fig. 15. Runtime comparison on the *mushroom* dataset.

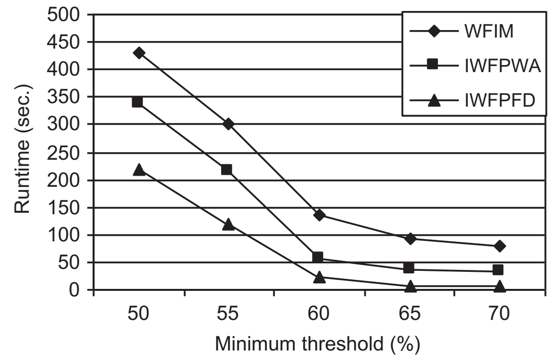


Fig. 16. Runtime comparison on the *chess* dataset.

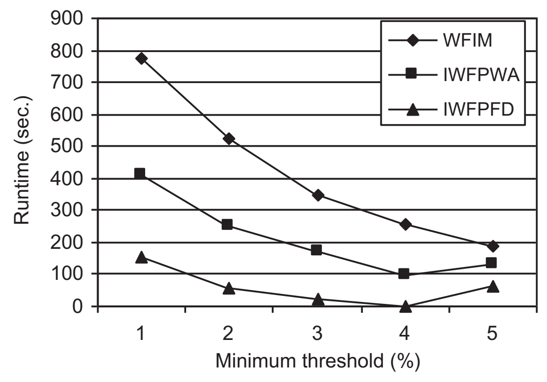


Fig. 17. Runtime comparison on the *T10I4D100K* dataset.

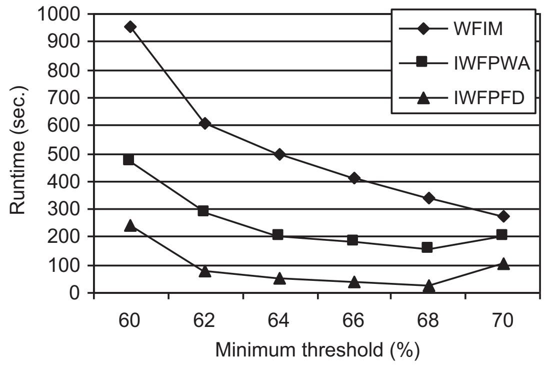


Fig. 18. Runtime comparison on the *pumsb∗* dataset.

Table 6
Runtime distribution (s).

		Tree construction time	Tree restructuring time	Mining time	Total time
mushroom minimum threshold (30%)	$IWFP_{WA}$	5.582	0	6.781	12.363
	$IWFP_{FD}$	5.623	0.798	0.597	7.018
chess minimum threshold (70%)	$IWFP_{WA}$	1.521	0	31.249	32.77
	$IWFP_{FD}$	1.533	0.731	5.657	7.921
T10I4D100K minimum threshold (5%)	$IWFP_{WA}$	50.485	0	82.528	132.013
	$IWFP_{FD}$	50.374	10.276	1.781	62.431
pumsb∗ minimum threshold (70%)	$IWFP_{WA}$	63.952	0	138.195	202.147
	$IWFP_{FD}$	64.258	28.018	13.219	105.495

Table 7
Effectiveness of IWFP with a comparison of no. of interactive weighted frequent patterns and no. of traditional interactive frequent patterns.

Dataset	Minimum threshold (%)	No. of frequent patterns	No. of weighted frequent patterns
T10I4D100K	1	385	174
	1.5	237	105
	2	155	69
	2.5	107	43
	3	60	24
mushroom	15	98,575	41,309
	20	53,663	24,087
	25	5545	2359
	30	2735	1076
	35	1189	492

Obviously, dense datasets have too many long frequent as well as weighted frequent patterns. The dataset *chess* is too dense and the dataset *mushroom* is moderately dense. Similarly, the dataset *T10I4D100K* is moderately sparse and the dataset *kosarak* is too sparse.

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 patterns. The experimental results in two dense and two sparse datasets show that our algorithms are better than WFIM even if in the worst case. Fig. 15 shows the runtime comparison in mushroom 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 taken 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

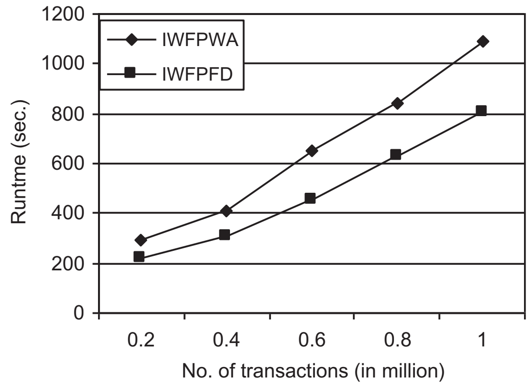


Fig. 19. Database is increasing by $db^+ = 0.2$ million on the *kosarak* dataset.

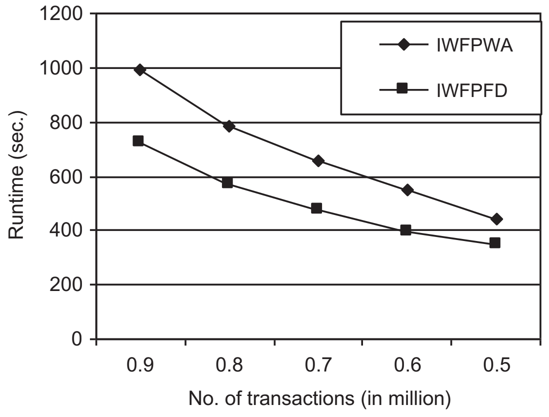


Fig. 20. Database is decreasing by $db^- = 0.1$ million on the *kosarak* dataset.

Table 8
Memory comparison (MB).

	mushroom	*chess*	*T10I4D100K*	*pumsb∗*	*kosarak*
$IWFPT_{WA}$	0.712	0.659	15.084	53.714	204.438
$IWFPT_{FD}$	0.431	0.362	11.237	42.731	173.127

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 comparison in *chess*, *T10I4D100K* and *pumsb∗* datasets, respectively for the worst case of interactive mining. Table 6 shows the runtime distribution (seconds) of the total runtime for our proposed two algorithms. It is remarkable that $IWFP_{FD}$ achieves a huge gain in mining time by sacrificing a small amount of restructuring time. The restructuring threshold $N = 10\%$ is used here for $IWFP_{FD}$.

Traditional frequent pattern mining methods may discover many spurious patterns as they consider equal weight values for every item. Many unimportant patterns having high frequency values and low weight values can be generated. By considering different weights of items, weighted frequent pattern mining can discover fewer but more important and interesting patterns compared to traditional frequent patterns. Table 7 shows the effectiveness of IWFP by comparing the number of traditional interactive frequent patterns and interactive weighted frequent patterns. One synthetic dataset *T10I4D100K* and one real-life dataset *mushroom* 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 performed mining operation with a minimum threshold of 5%. Another 0.2 million transactions were added in the tree and 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 adding each db^+, we have restructured the $IWFPT_{FD}$ before mining. 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 $IWFPT_{FD}$ 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 observe 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 (Cheung & Zaïane, 2003; Koh & Shieh, 2004; Leung et al., 2007; Li et al., 2006; Tanbeer et al., 2009; Zhang et al., 2007) 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. Therefore, our IWFPT prefix tree structures are efficient for weighted frequent pattern mining with the recently available gigabyte-range memory. Table 8 also shows that $IWFPT_{FD}$ requires much less amount of memory compared to $IWFPT_{WA}$ because of having more prefix-sharing.

6. Discussion

As discussed in Section 1, by considering different weights of items, WFP mining can discover more useful knowledge from real-life market basket databases. In this section, we elaborately discuss some other real-life applications of WFP mining.

In a stock market, each share may have different importance due to different real-life reasons. Therefore, mining only frequent patterns cannot extract the most interesting shares in a share market. By finding weighted frequent share patterns, stock investors can obtain more useful information to make their policies. In a similar way, WFP mining can be effective in extracting important knowledge from the auction market in which buyers enter competitive bids and sellers enter competitive offers simultaneously. In application domains such as financial data analysis, the telecommunications industry, and the retail industry, weighted frequent pattern mining can be used to detect unusual access patterns or sequences related to financial crimes, fraudulent telecommunications activities, and the purchase of many expensive items within a short time (Yun, 2007c). In this case, higher weights are given to items which have been previously found in fraudulent patterns (Yun, 2007c).

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. However, 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, IWFPT$_{WA}$ provides easier maintenance phase during incremental mining. On the other hand, a rather compact structure of IWFPT$_{FD}$ requires tree restructuring operation before mining. However, we have shown that the tree restructuring cost of IWFPT$_{FD}$ is insignificant compared to the gain achieved in mining phase from the tree compactness that reduces the overall runtime. Both of the algorithms are capable of using the previous tree structures and mining results to reduce the calculations by remarkable amount. They are also applicable in real time data processing like data stream as they require only one database scan. Extensive performance analyses show that our tree structures and algorithms are highly scalable that can efficiently handle lots of insertions, deletions and modifications, with a large number of distinct items, and interactive mining.

References

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).

Agrawal, R., Imieliński, 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).

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.

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.

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.

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.

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).

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

Grahne, G., & Zhu, J. (2005). Fast algorithms for frequent itemset mining using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering, 17*(10), 1347–1362.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery, 15*, 55–86.

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

Hu, T., Sung, S. Y., Xiong, H., & Fu, Q. (2008). Discovery of maximum length frequent patterns. *Information Science, 178*, 69–87.

Jiang, N., & Gruenwald, L. (2006). Research issues in data stream association rule mining. *SIGMOD Record, 35*(1).

Koh, J.-L., & Shieh, S.-F. (2004). An efficient approach for maintaining association rules based on adjusting FP-tree structures. *Proceedings of the DASFAA'04*, 417–424.

Lee, A. J. T., Tsao, W.-K., Chen, P.-Y., Lin, M.-C., & Yang, S.-H. (2010). Mining frequent closed patterns in pointset databases. *Information Systems, 35*, 335–351.

Lee, Y.-S., & Yen, S. –J. (2008). Incremental and interactive mining of web traversal patterns. *Information Science, 178*, 287–306.

Leung, C. K.-S., Khan, Q. I., Li, Z., & Hoque, T. (2007). CanTree: A canonical-order tree for incremental frequent-pattern mining. *Knowledge and Information Systems, 11*(3), 287–311.

Li, H.-F. (2009). Interactive mining of top-k frequent closed itemsets from data streams. *Expert Systems with Applications, 36*, 10779–10788.

Li, X., Deng, Z.-H., & Tang, S. (2006). A fast algorithm for maintenance of association rules in incremental databases. *Advanced Data Mining and Applications (ADMA 06), 43*, 56–63.

Lim, A. H. L., & Lee, C.-S. (2010). Processing online analytics with classification and association rule mining. *Knowledge-Based Systems, 23*, 248–255.

Lin, C.-W., Hong, T. –P., & Lu, W. –H. (2009). The Pre-FUFP algorithm for incremental mining. *Expert Systems with Applications, 36*, 9498–9505.

Metwally, A., Agrawal, D., & Abbadi, A. E. (2006). An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems (TODS), 31*(3), 1095–1133.

Pei, J., Han, J. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of ACM SIGMOD workshop on research issues in data mining and knowledge discovery* (pp. 21–30). USA.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., et al. (2004). Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering, 16*(11), 1424–1440.

Raissi, C., Poncelet, P., & Teisseire, M. (2007). Towards a new approach for mining frequent itemsets on data stream. *Journal of Intelligent Information Systems, 28*, 23–36.

Song, W., Yang, B., & Xu, Z. (2008). Index-BitTableFI: An improved algorithm for mining frequent itemsets. *Knowledge-Based Systems, 21*, 507–513.

Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S., & Lee, Y.-K. (2009). Efficient single-pass frequent pattern mining using a prefix-tree. *Information Sciences, 179*(5), 559–583.

Tao, F. (2003). Weighted association rule mining using weighted support and significant framework. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 661–666). Washington, DC, USA.

Verma, K., & Vyas, O. P. (2005). Efficient calendar based temporal association rule. *SIGMOD Record, 34*(3), 63–70.

Wang, J., Han, J., Pei, J. (2003). CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 236–245). Washington, DC, USA.

Wang, W., Yang, J., & Yu, P. S. (2004). WAR: Weighted association rules for item intensities. *Knowledge Information and Systems, 6*, 203–229.

Xiong, H., Tan, P.-N., & Kumar, V. (2006). Hyperclique pattern discovery. *Data Mining and Knowledge Discovery, 13*, 219–242.

Ye, F.-Y., Wang, J.-D., Shao, B.-L. (2005). New algorithm for mining frequent itemsets in sparse database. In *Proceeding of the 4th International Conference on Machine learning and Cybernetics* (pp. 1554–1558).

Yu, J. X., Chong, Z., Lu, H., Zhang, Z., & Zhou, A. (2006). A false negative approach to mining frequent itemsets from high speed transactional data streams. *Information Sciences, 176*, 1986–2015.

Yun, U. (2007a). Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. *Information Science, 177*, 3477–3499.

Yun, U. (2007b). WIS: Weighted interesting sequential pattern mining with a similar level of support and/or weight. *ETRI Journal, 29*(3).

Yun, U. (2007c). Mining lossless closed frequent patterns with weight constraints. *Knowledge-Based Systems, 20*, 86–97.

Yun, U. (2008a). A new framework for detecting weighted sequential patterns in large sequence databases. *Knowledge-Base Systems, 21*, 110–122.

Yun, U. (2008b). An efficient mining of weighted frequent patterns with length decreasing support constraints. *Knowledge-Based Systems, 21*, 741–752.

Yun, U., Leggett, J. J. (2005). WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In *Proceedings of the fourth SIAM international conference on data mining* (pp. 636–640). USA.

Yun, U., Leggett, J. J. (2005b). WLPMiner: Weighted frequent pattern mining with length decreasing support constraints. In: *Proceedings of the ninth Pacific–Asia conference on knowledge discovery and data mining (PAKDD)* (pp. 555–567). Hanoi, Vietnam.

Yun, U., & Leggett, J. J. (2006). WIP: Mining weighted interesting patterns with a strong weight and/or support affinity. *SDM'06*.

Zhang, S., Zhang, J., & Zhang, C. (2007). EDUA: An efficient algorithm for dynamic database mining. *Information Science, 177*, 2756–2767.