

WFIM: Weighted Frequent Itemset Mining with a weight range and a minimum weight

Unil Yun and John J. Leggett
Department of Computer Science
Texas A&M University
College Station, TX 77843, U.S.A.
{yunei, leggett@cs.tamu.edu}

ABSTRACT

Researchers have proposed weighted frequent itemset mining algorithms that reflect the importance of items. The main focus of weighted frequent itemset mining concerns satisfying the downward closure property. All weighted association rule mining algorithms suggested so far have been based on the Apriori algorithm. However, pattern growth algorithms are more efficient than Apriori based algorithms. Our main approach is to push the weight constraints into the pattern growth algorithm while maintaining the downward closure property. In this paper, a weight range and a minimum weight constraint are defined and items are given different weights within the weight range. The weight and support of each item are considered separately for pruning the search space. The number of weighted frequent itemsets can be reduced by setting a weight range and a minimum weight, allowing the user to balance support and weight of itemsets. WFIM generates more concise and important weighted frequent itemsets in large databases, particularly dense databases with low minimum support, by adjusting a minimum weight and a weight range.

1. Introduction

Before the FP-tree based mining method [7] was developed, Apriori approaches [13, 14] were usually used based on the downward closure property. That is, if any length k pattern is not frequent in a transaction database, superset patterns can not be frequent. Using this characteristic, Apriori based algorithms prune candidate itemsets. However, Apriori based algorithms need to generate and test all candidates. Moreover, they must repeatedly scan a large amount of the original database in order to check if a candidate is frequent or not. This is inefficient and ineffective. To overcome this problem, pattern growth based approaches [7, 8, 9, 10] were developed. FP-tree based methods mine the complete set of frequent patterns using a divide and conquer method to reduce the search space without generating all the candidates. An association mining algorithm generates frequent patterns and then makes association rules satisfying a minimum support. One of the main limitations of the traditional model for mining frequent itemsets is that all the items are treated uniformly, but real items have different importance. For this reason, weighted frequent itemset mining algorithms [1, 2, 5] have been suggested. The items are given different weights in the transaction database. The main focus in weighted frequent

itemset mining concerns satisfying the downward closure property. The downward closure property is usually broken when different weights are applied to the items according to their significance. Most of the weighted association rule mining algorithms such as [1, 2, 5] have adopted an Apriori algorithm based on the downward closure property. They have suggested the sorted closure property [4], the weighted closure property [1] or other techniques [2, 3, 5] in order to satisfy the downward closure property. However, Apriori based algorithms use candidate set generation and test approaches. It can be very expensive to generate and test all the candidates. Performance analyses [11, 12] have shown that frequent pattern growth algorithms are efficient at mining large databases and more scalable than Apriori based approaches. However, there has been no weighted association rule mining using the pattern growth algorithm because the downward closure property is broken by simply applying the FP growth methodology.

We propose an extended model to tackle these problems of previous frequent itemset mining. Our main goal in this framework is to push weight constraints into the pattern growth algorithm while keeping the downward closure property. The remainder of the paper is organized as follows. In section 2, we describe the problem definition of weighted association rules. In Section 3, we develop a WFIM (Weighted Frequent Itemset Mining). Section 4 shows the extensive experimental results. Finally, conclusions are presented in section 5.

2. Problem definition

Let $I = \{i_1, i_2, \dots, i_n\}$ be a unique set of items. A transaction database, TDB, is a set of transactions in which each transaction, denoted as a tuple $\langle tid, X \rangle$, contains a unique tid and a set of items. An itemset is called a k -itemset if it contains k items. An itemset $\{x_1, x_2, \dots, x_n\}$ is also represented as x_1, x_2, \dots, x_n . The support of itemset A is the number of transactions containing itemset A in the database. A weight of an item is a non-negative real number that shows the importance of each item. We can use the term, *weighted itemset* to represent a set of weighted items. A simple way to obtain a weighted itemset is to calculate the average value of the weights of the items in the itemset. As defined by previous studies [1, 2, 5], the problem of weighted association rule mining is to find the complete set of association rules satisfying a support constraint and a weight constraint in the database.

TID	Set of items	Frequent Item list
100	a, c, d, f, i, m	c, d, f, m
200	a, c, d, f, m, r	c, d, f, m, r
300	b, d, f, m, p, r	d, f, m, p, r
400	b, c, f, m, p	c, f, m, p
500	c, d, f, m, p, r	c, d, f, m, p, r
600	d, m, r	d, m, r

Table 1: transaction database TDB.

3. WFIM (Weighted Frequent Itemsets Mining)

We suggest an efficient weighted frequent itemset mining algorithm in which the main approach is to push weight constraints into the pattern growth algorithm and provide ways to keep the downward closure property. WFIM adopts an ascending weight ordered prefix tree. The tree is traversed bottom-up because the previous matching can not maintain the downward closure property. A support of each itemset is usually decreased as the length of an itemset is increased, but the weight has a different characteristic. An itemset which has a low weight sometimes can get a higher weight after adding another item with a higher weight, so it is not guaranteed to keep the downward closure property. We present our algorithm in detail and show actual examples in order to illustrate the steps in the FP-tree construction for weighted frequent itemset mining and the mining of a weighted frequent itemset from the FP-tree.

3.1 Preliminaries

Definition 3.1 Weight Range (WR)

The weight of each item is assigned to reflect the importance of each item in the transaction database. A weight is given to an item with a weight range, $W_{\min} \leq W \leq W_{\max}$.

Definition 3.2 minimum weight constraint (min_weight)

In the WFIM, we want to give a balance between the two measures of weight and support. Therefore, we define a minimum weight constraint like a minimum support (min_sup) in order to prune items which have lower weights. Itemset X is defined as a useless itemset if the support of itemset X is less than a minimum support and its weight is also less than a minimum support.

Definition 3.3 Maximum Weight and Minimum Weight

The maximum weight (MaxW) is defined as the value of the maximum weight of items in a transaction database or a conditional database. The minimum weight (MinW) is defined as the value of the minimum weight of items in a transaction database or a conditional database. In WFIM, a MaxW is used in a transaction database and a MinW is used in a conditional database.

Definition 3.4 Weighted Frequent Itemset (WFI)

An itemset X is a weighted infrequent itemset if following pruning, condition 1 or condition 2 below is satisfied. If the itemset X does not satisfy both of these, the itemset X is called a weighted frequent itemset.

Item (min_sup = 3)	a	b	c	d	f	i	m	p	r
Support	2	2	4	5	5	1	6	3	4
Weight ($1.0 \leq WR_1 \leq 1.5$)	1.3	1.1	1.4	1.2	1.5	1.1	1.3	1.0	1.5
Weight ($0.7 \leq WR_2 \leq 1.3$)	1.1	1.0	0.9	1.0	0.7	0.9	1.2	0.8	1.3
Weight ($0.7 \leq WR_3 \leq 0.9$)	0.85	0.75	0.8	0.9	0.75	0.7	0.85	0.7	0.9
Weight ($0.2 \leq WR_4 \leq 0.7$)	0.5	0.3	0.6	0.4	0.7	0.3	0.5	0.2	0.7

Table 2: example of sets of items with different WRs.

Pruning condition 1 (support < min_sup && weight < min_weight) The support of an itemset is less than a minimum support and the weight of an itemset is less than a minimum weight constraint.

Pruning condition 2 (support * MaxW (MinW) < min_sup)

In a transaction database, the value of multiplying itemset's support with a maximum weight (MaxW) among items in the Transaction database is less than a minimum support. In conditional databases, the value of multiplying the support of an itemset with a minimum weight (MinW) of a conditional pattern in the FP-trees is less than a minimum support.

TID	WFI list ($1.0 \leq WR_1 \leq 1.5$) MinW = 1.0	WFI list ($0.7 \leq WR_2 \leq 1.3$) MinW = 0.7	WFI list ($0.7 \leq WR_3 \leq 0.9$) MinW = 0.7	WFI list ($0.2 \leq WR_4 \leq 0.7$) MinW = 0.2
100	a, c, d, f, m	c, d, f, m	c, d, f, m	d, f, m
200	a, c, d, f, m, r	c, d, f, m, r	c, d, f, m, r	d, f, m
300	b, d, f, m, p, r	d, f, m, p, r	d, f, m, r	d, f, m
400	b, c, f, p, m	c, f, p, m	c, f, m	f, m
500	c, d, f, m, p, r	c, d, f, m, p, r	c, d, f, m, r	d, f, m
600	d, m, r	d, m, r	d, m, r	d, m

Table 3: weighted frequent itemsets with different WRs.

Example 1: Table 1 shows transaction database TDB. Table 2 shows example sets of items with different weights. Assume that the minimum support threshold is 3. The frequent list is: Frequent_list = <a:2, b:2, c:4, d:5, f:5, i:1, m:6, p:3, r:4>. The columns in Table 3 show the set of weighted frequent itemsets after pruning weighted infrequent itemsets using pruning condition1 and pruning condition 2 in definition 3.4 by applying different WRs. For example, when WR_3 is applied, item p's support is 3, MaxW is 0.9 and the value (2.7) of multiplying item's support (3) with a MaxW (0.9) of an itemset in the TDB is less than minimum support (3) so item "p" can be removed. Meanwhile, the number of WFI can be increased when WR_1 is used as the weight range. The support of Item "a" in the transaction database is 2. However, a maximum weight is 1.5 so the value (3) of multiplying item's support (2) with a MaxW (1.5) of an itemset is greater than or equal to a minimum support (3) so this item is added in the WFI list.

Example 2: Let us show another example by changing a WR and a min_weight. In this example, Table1 and Table 2 are used and pruning condition 1 in definition 3.4 is applied using WR_2 as a weight range. If the min_weight is 1.2, items "a", "b" and "i" are pruned because the support of these items is less than a minimum support and the weight of these items is also less than a minimum weight.

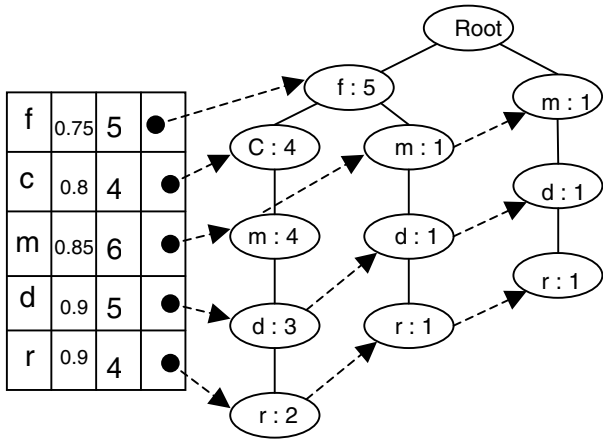


Fig. 1. The global FP tree

In a similar way, we can get the following results by changing min_weight. If min_weight is 1.1, items “i” and “b” are pruned and if min_weight is 1.0, item “i” is pruned. As a result, the number of weighted frequent items can be changed according to different min_weights. When pruning condition 2 of definition 3.4 is considered, if a weight range is $1.0 \leq WR_1 \leq 1.5$, only item i is pruned because the value of multiplying item i’s support (1) with a maximum weight (1.5) is less than a minimum support (3). However, the items “a” and “b” with 2 as a support are not pruned because the value of multiplying the support (2) of item “a” and “b” with a maximum weight (1.5) is equal to a minimum support (3). In a similar fashion, we can get the following results by changing WRs. If the weight range is $0.7 \leq WR_3 \leq 0.9$, item “a”, “b”, “i” and “p” are pruned and if the weight range is $0.2 \leq WR_4 \leq 0.7$, item “a”, “b”, “c”, “i”, “p” and “r” are pruned. Thus, the number of weighted frequent items can be adjusted by using the different WRs.

3.2 FP (Frequent Pattern) tree structure

WFIM uses FP-trees as a compression technique. FP-trees are mainly used in pattern growth algorithms. WFIM computes local frequent items of a prefix by scanning its projected database. The FP-trees in our algorithm are made as follows. Scan the transaction database one time and count the support of each item and check the weight of each item. After this, sort the items in weight ascending order. Although supports of items may be lower than the minimum support and infrequent, the items can not be deleted since infrequent items may become weighted itemsets in the next step. The weighted infrequent items are removed according to pruning conditions 1 and 2 in definition 3.4. For instance, assume that WR_3 is used as a WR, min_sup is 3 and min_weight is 0.7. Then, items “a”, “b”, “p”, and “i” are removed. When an item is inserted in the FP-tree, as already discussed, a weighted infrequent item is removed and the rest, weighted frequent items and itemsets, are sorted by weight ascending order. As shown in [7], each node in the FP-tree has item-id, a weight, count and node link. Separate header tables exist for each FP tree and there is an entry for each item in the header table. Fig. 1 presents the global FP-tree and corresponding header table for this example in WFIM.

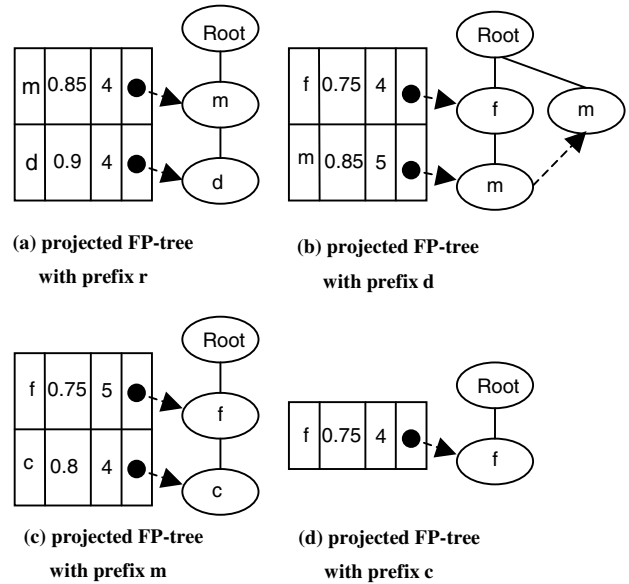


Fig. 2. Conditional FP trees.

3.3 Bottom up Divide and Conquer

After a global FP-tree is generated from the transaction database, WFIM mines frequent itemsets from the FP-tree. The weighted frequent itemsets are generated by adding items one by one. WFIM adapts the divide and conquer approach for mining weighted frequent itemsets. It divides mining the FP-tree into mining smaller FP trees. In the above example of Fig 1, WFIM mines (1) the patterns containing item “r” which has the highest weight, (2) the patterns including “d” but not “r”, (3) the patterns containing “m” but no “r” nor “d”, (4) the patterns containing “c” but no “r”, “d” nor “m”, and finally the patterns containing item “f”. WFIM can find all the subsets of weighted frequent itemsets.

To begin with, for node r, we generate a conditional database by starting from r’s head and following r’s node link. The conditional database for prefix “r:4” contains three transactions: <dmcf:2>, <dmf:1> and <dm:1>. Previous FP-growth algorithms only consider a support in each conditional database so an item “c” is only pruned because the support (2) of item “c” is less than a minimum support (3). However, in WFIM, before constructing the FP-tree, conditions in definition 3.4 are applied in order to check whether it is a weighed frequent itemset. From Table 2, we know that the weights of item “r”, “d”, “m”, “c”, and “f” are 0.9, 0.9, 0.85, 0.8 and 0.75 respectively. There is only one item “r” in the conditional pattern, so the MinW of the conditional pattern is 0.9. By applying pruning conditions, we can see that not only an item “c” but also an item “f” are pruned because the value (2.7) of multiplying item f’s support (3) with a minimum weight (0.9) is less than a minimum support (3). Although the support of item “f” is equal to the minimum support, the item “f” with a lower weight can be deleted. After that, Fig. 2 (a) shows a FP- tree with prefix “r”. As you can see, the conditional FP-tree for prefix “r:4” is a single path tree <md:4>. We can generate all kinds of combinations of items including a conditional pattern “r”. That is, “r”, “rm”, “rd” and “rmd”. The important point is that we can use the minimum weight (MinW) of conditional patterns instead of the maximum

weight (MaxW) of the conditional database or the MaxW of a conditional pattern to prune weighted infrequent itemsets because WFIM uses an ascending weight ordered prefix tree structure to construct conditional databases and the MinW of a prefix is always greater than or equal to the weights of all the items in a conditional database. Hence, the minimum weight of a conditional pattern in the conditional database can be used for applying pruning conditions in definition 3.4.

For node “d”, WFIM derives a frequent pattern (d:5) and three paths in the FP-tree : <mf:3>, <mf:1> and <m:1>. In the FP-growth algorithm, no item is pruned. However, in WFIM, item “c” is pruned in this conditional database with prefix “d”, since the value (2.7) of multiplying item c’s support (3) with the minimum weight (0.9) is less than minimum support (3) although the supports of these items are greater than a minimum support. However, item “m” and item “f” are not pruned. After removing weighted infrequent itemsets in the conditional database, the projected FP-tree for the prefix “d:5” is constructed. Fig 2 (b) shows a conditional FP-tree for prefix d:5. It’s not a single FP-tree, so we divide the conditional FP-tree to mine even smaller conditional FP-trees recursively. As a result, after a recursive process, we obtain weighted frequent itemsets based on conditional pattern “d”: “d:5”, “dm:5”, “df:4” and “dmf:4”. Similarly, we can build projected FP-trees from the global FP-tree and mine weighted frequent itemsets from them recursively. These FP-trees are shown in Fig. 2(c)-(d). (The FP-tree for prefix f:5 is empty and not shown here).

3.4 WFIM algorithm

WFIM can push weight constraints into the pattern growth algorithm and show how to keep the downward closure property. A weight range and a minimum weight are defined and items are given different weights within the weight range. Now, we summarize the weighted frequent itemset mining process and present the mining algorithm.

ALGORITHM. (WFIM): Weighted frequent itemset mining with a weight range and a minimum weight constraint in a large transaction database.

- Input:** (1) A transaction database : TDB,
 (2) minimum support threshold : min_sup,
 (3) weights of the items within weight range : w_i ,
 (4) minimum weight threshold : min_weight

Output: The complete set of weighted frequent itemsets.

Method:

1. Scan TDB once to find the global weighted frequent items satisfying the following definition: An itemset X is a weighted frequent itemset if the following pruning conditions 1 and 2 are not satisfied.

Condition 1.1: (support < min_sup && weight < min_weight)

Condition 1.2: (support * MaxW < min_sup)

2. Sort items in weight ascending order. The sorted weighted frequent item list forms the weight_order and header of FP tree.

3. Scan the TDB again and build a global FP-tree using weight_order.

4. Mine a global FP-tree for weighted frequent itemset mining in a bottom up manner. Form conditional databases for all remaining

Data sets	Size	#Trans	#Items	A.(M.) t. l.
Connect	12.14M	67557	150	43 (43)
BMS-webView-1	1.28M	59601	497	2.51 (267)
T10I4Dx	5.06-50.6M	200k-1000k	1000	10 (31)

Table 4: Data Characteristics

items in weight_list and complete local weighted frequent itemsets for the conditional databases. (An itemset X is a weighted frequent itemset if the following pruning conditions 1 and 2 are not satisfied).

Condition 4.1: (support < min_sup && weight < min_weight)

Condition 4.2: (support * MinW < min_sup)

5. When all the items in the global header table have been mined, WFIM is finished.

4. Performance Evaluation

In this section, we present our performance study over various datasets. In our experiments, we compared WFIM with FP-growth. We used two real datasets and one synthetic dataset which have been used in pervious experiments [7, 8, 9, 10, 11, 12]. Table 4 shows the characteristic of these datasets. The two real datasets used are connect, and BMS-webView-1. The connect dataset is very dense and includes game state information. The BMS-webView-1 dataset is a very sparse dataset with a web log. These real datasets can be obtained from the UCI machine learning repository [15]. The synthetic datasets were generated from the IBM dataset generator. We use T10I4D100k which is very sparse and contains 100,000 transactions. However, the synthetic datasets T10I4Dx contain 100k to 1000k transactions. These datasets have been used to test scalability in the previous performance evaluations [9, 10, 11, 12]. WFIM is written in C++. In our experiment, a random generation function generates weights for each item. All experiments were performed on a Unix machine. First, we show how the number of weighted frequent itemsets in dense databases and sparse databases with very low minimum support can be adjusted by user’s feedback. Second, we show how WFIM has good scalability against the number of transactions in the datasets. In Fig. 3, we can see that the number of WFI is decreased as the weight range is decreased. In addition, WFIM can adjust the number of weighted frequent itemsets in the dense database with very low minimum support. For instance, in the connect dataset with 10% minimum support, WFIM generates 11003 with a WR: 0.05 – 0.15 and 1989 with a WR: 0.05 – 0.14. Therefore, the number of weighted frequent itemsets can be adjusted by user’s feedback. From Fig. 4, the runtime of WFIM is shown under different WRs.

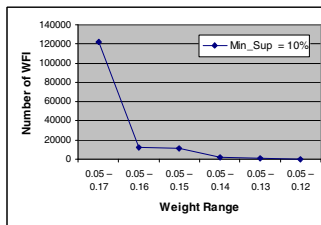


Fig. 3. Num of WFI (Connect dataset)

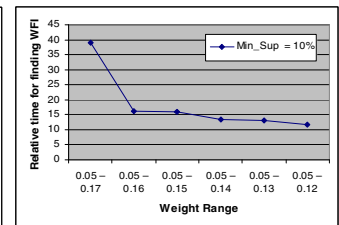


Fig. 4. Runtime (Connect dataset)

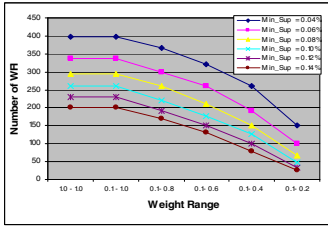


Fig. 5. Num of WFI (BMS-WebView-1)

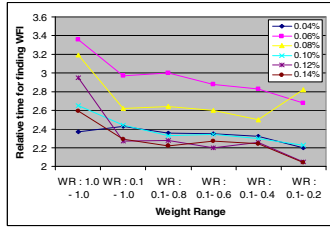


Fig. 6. Runtime (BMS-WebView-1)

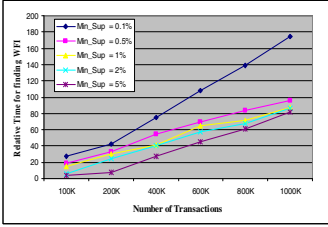


Fig. 7 Runtime (T1014DxK) (WR: 0.2-0.8)

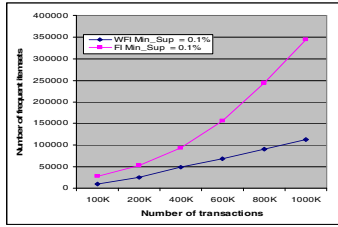


Fig. 8. Num of WFI (T1014DxK) (WR: 0.2-0.8)

The runtime is sharply reduced even in the connect dataset with a low minimum support as the weight range becomes lower. Fig. 5 and Fig. 6 demonstrate the results of using sparse dataset BMS-WebView-1. In Fig. 5, we can see that the number of WFI is reduced as the minimum support becomes lower in the sparse dataset with the same WR. Moreover, the number of weighted frequent itemsets becomes smaller by diminishing WRs although the minimum support is the same. In Fig. 6, the runtime for finding weighted frequent itemsets is shown according to different WRs as the minimum support is lower. We can see that the run time also can be reduced by diminishing WRs. In the several datasets, WFIM can generate smaller weighted frequent itemsets. It is difficult to reduce frequent itemsets without changing a minimum support. However, WFIM can reduce only the number of frequent itemsets by adjusting WRs when giving weights to each item. In addition, *WFIM can generate orders of magnitude fewer patterns than the traditional weighted frequent itemset mining algorithms*. To test the scalability with the number of transactions, the T1014DxK dataset was used. WFIM scales much better than previous weighted frequent itemset mining algorithms which are based on Apriori like approaches [13, 14] since WFIM is based on FP-growth algorithms. In this scalability test, WFIM is compared with FP-growth. Both WFIM and FP-growth show linear scalability with the number of transactions from 100k to 1000k. However, WFIM is much more scalable than FP-growth. In Fig. 7 and Fig. 8, the difference between WFIM and FP-growth becomes clear. We first tested the scalability in terms of base size from 100K tuples to 1000K tuples and different minimum support of 0.1% to 5%. From Fig. 7, we can see that WFIM has much better scalability in terms of base size. The slope ratio for each different minimum support is almost similar. We can also compare WFIM with the FP-growth algorithm in terms of run time. In Fig. 7, you can see that WFIM is faster than FP-growth. More importantly, WFIM can reduce the runtime by adjusting WRs. That is, users can diminish the runtime and the number of WFI by reducing a WR. From Fig. 8, we compared WFIM with FP-growth in order to show that WFIM can generate a more concise result set considering not only frequency of itemsets but also their importance. Our above experiments showed that WFIM can generate smaller but important weighted frequent itemset with various WRs.

Support of connect dataset	Number of W.F.I WR: 0.5-1.5 MW: 1.5	Number of W.F.I WR: 0.5-1.5 MW: 1.0	Number of W.F.I WR: 0.5-1.5 MW: 0.5	Num of F.C.I	Num of F.I
64179 (95%)	125	784	1471	812	2205
60801 (90%)	690	2346	5312	3486	27127
54046 (80%)	2769	2989	3044	15107	533975
47290 (70%)	3997	4089	4093	35875	4129839

Table 5. Comparison of frequent itemsets.

The Table 5 lists the number of weighted frequent itemsets (WFI) with various WRs and min_weights, frequent itemsets (FI) and frequent closed itemsets (FCI). From Table 5, WFIM can generate smaller WFI by using different WRs and min_weights. For example, in Table 4, the number of WFI at a minimum support: 90%, a WR: 0.5 - 1.5 and a min_weight: 0.5, is 5312. However, the number of WFI can be reduced to 2346 with a min_weight: 1.0 and can be further reduced to 690 with a MW: 0.5. The numbers of frequent closed itemsets and frequent itemsets are 3486 and 27127 respectively. In this way, the proper number of weighted frequent itemsets can be found by adjusting a minimum weight.

5. Conclusion

In this paper, we developed WFIM which focuses on weighted frequent itemset mining based on a pattern growth algorithm. The extensive performance analysis shows that WFIM is efficient and scalable in weighted frequent itemset mining. Many improved algorithms using divide and conquer methods have been suggested. In future work, the WFIM can be extended by using a combination of FP-growth based algorithms with better performance.

6. REFERENCES

- [1] Feng Tao, *Weighted Association Rule Mining using Weighted Support and Significant framework*. ACM SIGKDD, Aug 2003.
- [2] Wei Wang, Jiong Yang, Philip S. Yu, *Efficient mining of weighted association rules (WAR)*, ACM SIGKDD, Aug 2000.
- [3] K. Wang, Y. He and J. Han, *Mining Frequent Itemsets Using Support Constraints*, VLDB, Sep 2000.
- [4] Bing Liu, Wynne Hsu, Yiming Ma, *Mining Association Rules with Multiple Minimum Supports*. ACM SIGKDD, June 1999.
- [5] C. H. Cai, Ada Wai-Chee Fu, C. H. Cheng, and W. W. Kwong. *Mining association rules with weighted items*. IDEAS'98, July 1998.
- [6] J. Han and Y. Fu, *Mining Multiple-Level Association Rules in Large Databases*, IEEE TKDE, September/October 1999, pp. 798-805.
- [7] Jiawei Han, Jian Pei, Yiwen Yin, *Mining frequent patterns without candidate generation*, ACM SIGMOD, May 2000.
- [8] Guimei Liu, Hongjun Lu, Yabo Xu, Jeffrey Xu Yu: *Ascending Frequency Ordered Prefix-tree: Efficient Mining of Frequent Patterns*. DASFAA 2003: 65-72.
- [9] Jian Pei, Jiawei Han, *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, DMKD, May 2000.
- [10] Jianyong Wang, Jiawei Han, Jian Pei, *CLOSET+: searching for the best strategies for mining frequent closed itemsets*, ACM SIGKDD, Aug 2003.
- [11] Zijian Zheng, *Real World Performance of Association Rule Algorithms*. ACM SIGKDD, 2001.
- [12] Bart Goethals, Mohammed J. Zaki, *FIMI '03: Workshop on Frequent Itemset Mining Implementations*. FIMI'03, 2003.
- [13] Rakesh Agrawal, Tomasz Imieliński, Arun Swami, *Mining association rules between sets of items in large databases*, ACM SIGMOD, May 1993.
- [14] Rakesh Agrawal, Ramakrishnan Srikant, *Fast Algorithms for Mining Association Rules in Large Databases*, VLDB, September, 1994.
- [15] <http://www.ics.uci.edu/~mllearn/MLRepository.html>.