Mining Association Rules with Weighted Items

C.H. Cai, Ada W.C. Fu, C.H. Cheng and W.W. Kwong

Department of Computer Science and Engineering
The Chinese University of Hong Kong
chcai,adafu,chcheng,wwkwong@cse.cuhk.edu.hk

Abstract

Discovery of association rules has been found useful in many applications. In previous work, all items in a basket database are treated uniformly. We generalize this to the case where items are given weights to reflect their importance to the user. The weights may correspond to special promotions on some products, or the profitability of different items. We can mine the weighted association rules with weights. The downward closure property of the support measure in the unweighted case no longer exist and previous algorithms cannot be applied. In this paper, two new algorithms will be introduced to handle this problem. In these algorithms we make use of a metric called the k-support bound in the mining process. Experimental results show the efficiency of the algorithms for large databases.

Keywords: *data mining, association rules, basket data, support, confidence, weighted items.*

1 Introduction

Computers store large amounts of retailing transactions in a retailing business. Marketing managers will be interested in useful information that can be extracted from these large databases. As the amount of retailing information increases dramatically, there is a new challenge of finding interested information efficiently. One promising approach is the mining of association rules for basket databases, which has been investigated by [1], [2], [6], [7], [5], etc. Most of these works are focused on mining binary association rules. A binary association rule

Buys <Bread> $\Rightarrow$ Buys <Ham>
with $support = 0.6$, $confidence = 0.8$

says that the probability of buying bread and ham is 0.6, and the probability of buying ham is 0.8 given that a transaction contains bread. The interestingness of the rule depends on the number of occurrences of bread and ham.

In this paper, we introduce the notion of weighted items to represent the importance of individual items. In a retailing business, a marketing manager may want to mine the association rules with more emphasis on some particular products in mind, and less emphasis on other products. For example, some products may be under promotion and hence are more interesting, or some products are more profitable and hence rules concerning them are of greater values. This results in a generalized version of association rule mining problem, which we call weighted association rule mining.

For example, if the profit of the sofa is much higher than the bed, then the rule

Buys <Pillow> $\Rightarrow$ Buys <Sofa>

is more interesting than

Buys <Pillow> $\Rightarrow$ Buys <Bed>

When we compute the weighted support of the rule, we can consider both the support and the important ratio (weights) factors.

A simple attempt to solve the problem is to eliminate the entries of items with small weights. However, a rule for a heavy weighted item may also consist of low weighted items. For example, we may be promoting a product A, and find that it is affected by another product B, for which we have initially no interest. Hence the simple approach does not work in this case.

Another approach is adopting the existing fast algorithms for finding binary association rules, such as the Apriori Gen Algorithm [1]. Such algorithms depend on the downward closure property which governs that subsets of a large itemset are also large. However, it is not true for the weighted case in our definition, and the Apriori Algorithm cannot be applied.

In this paper, we propose new algorithms to mine weighted binary association rules. Two algorithms, MINWAL(O) and MINWAL(W) are designed for this purpose. Experimental result shows that these algorithms have reasonable performance for large databases and MINWAL(O) performs better than MINWAL(W) in most cases.

The paper is organized as follows. In Section 2, the definitions of mining weighted association rules will be introduced. In Sections 3 and 4 algorithms will be described for two different definitions of weighted support. Performance study will be reported in Section 5. Finally, Section 6 is a conclusion.

2 Weighted Association Rules

Similar to [1] and [5], we consider a database with a set of transactions $\mathcal{D}$, and a set of items $\mathcal{I} = \{i_1, i_2, ..., i_n\}$. Each transaction is a subset of $\mathcal{I}$, and is assigned a transaction identifier $\langle$TID$\rangle$.

Definition 1 *An* **association rule** *has the form of* $X \Rightarrow Y$, *where* $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, *and* $X \cap Y = \emptyset$.

We define the terms of support and the confidence as in [1].

Definition 2 *The* **support** *of the association rule* $X \Rightarrow Y$ *is the probability that* $X \cup Y$ *exists in a transaction in the database* $\mathcal{D}$.

Definition 3 *The* **confidence** *of the association rule* $X \Rightarrow Y$ *is the probability that* Y *exists given that a transaction contains* X, *i.e.,*

$$Pr(Y \backslash X) = \frac{Pr(X \cup Y)}{Pr(X)} \tag{1}$$

In large databases, the support of $X \Rightarrow Y$ is taken as the fraction of transactions that contain $X \cup Y$. The confidence of $X \Rightarrow Y$ is the number of transactions containing both X and Y divided by the number of transactions containing X.

Given a set of items $\mathcal{I} = \{i_1, i_2, \cdots, i_n\}$, we assign a **weight** w_j for each item i_j, with $0 \leq w_j \leq 1$, where $j = \{1, 2, \cdots, n\}$, to show the importance of the item.

According to Definition 2, we can define the weighted support for the weighted association rules.

Definition 4 *The* **weighted support** *of a rule* $X \Rightarrow Y$ *is*

$$\left(\sum_{i_j \in (X \cup Y)} w_j \right) (Support(X \cup Y)) \tag{2}$$

Similar to [1], a support threshold and a confidence threshold will be assigned to measure the strength of the association rules.

Bar code	Item	Total Profit	weights	....
1	Apple	100	0.1	....
2	Orange	300	0.3	....
3	Banana	400	0.4	....
4	Milk	800	0.8	....
5	Coca-cola	900	0.9	....

Table 1: Example of product database

TID	Bar codes	TID	Bar codes
1	1 2 4 5	2	1 4 5
3	2 4 5	4	1 2 4 5
5	1 3 5	6	2 4 5
7	2 3 4 5		

Table 2: Transaction database

Definition 5 *A* k*-itemset* X *is called a* **small itemset** *if the weighted support of such itemset is less than the minimum weighted support (wminsup) threshold, or*

$$\left(\sum_{i_j \in X} w_j \right) (Support(X)) < wminsup \tag{3}$$

Otherwise, X *is a* **large** k**-itemset**.

Definition 6 *An association rule* $X \Rightarrow Y$ *is called an* **interesting rule** *if* $X \cup Y$ *is a large itemset and the confidence, defined in definition (3), of the rule is greater than or equal to a minimum confidence threshold.*

Example 1

Suppose in a retailing store, a database is shown in Tables 1 and 2. Table 1 shows the information about the items in the retailing store. The information includes the bar code number of the items, the names of such item, the total profits of the items, the given weights, etc. Table 2 shows the transaction database. For each transaction, there will be a transaction identifier $\langle$TID$\rangle$, and the bar code numbers of the items. For simplicity, the bar code numbers will be represented in the form of natural numbers $\{1, \cdots, 5\}$.

Suppose that there are 5 items and totally 7 transactions in the transaction database. If the value of *wminsup* is 0.4, then {2, 5} will be a large itemset since

$$(0.3+0.9) \times \frac{5}{7} = 0.86 \geq 0.4$$

By the same argument, {4, 5} and {2, 4, 5} will be large itemsets. □

2.1 Reasoning behind the problem definition

In this problem, we want to get a balance between the two measures, which are weight and support. We have three parameters to consider in the weighted association rule: weights of items, support of itemsets and the confidence factor. We have chosen to compute a weighted support of an itemset which is the product of the total weight of items in the itemset and the support of the itemset.

Suppose we separate the supports and weights. We can only find itemsets having both sufficient support and weight. However, this may ignore some interesting knowledge. The semantics of weight is a measure of the importance of an itemset. If an itemset is very important, for example, it is under promotion, or it is highly profitable, then even if not many customers have bought it, it is still an interesting itemset to the user. On the other hand, if an itemset is not considered very important in terms of the weights, but it is very popular so that many transactions contain it, it is also an interesting itemset.

Another feasible alternative is to find itemsets that have either sufficient support or weight. However, this may not allow us to handle zero weight (no interest) items efficiently, no matter how high the support may be.

There is one possible problem with our definition. Even if each item has a small weight, when the number of items in an itemset is large, the total weight may be large. Depending on the application requirements this may or may not be desirable. It may be desirable if the user considers a rule with a number of items, which contribute to a sufficient profitability together, is interesting. It may not be desirable if a rule with many light weighted items should not be considered interesting. Here we also consider another problem definition in which the weighted support of an itemset is normalized by the size of the itemset. This will be discussed in Section 4. The semantics of the rules will be different, and it will depend on the need of each application to consider the applicability of this normalization.

3 Mining Weighted Association Rules

A new algorithm is needed to solve the mining of the weighted association rules. An efficient algorithm for mining binary association rules has been proposed in [1], called Apriori Gen. The reason why Apriori Gen works is because if an itemset is large, all the subsets of that itemset must be large. However, for the weighted case, the meaning of large itemset is modified to handle weighted support. It is not necessary true for all subsets of a large itemset being large. In Example 1, {2, 4} is a subset of the large itemset {2, 4, 5}, but it is not a large itemset.

3.1 k-Support Bound

Given a database with T transactions, we define the **support_count (SC)** of a large k-itemset X to be the transaction number containing X, and it must satisfy:

$$SC(X) \geq \frac{wminsup \times T}{\sum_{i_j \in X} w_j} \tag{4}$$

Let $\mathcal{I}$ be the set of all items. Suppose that Y is a q-itemset, where $q < k$. In the set of the remaining items $(\mathcal{I} - Y)$, let the items with the $(k-q)$ greatest weights $i_{r_1}, i_{r_2}, ..., i_{r_{k-q}}$. We can say the maximum possible weight for any k-itemset containing Y is

$$W(Y,k) = \sum_{i_j \in Y} w_j + \sum_{j=1}^{k-q} w_{r_j} \tag{5}$$

in which the first sum is the sum of the weights for the q-itemset Y, and the second sum is the sum of the $(k-q)$ maximum remaining weights.

From Inequality (4), the minimum count for a large k-itemset containing Y is given by

$$B(Y,k) = \left\lceil \frac{wminsup \times T}{W(Y,k)} \right\rceil \tag{6}$$

We called this the **k-support bound of Y**. We take an upper bound of the value since the function $B(Y,k)$ is an integer.

Example 2

Refer to Tables 1 and 2, the 3-support bound for the itemset {2, 4} is

$$\left\lceil \frac{0.4 \times 7}{(0.3 + 0.8) + 0.9} \right\rceil = 2 \tag{7}$$

It means if the itemset {2, 4} is the subset of any large 3-itemsets, the count of the itemset {2, 4} must be greater than or equal to 2. □

The algorithm for mining weighted association rules can be established according to the above properties of the k-support bound for all possible k-itemsets.

3.2 Algorithm for Mining Weighted Association Rules

An algorithm for mining weighted association rules has the following inputs and output.

Inputs: A database $\mathcal{D}$ with the transactions $\mathcal{T}$, two threshold values $wminsup$ and $minconf$, weights of the items w_i, with ascending order, total number of transactions and the total number of the items.

Output: A list of interesting rules.

Notations:

$\mathcal{D}$	The database
w	the set of item weights
L_k	set of large k-itemsets
C_k	set of k-itemsets which may be k-subsets of large j-itemsets for $j \geq k$
$SC(X)$	no. of transactions containing itemset X
$wminsup$	weighted support threshold
$minconf$	confidence threshold

Algorithm 1 MINWAL(O)

Main Algorithm ($wminsup, minconf$, $\mathcal{D}$, w)
    size=**Search**($\mathcal{D}$);
    L=$\emptyset$;
    for (i=1;i$\leq$size;i++)
        C_i=L_i=$\emptyset$;
    for each transaction do
        (SC,C_1)=**Counting**($\mathcal{D}$,w);
    k=1;
    while ($|C_k| \geq k$)
        k++;
        C_k=**Join** (C_{k-1});
        C_k=**Prune** (C_k);
        (C_k,L_k)=**Checking** (C_k,$\mathcal{D}$);
        L= L $\bigcup L_k$;
    Rules (SC, L);
ends;

The subroutines are outlined as follows:

1. **Search**($\mathcal{D}$): The subroutine accepts the database, finds out the maximum size of the large itemset in that transaction database $\mathcal{D}$, and returns the maximum size.

2. **Counting**($\mathcal{D}$,w): This subroutine cumulates the support counts of the 1-itemsets. The k-support bounds of each 1-itemset will be calculated, and the 1-itemsets with support counts greater than any of the k-support bounds will be kept in C_1.

3. **Join** (C_{k-1}): The Join step generates C_k from C_{k-1} as in [1]. For example, if we have {1, 2, 3} and {1, 2, 4} in C_{k-1}, {1, 2, 3, 4} will be generated in C_k.

4. **Prune** (C_k): During the prune step, the itemset will be pruned in either of the following cases :

   (a) A subset of the candidate itemset in C_k does not exist in C_{k-1}.

   (b) Estimate an upper bound on the support_count (SC) of the joined itemset X, which is the minimum support_count among the k different $(k-1)$-subsets of X in C_{k-1}. If the estimated upper bound on the support_count shows that the itemset X cannot be a subset of any large itemset in the coming passes (from the calculation of k-support bounds for all itemsets), that itemset will be pruned.

5. **Checking** (C_k,$\mathcal{D}$): The checking procedure scans the transaction database, updating the counts of all candidate itemsets in C_k. By the similar method in the prune step, prune the candidate itemsets for those not satisfying the support_count bounds for all possible large itemsets. The remaining candidate itemsets will be kept in C_k. At the same time, the large itemsets L_k will be generated from C_k by checking the exact weighted support of the itemsets.

6. **Rules** ($Support\_count, L$): Find the rules from the large itemsets L, similar to [1].

The framework of our proposed algorithm for mining weighted association rules is similar to the Apriori Gen Algorithm [1], but the detailed steps contain some significant differences. To begin with, we also generate large itemsets with increasing sizes. However, since the subset of a large itemset may not be large, we cannot generate candidate k-itemsets simply from the large $(k-1)$-itemsets as in Apriori Gen. We shall find a way to keep the k-itemsets which may generate large j-itemsets, for $j \leq k$, in the following passes. In order to extract such k-itemsets from the database, we use the j-support bound values. During the operation, the j-support bounds will be calculated for all the candidate k-itemsets, where j is any number between k and the maximum possible size of the large itemset. If the count of the existing k-itemset is less than all of the j-support bounds, we can say that it cannot be the subset of any large itemsets in the coming passes, and it can be pruned. The k-itemset, which may contribute to (be subsets of) future large itemsets, will be kept in C_k.

Example 3

From the Tables 1 and 2, we will show how the large itemsets are generated from the transaction database.

Suppose the *wminsup* (weighted minimum support threshold) is 1.

1. During the search process, the algorithm will scan the size of each transaction only, and returns the maximum possible size of the large itemsets, which is 4 in this example.

2. **Pass I** (k=1, where k is the size of the itemset)

   During the counting step, the transaction database will be scanned once, similar in Pass I of the Apriori Gen [1]. The counts of the items (1-itemsets) will be found during this stage. For each item, we can calculate the support bounds for all coming passes from the information (item supports_count and weights).

   For this example, the counts of the items $\{1,2,3,4,5\}$ are $\{4,5,1,6,7\}$ respectively. Let us denote the 1-itemset containing item 1 by I_a. The k-support bounds of the k-itemsets containing I_a are given by $B(I_a, k)$:

   $B(I_a, 2) = \left\lceil \frac{1\times 7}{0.1+0.9} \right\rceil = 8.$

   $B(I_a, 3) = \left\lceil \frac{1\times 7}{0.1+(0.8+0.9)} \right\rceil = 4.$

   $B(I_a, 4) = \left\lceil \frac{1\times 7}{0.1+(0.4+0.8+0.9)} \right\rceil = 4.$

   The k-support bounds of the other items are calculated as similar as above.

   The count of the item 1 is 4, which implies that it may be the subset of the large 3 or 4-itemsets. We should keep item 1 in C_1. By similar argument, items 2, 4 and 5 will be kept in C_1. Item 3 will be pruned because none of the k-support bounds for item 3 is less than or equal to the count of item 3. Therefore, C_1 will become $\{\{1\},\{2\},\{4\},\{5\}\}$. By similar method, all the candidate and large itemsets will be generated as the following.

   Pass II (k=2)

   During the join step, the algorithm will generate the following potentially large itemsets:

   $\{1, 2\},\{1, 4\},\{1, 5\},\{2, 4\},\{2, 5\},\{4, 5\}$

   During the prune step, the estimated upper bounds for the above corresponding itemsets will be (4,4,4,5,5,6) respectively. All the support bounds are calculated as above Pass I.

   After calculating all the support bounds, all the itemsets in C_2 will remain, as all of them may be large in the coming passes.

   During the checking step, the updated counts for the itemsets will be (2,3,4,5,5,6) respectively. From the support bounds calculated in the prune step, $\{1, 2\}$ cannot contribute to any large itemsets in the current or future passes, and it will be pruned in this stage.

   After calculating the weighted support in the remaining candidate set

   $$C_2 = \{\{1,4\},\{1,5\},\{2,4\},\{2,5\},\{4,5\}\}$$

   we found that the large itemset is $\{4, 5\}$, and we put $\{4, 5\}$ into L_2. The itemsets in C_2 will be used in next pass.

   We will use the above method in the remaining pass.

3. Rule generation from the $L = L \cup L_k$ is exactly the same as the Apriori Gen [1]. □

4 Mining Normalized Weighted association rules

In this section, we focus on the mining of weighted association rules for which the weight of an itemset is normalized by the size of the itemset. We can still apply the previous algorithm MINWAL(O) in Section 3 for this case simply, with a modification of the definitions of large itemsets and k-support bound (see below). However, we will design another new algorithm, and we shall present this, called MINWAL(W), in Section 4.1.

Definition 7 *The* **normalized weighted support** *of a rule $X \Rightarrow Y$ is given by*

$$\frac{1}{k}\left(\sum_{i_j\in(X\cup Y)} w_j\right)(Support(X\cup Y)) \qquad (8)$$

where k = size of the itemset $(X \cup Y)$

Definition 8 *A k-itemset X is called a* **small itemset** *if the normalized weighted support of such an itemset is less than the minimum weighted support (wminsup), or*

$$\left(\frac{\sum_{i_j\in(X)} w_j}{k}\right) Support(X) < wminsup \qquad (9)$$

Otherwise, it is a **large** *k-itemset.*

We can define the k-support bound for the normalized case.

Definition 9 *The minimum support_count for a large k-itemset which contains Y is called the* **k-support bound** *of the itemset Y with normalized weight and it is given by*

$$B(Y,k) = \left\lceil k \frac{wminsup}{W(Y,k)} \times T \right\rceil \tag{10}$$

4.1 Another approach for normalized weighted case

In the following, we will design an algorithm which generate large itemsets in an iterative manner as in Apriori-Gen and also effectively pruning candidate itemsets in each iteration. Although the closure property of "the subset of a large itemset must be large" still does not hold in the normalized weighted case, we can find the closure property in a different manner.

Definition 10 *For an itemset $X = \{x_1 \cdots x_n\}$, let the smallest weight of the items be w_i. An itemset $Y = X \cup Z$, where the items in Z have weights all less than w_i, is called a* **low-order superset** *of X.*

Definition 11 *An itemset $Y \subset X$, such that each item in Y has a weight greater than or equal to each item in $X - Y$, is called a* **high-order subset** *of X.*

Lemma 1 *If an itemset Y is large, then any high-order subset of Y must also be large.*

Proof Let X be a high-order subset of Y. The average weight of X is greater than or equal to the average weight of Y. The support of X is also greater than or equal to that of Y. Hence the weighted support of X is greater than or equal to that of Y. If Y is large, then X will be also large.

For example, if weights of items 1,2,4,5 are in ascending order, and $\{1, 2, 4, 5\}$ is a large 4-itemset, then itemsets $\{5\}$, $\{4, 5\}$ and $\{2, 4, 5\}$ must also be large.

Lemma 2 *A large $(k+1)$-itemset X must be a low-order superset of some large k-itemset Y.*

Proof: If X is large, then from Lemma 1, any high-order subset of X must also be large. Let x be the item in X with the lowest weight. Then $Y = X - x$ is a high-order subset of X and it must be large. Hence X is a low-order superset of Y.

With the above findings, we can modify some steps in the previous algorithm. The modified algorithm is presented in the following.

4.2 Algorithm for Mining Normalized Weighted Association Rules

The inputs and output for an algorithm for mining normalized weighted association rules are the following.

Inputs: Same as Algorithm 1

Output: Interesting normalized weighted association rules.

Most of the notations are similar to that of Algorithm 1, except for the following:

Notations:

C_k	set of candidate k-itemsets.
$wminsup$	normalized weighted support threshold.

Algorithm 2 MINWAL(W)

Main Algorithm ($wminsup$, $minconf$, $\mathcal{D}$, w)
  size=**Search**($\mathcal{D}$);
  L=$\emptyset$;
  for (i=1;i$\leq$size;i++)
    C_i=L_i=$\emptyset$;
  for each transaction do
    (SC,C_1)=**Counting**($\mathcal{D}$,w);
  k=1;
  while ($C_k \neq \phi$)
    k++;
    C_k=**Join** (L_{k-1});
    C_k=**Prune** (C_k);
    (L_k)=**Checking** (C_k,$\mathcal{D}$);
    L=$L \bigcup L_k$;
  Rules (SC, L);
ends;

The subroutines of MINWAL(W) are outlined as follows:

1. **Search**($\mathcal{D}$): Same as in Algorithm 1.

2. **Counting**($\mathcal{D}$,w): Same as in Algorithm 1.

3. **Join** (L_{k-1}): The subroutines Join, Prune, and Checking generate L_k and C_k. The main job in the join step is to generate C_k. From Lemma 2, a candidate k-itemset must be a low-order superset of some large $(k-1)$-itemset. In this step, we join the large itemsets in L_{k-1} with one of the items that have lower weights to form a low-order superset. For example, if $w_1 \leq w_2 \leq \cdots \leq w_5$ are the weights of the item $\{1,2,3,4,5\}$, and $\{3, 5\}$ is a large itemset found in pass II, the join step will construct $\{1, 3, 5\}$ and $\{2, 3, 5\}$ itemsets only. Those joined itemset will be put in C_k. $\{3, 4, 5\}$ cannot be a large itemset because if it is, then $\{4, 5\}$ should be a large itemset.

4. **Prune** (C_k): During the prune step, a candidate k-itemset X will be pruned if all the j-support

bounds of the X, $j \leq k$, are greater than the smallest known support_count among the $(k-1)$-subsets of X, which is an estimation and an upper bound of the support_count of the k-itemset X.

5. **Checking** $(C_k, \mathcal{D})$: The checking procedure will be done similar to Algorithm 1, except that the remaining candidate itemset will be the set of large itemset L_k, and the next pass will be based on the L_k to generate the candidate sets.

6. **Rules** (SC, L): Same as in Algorithm 1.

The framework for our proposed algorithm for mining weighted association rules (with averaging of weights) is similar to Apriori Gen [1] and Algorithm 1, but with some major differences in the details. Although the large k-itemsets can be generated from large $(k-1)$-itemsets, but it is not true that all the subsets of a large itemset should be large. The modification is on the generation of candidate sets. In the Apriori Gen process [1], generating the candidate sets C_k is based on the large itemsets L_{k-1}. Here we also generate C_k from L_{k-1}. However, we would not consider k-itemsets where all $(k-1)$-subsets are in L_{k-1}, since not all $(k-1)$-subsets of a large k-itemset should be large. Instead, we shall consider the low-order supersets of the itemsets in L_{k-1} (from Lemma 2).

During the prune step, we check the estimated weighted support of the itemsets. The difference between this and the Apriori Gen is not to check the subsets of the itemsets being large. We use the support bound values instead.

Example 4

Refer to Tables 1 and 2, we will show how the large itemsets are generated from the transaction database. Suppose that the *wminsup* (weighted minimum support) is 0.45. Similar to Algorithm 1, we find the maximum possible size of the large itemsets, which is 4.

Pass I (k=1, where k is the size of the itemset)

The support_counts (SC) of the 1-itemset {1,2,3,4,5} will be {4,5,1,6,7}. For these counts, for example, the weighted support of item {1} will be

$$0.1 \times \frac{4}{7} = 0.06$$

By the closure property, the high-order subset of a large k-itemset should be large. Therefore, items {3}, {2} and {1} will be pruned. Hence $C_1 = \{\{4\}, \{5\}\}$.

Pass II (k=2)

During the join step, the algorithm will generate the following potentially large itemsets:

D	Number of transactions
T	Average size of the transactions
I	Ave. size of the max. potentially large itemsets
L	Number of maximal potentially large itemsets
$\mathcal{N}$	Number of items

Table 3: Parameters of the synthetic database

{5, 4},{5, 3},{5, 2},{5, 1},{4, 3},{4, 2},{4, 1}

During the prune step, the estimated weighted support of itemset {5, 4} will be $\frac{(0.8+0.9)}{2} \times \frac{6}{7} = 0.72$

Others will be calculated as the above.

During the checking step, the updated count for the itemset {5, 4} will be 6. From the calculation of the weighted support, {4, 5} is found to be a large itemset, and is put in L_2.

We will use the same method for the remaining passes, until no itemset found in candidate set. □

5 Performance Study

A performance study is carried out for the two algorithms, MINWAL(O) and MINWAL(W). A series of experiments are done to show the performance of these algorithms on an UltraSparc 1 machine with 256MB of main memory. The algorithms are written in C and the timing is measured by the cpu time calculated from the built-in timing functions of UNIX Shell. In order to control different parameters in the experimental setup, we use synthetic databases and weights in the experiment. The method of generating such synthetic data will be explained in Section 5.1.

In the following experiments, we use different transaction databases with the same set of item weights. The reason for using the same item weights is to keep this factor constant in order to compare the efficiency of the algorithms.

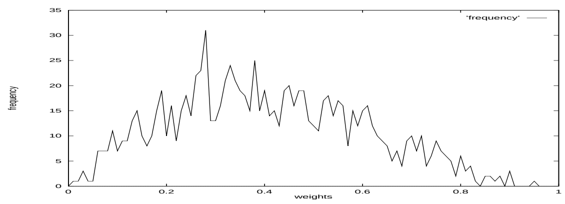

Figure 1: Distribution of Weights

5.1 Generation of Synthetic Data and Weights

During the experiment, we will use synthetic data to evaluate the performance. The data generation procedure is similar to [1].

Before the generation of the database, the parameters $D, L, \mathcal{N}$, T and I are set to be 100K, 2000, 1000, 5 and 2 respectively, with 2.4MB of database.

In the generation of the weights, we assume that the number of the low weight products will be much more than the high-weight products. We generate the weights according to an exponential distribution, and the result is shown in Figure 1.

5.2 Performance Evaluation

We study the effect of different values of *wminsup*, number of transactions and items, etc., on the processing time, for algorithms MINWAL(O) and MINWAL(W). We use the hash-tree data structures [1] in the following experiments.

5.2.1 Performance Evaluation of the two algorithms

The experiment will be done on the two algorithms (MINWAL(O) and MINWAL(W)) under two conditions: the unnormalized case (as describe in Section 2) and the normalized case (as discussed in Section 4). Since MINWAL(W) cannot be applied to the unnormalized case, we need to consider only three cases: MINWAL(O) for cases with and without normalization, and MINWAL(W) for the normalized case.

The test will be based on the synthetic databases. We use 5 values of thresholds (*wminsup*) for each test. We use $\{0.0016, 0.0017, 0.0018, 0.0019, 0.0020\}$ in the mining of unnormalized weighted association rules, while *wminsup* of $\{0.0006, 0.0007, 0.0008, 0.0009, 0.0010\}$ are used in the mining for the normalized weighted case. These threshold values are chosen to generate a reasonable number of large itemsets and rules. For simplicity we use the values of $\{1,2,3,4,5\}$ to represents these sets of threshold values in most of the figures.

Figure 2 shows the decreasing trend of the execution time when the support threshold increases. As the threshold increases, the candidate itemsets decreases. The execution time would decrease for the smaller resulting hash-tree because of the decreasing searching time.

In Figure 3, it is noted that the time needed for each pass for the MINWAL(O) is much less than the MINWAL(W), especially from pass 2 to pass 4. Furthermore, when comparing the two algorithms MINWAL(W) and MINWAL(O) in the normalized weighted case, it is noticed that the execution time of MINWAL(O) is much less than the algorithm MINWAL(W), especially for the cases with smaller thresh-

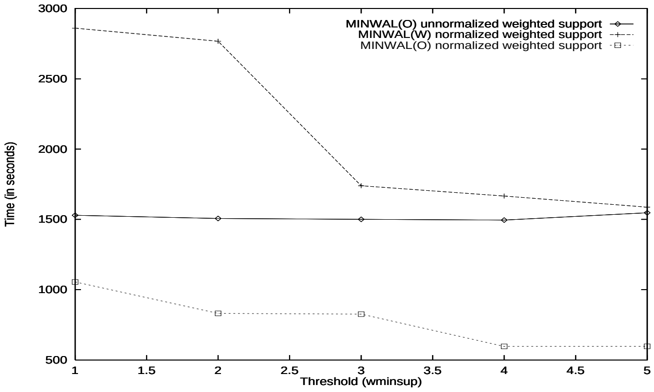

Figure 2: Overall execution time

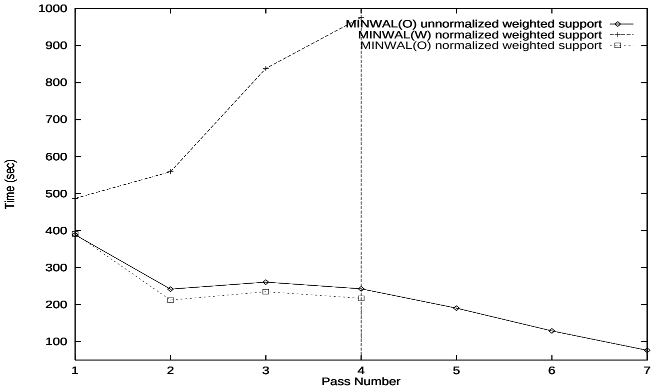

Figure 3: Execution time for each pass

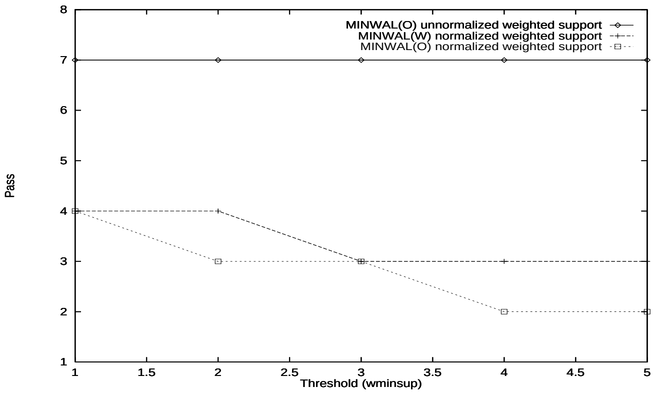

Figure 4: Passes needed

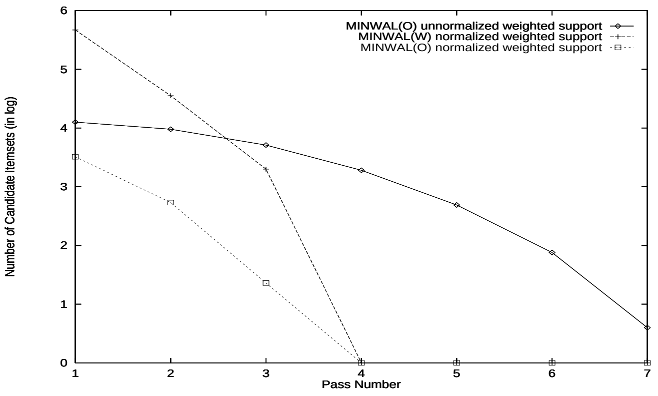

Figure 5: Size of the candidate itemsets

olds. It was because the number of the candidate itemsets needed to be checked in MINWAL(W) is much more than in MINWAL(O).

Figure 4 shows the number of passes needed in mining of the association rules. From the graph, the number of passes needed for mining the unnormalized weighted case is more than the normalized case.

The number of candidate itemsets mined is shown in Figure 5 , where we set $wminsup = 0.006$ for the normalized weighted case and $wminwup = 0.0016$ for the unnormalized case. The number of candidate itemsets generated by MINWAL(W) is greater than that of MINWAL(O) for the normalized weighted case.

Based on the all the above figures, we can see that MINWAL(O) is in general more efficient in the mining of normalized weighted association rules. This is because that the time needed to check the candidate itemsets is much less than MINWAL(W).

5.2.2 Scale-Up Experiment

In the following, we examine how the performance varies with the number of items and transactions. All other things being equal.

With $wminsup = 0.002$ for MINWAL(O) for the unnormalized case, and $wminsup = 0.001$ for other cases, the execution time increases with the number of items since the more the items, the larger the hash tree. As the hash tree grows directly with the items, the execution time in the hash tree searching would be increased, which is shown in Figure 6.

In Figure 7, we are interested in the relation between the number of transactions and the execution time. Keeping other things equal (1000 items and weights), we generate different transaction database

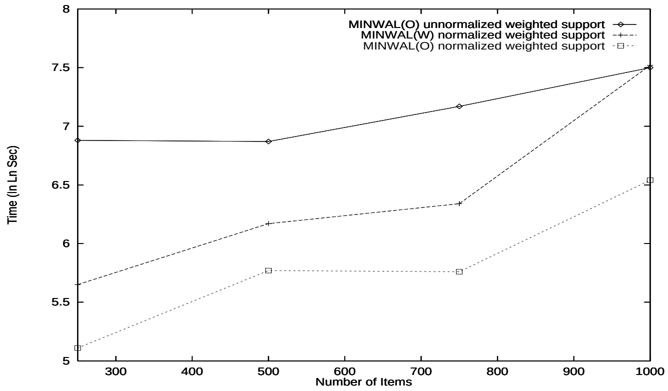

Figure 6: Number of items scale-up

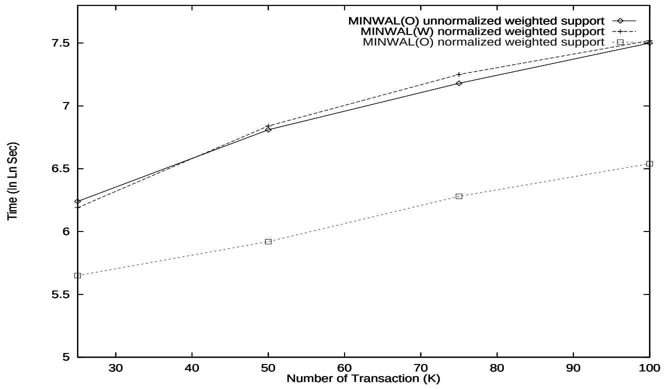

Figure 7: Number of transactions scale-up

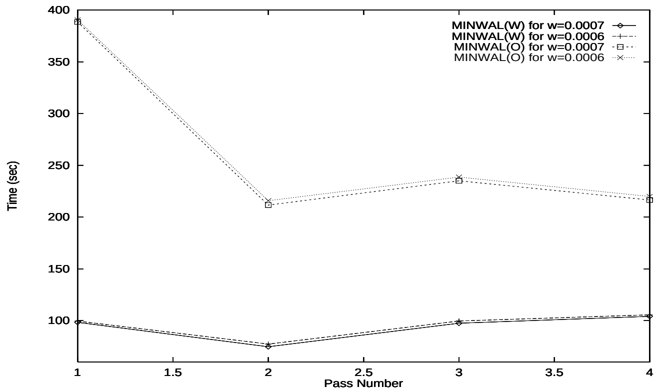

Figure 8: Special case with 0/1 weights for normalized case

with the same parameters but different in size, ranging from 25K to 100K. The values of *wminsup* are set as the above scale-up experiment. In this figure, the time is given in $ln(sec)$. From the figure, the execution time increases with the number of transactions linearly with ln scale, implying that the complexity of the algorithms is exponential in the number of transactions [1].

5.2.3 Experiment for special case

In this section, we are interested in the performance in the special case, which is the item weights equal to 0 or 1 only. In this case, we make the first 900 weights be 0 and the remaining weights be 1. Other things, including database and threshold, equal as above section. We carried out the experiment for the normalized weighted case to compare the two algorithms. There are two major findings:

1. The performance of the special case is much better than the general case where item weights follow a distribution between 0 and 1.

2. Contrary to the previous cases, MINWAL(W) performs better than MINWAL(O). From Figure 8, we notice that the time needed in MINWAL(W) is much less than the MINWAL(O), for all the thresholds. This is because in the joining step, the number of starting seed (candidate itemsets in C_1 to generate itemsets in C_2) is less than MINWAL(O) case. In this situation, the 0/1 weights give the advantage to MINWAL(W). During the first step, the algorithm MINWAL(W) will easily prune all the small itemsets with 0 weights, while MINWAL(O) will keep those small itemsets with 0 weights. As the starting seed is smaller in size, MINWAL(W) would perform well in this case.

6 Conclusion

We have proposed to study a new problem of mining weighted association rule. This is a generalization of the association rule mining problem. In this generalization, the items are assigned weights to reflect their importance to the user. The main difference between mining weighted association rules and the mining unweighted association rules is the downward closure property.

We proposed two different definition of weighted support: without normalization, and with normalization. We proposed new algorithms based on the *support bounds*: the algorithms MINWAL(O) and MINWAL(W). MINWAL(O) is applicable to both normalized and unnormalized cases, and MINWAL(W) is applicable to the normalized case only. The performance evaluation has been done on these two algorithms. We found that MINWAL(O) outperforms MINWAL(W) in most cases, but MINWAL(W) performs better for the special case with only 0/1 item weights.

So far we have only considered the mining of binary weighted association rules. Some of the researchers did the research for the problem of the quantitative assoication rules, such as [4], [3]. We may investigate the problem of quantitative association rules with weighted items, which is an interesting topic in the future.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pages 487–499, 1994.

[2] D. Cheung, V.T. Ng, A. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. In *IEEE Transactions on Knowledge and Data Engineering*, pages 1–23, 1996.

[3] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, an visualization. In *Proceedings of ACM SIGMOD*, pages 13–23, 1996.

[4] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Mining optimized association rules for numeric attributes. Technical Report 1623-14, IBM Tokyo Research Laboratory, 1996.

[5] J. Han, M. Kamber, and J. Chiang. Mining multidimensional association rules using data cubes. Technical report, Database Systems Research Laboratory, School of Science, Simon Fraser University, 1997.

[6] J.S. Park, M-S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of ACM SIGMOD*, pages 175–186, 1995.

[7] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 432–444, 1995.