SoC Technical Report, National University of Singapore, 1999. A shorter version of the paper appeared in SIGKDD-99.

Mining Association Rules with Multiple Minimum Supports

Bing Liu, Wynne Hsu, and Yiming Ma

School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119260
Tel: +65 874 6736
{liub, whsu, maym}@comp.nus.edu.sg

Abstract

Association rule mining is an important model in data mining. Its mining algorithms discover all item associations (or rules) in the data that satisfy the user-specified minimum support (minsup) and minimum confidence (minconf) constraints. Minsup controls the minimum number of data cases that a rule must cover. Minconf controls the predictive strength of the rule. Since only one minsup is used for the whole database, the model implicitly assumes that all items in the data are of the same nature and/or have similar frequencies in the data. This is, however, seldom the case in real-life applications. In many applications, some items appear very frequently in the data, while others rarely appear. If minsup is set too high, those rules that involve rare items will not be found. To find rules that involve both frequent and rare items, minsup has to be set very low. This, however, may cause combinatorial explosion because those frequent items will be associated with one another in all possible ways. This dilemma is called the *rare item problem*. This paper proposes a novel technique to solve this problem. The technique allows the user to specify multiple minimum supports to reflect the natures of the items and their varied frequencies in the database. In rule mining, different rules may need to satisfy different minimum supports depending on what items are in the rules. Experiment results on both synthetic data and real-life data show that the proposed technique is very effective.

1 Introduction

Association rules are an important class of regularities that exist in databases. Since it was first introduced in [2], the problem of mining associations has received a great deal of attention. The classic application is market basket analysis [2]. It analyzes how the items purchased by customers are associated. An example of an association rule is as follows,

cheese $\rightarrow$ *beer* [sup = 10%, conf = 80%]

This rule says that 10% of customers buy *cheese* and *beer* together, and those who buy *cheese* also buy *beer* 80% of time. There are many other applications. The basic model of association rules is as follows:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Let T be a set of transactions (the database), where each transaction t (a data case) is a set of items such that $t \subseteq I$. An *association rule* is an implication of the form, $X \to Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \to Y$ holds in the transaction set T with *confidence c* if c% of transactions in T that support X also support Y. The rule has *support s* in T if s% of the transactions in T contains $X \cup Y$.

Given a set of transactions T (the database), the problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*).

Minsup controls the number of transactions that a rule must cover. Minconf controls the predictive strength of the rule. The two values are set according to applications. A typical association rule mining algorithm works in two steps,

1. generate all *large* itemsets that satisfy minsup.
2. generate all association rules that satisfy minconf using the large itemsets.

An itemset is simply a set of items. A large itemset is an itemset that has transaction support above minsup.

Association rule mining has been studied extensively in the past. Existing studies have been focused on: improving the performance of mining algorithms [e.g., 2, 4, 7, 16, 19, 21, 23, 24, 28]; mining generalized or multi-level rules [11, 26]; devising parallel algorithms [e.g., 3]; mining quantitative association rules [e.g., 10, 23, 14]; mining constrained association rules [27, 21], etc. The model used in these studies, however, has always been the same, i.e., finding all rules that satisfy the user-specified minimum support and minimum confidence constraints.

Theoretically, both the computational complexity and the number of rules produced grow exponentially for association rule mining. Minsup holds the key to making association rule mining practical. Minsup is used to prune the search space and to limit the number of rules generated. However, using only a single minsup implicitly makes the assumption that all items in the data are of the same nature (to be explained below) and/or have similar frequencies [15] in the database. This is often not the case in real-life applications. In many applications, some items appear very frequently in the data, while others rarely appear. If the frequencies of items vary a great deal, we will encounter two problems:

1. If minsup is set too high, we will not find those rules that involve infrequent items (or *rare items*) in the data.
2. In order to find rules that involve both frequent items and rare items, we have to set minsup very low. However, this may cause combinatorial explosion, i.e., producing too many associations, because those frequent items will be associated with one another in all possible ways and many of them are meaningless.

   Example 1: In a supermarket transaction data, in order to find rules involving those infrequently purchased items such as *food processor* and *cooking pan* (they generate more profit per item), we need to set the minsup to very low (say, 0.5%). We may find the following useful rule:

   foodProcessor → *cookingPan* [sup = 0.5%, conf = 60%]

   However, this low minsup may also cause the following meaningless rule to be found:

   bread, *cheese*, *milk* → *beer* [sup = 0.5%, conf = 60%].

   Knowing that 0.5% of the customers buy the 4 items together is useless because all these items are frequently purchased in a supermarket. For this rule to be useful, the support needs to be much higher.

This dilemma is called the *rare item problem* [15]. When confronted with this problem in applications, researchers either split the data into a few blocks according to the frequencies of the items and then mine association rules in each block with a different minsup [13], or group a number of related rare items together into an abstract item so that this abstract item is more frequent [11, 13]. The first approach is not satisfactory because rules that involve items across different blocks are difficult to find. Similarly, the second approach is unable to find rules involving individual rare items and the more frequent items. Furthermore, in many applications it may not even be meaningful to group rare items together [13]. Clearly, both approaches are ad hoc and “approximate” [13].

This paper argues that using a single minsup for the whole database is inadequate because it cannot capture the inherent natures and/or frequency differences of the items in the database. By the natures of items we mean that some items, by nature, appear more frequently than others. For example, in a supermarket, people buy *food processor* and *cooking pan* much less frequently than they buy *bread* and *milk*. In general, those durable and/or expensive goods are bought less frequently, but each of them generates more profit. It is thus important to capture those rules involving less frequent items. But we must do so without allowing frequent items to produce too many meaningless rules with very low supports (causing combinatorial explosion).

In this paper, we extend the existing association rule model to allow the user to specify multiple minimum supports to reflect the different natures and/or frequencies of items. Specifically, the user can specify a different *minimum item support* for each item. Thus, different rules may need to satisfy different minimum supports depending on what items are in the rules. This new model enables us to achieve our objective of producing rare item rules without causing frequent items to generate too many meaningless rules. An efficient algorithm for discovering association rules in the model is also presented. Experiment results on both synthetic data and real-life data show that the proposed technique is very effective.

2 The Extended Model

In our extended model, the definition of association rules remains the same. However, the definition of minimum support is changed.

In the extended model, the minimum support of a rule is expressed in terms of *minimum item supports* (MIS) of the items that appear in the rule. That is, each item in the database can have a minimum item support specified by the user. By providing different MIS values for different items, the user effectively expresses different support requirements for different rules.

Let MIS(i) denote the MIS value of item i. The *minimum support of a rule R* is the lowest MIS value among the items in the rule. That is, a rule R,

$$a_1, a_2, \ldots, a_k \rightarrow a_{k+1}, \ldots, a_r$$

where $a_j \in I$, satisfies its minimum support if the rule's actual support in the data is greater than or equal to:

$$\min(\text{MIS}(a_1), \text{MIS}(a_2), \ldots, \text{MIS}(a_r)).$$

Minimum item supports thus enable us to achieve the goal of having higher minimum supports for rules that only involve frequent items, and having lower minimum supports for rules that involve less frequent items.

Example 2: Consider the following items in a database,

bread, *shoes*, *clothes*.

The user-specified minimum item supports are as follows:

MIS(*bread*) = 2%

MIS(*shoes*) = 0.1%

MIS(*clothes*) = 0.2%

The following rule does not satisfy its minimum support:

clothes $\rightarrow$ *bread* [sup = 0.15%, conf = 70%]

because min(MIS(*bread*), MIS(*clothes*)) = 0.2%. The following rule satisfies its minimum support:

clothes $\rightarrow$ *shoes* [sup = 0.15%, conf = 70%]

because min(MIS(*clothes*), MIS(*shoes*)) = 0.1%.

While a single minsup is inadequate for real-life applications, we also realize that there are some deficiencies with the minconf measure of the existing model, which are briefly discussed below (similar ideas are also discussed in [5]). However, it is not the focus of this paper.

Although minconf is well-suited to the market basket analysis, where it ensures that a rule is true often enough to make a marketing campaign effective. However, in many other basket data analysis problems, this may not be adequate. For example, in a database, it is known that 80% of the customers buy *coffee*, and 5% buy *tea*, and 10% buy *creamer*. Then, given the rule below is of limited use:

Rule 1: *creamer* $\rightarrow$ *coffee* [sup = 6%, conf = 80%]

because buying *creamer* and *coffee* are (statistically) independent if we perform a statistic test, e.g., Chi-square test [18]. That is, whether one buys *coffee* or not is not correlated to one buying *creamer* because 80% of the customers buy *coffee* anyway. The following rule, on the other hand, is quite significant:

Rule 2: *creamer* $\rightarrow$ *tea* [sup = 4%, conf = 70%]

because in general only 5% of the customers buy *tea* in the shop. But when they buy *creamer*, the chance that they also buy *tea* increases dramatically. If we set minconf to 75%, then Rule 2 will not be generated, but only Rule 1 (which is useless). Clearly, in this case, using a single minconf is inadequate. The problem is that minconf does not consider the existing characteristics of the data iteself. In this aspect, statistic correlation test (e.g., Chi-square test) is more appropriate. Thus, we could use a statistical test rather than minconf as the measure for generating interesting rules. Clearly, there can be many other alternative measures for specific applications. We will not discuss this issue further in this paper as it is relatively easy to deal with and can be application specific. Below, we present the algorithm for mining large itemsets with multiple minimum item supports.

3 Mining Large Itemsets with Multiple Minimum Item Supports

3.1 Downward closure property

As mentioned, existing algorithms for mining association rules typically decompose the task into two subtasks.

1. Find all large itemsets that satisfy minsup.
2. Generate association rules that satisfy minconf using the large itemsets.

Almost all research in association rule mining algorithms focused on the first task since it is usually more computationally expensive. Also, the second task does not lend itself as well to smart algorithms because confidence does not possess closure property. Support, on the other hand, is *downward closed*. If a set of items satisfies the minsup, then all its subsets also satisfy the minsup. Alternatively, if a set of items does not satisfy the minsup, then any of its superset will not satisfy the minsup. Downward closure property holds the key to pruning in all mining algorithms.

Efficient algorithms for finding large itemsets are based on level-wise search [4]. Let k-itemset denote an itemset with k items. At level 1, all large 1-itemsets are generated. At level 2, all large 2-itemsets are generated and so on. If an itemset is not large at level k-1, it is discarded because any addition of items to the set cannot be large (downward closure property). All the potentially large itemsets at level k are generated from large itemsets at level k-1.

However, in the proposed model, if we use an existing algorithm to find all large itemsets, the downward closure property no longer holds as shown in the following example.

Example 3: Consider four items 1, 2, 3 and 4 in a database. Their minimum item supports are given as follows:

MIS(1) = 10% MIS(2) = 20%

MIS(3) = 5% MIS(4) = 6%

If we find that itemset {1, 2} has 9% of support in the database at level 2, then it does not satisfy either MIS(1) or MIS(2). Using an existing algorithm, this itemset is discarded since it is not large. Then, the potentially large itemsets {1, 2, 3} and {1, 2, 4} will not be generated for level 3. Clearly, itemsets {1, 2, 3} and {1, 2, 4} may be large because MIS(3) is only 5% and MIS(4) is 6%. It is thus wrong to discard {1, 2}. But if we do not discard {1, 2}, the downward closure property is lost.

Below, we propose an algorithm to solve the problem. The essential idea is to sort the items according to their MIS values in ascending order to avoid the problem.

3.2 The algorithm

The proposed algorithm generalizes the well-known Apriori algorithm for finding large itemsets given in [4]. We call the new algorithm, MSapriori. When there is only one MIS value (for all items), it reduces to the Apriori algorithm.

Like algorithm Apriori, our algorithm is also based on level-wise search. It generates all large itemsets by making multiple passes over the data. In the first pass, it counts the supports of individual items and determines whether they are large. In each subsequent pass, it starts with the seed set of itemsets found to be large in the previous pass. It uses this seed set to generate new possibly large itemsets, called *candidate itemsets*. The actual supports for these candidate *itemsets*. are computed during the pass over the data. At the end of the pass, it determines which of the candidate *itemsets* are actually large. However, there is an important exception in the second pass as we will see later.

A key operation in the proposed algorithm (which is not in algorithm Apriori) is the sorting of the items in I in ascending order of their MIS values. This ordering is used in all subsequent operations of the algorithm. The items in each itemset also follow this order. For example, in Example 3 of the four items 1, 2, 3 and 4, and their given MIS values, the items are sorted as follows: 3, 4, 1, 2. This ordering helps to solve the problem identified in Section 3.1.

Let L_k denote the set of large k-itemsets. Each itemset c is of the following form:

$<c[1], c[2], \ldots, c[k]>$

which consists of items, $c[1]$, $c[2]$, …, $c[k]$, where $\mathrm{MIS}(c[1]) \leq \mathrm{MIS}(c[2]) \leq \ldots \leq \mathrm{MIS}(c[k])$. The algorithm is given below:

Algorithm MSapriori

```
M = sort(I, MS);  /* according to MIS(i)'s stored in MS */
F = init-pass(M, T);   /* make the first pass over T */
L1 = {<f> | f ∈ F, f.count ≥ MIS(f)};
for (k = 2; Lk-1 ≠ ∅; k++) do
    if k = 2 then
        C2 = level2-candidate-gen (F, MS)
    else  Ck = candidate-gen (Lk-1);
    end
    for each transaction t ∈ T do
         Ct = subset(Ck, t);
         for each candidate c ∈ Ct do
             c.count++;
    end
    Lk = {c ∈ Ck | c.count ≥ MIS(c[1])}
end
Answer = ∪k Lk;
```

Line 1 performs the sorting operation on I according to the MIS value of each item (stored in MS). Line 2 makes the first pass over the data using the function init-pass, which takes two arguments, the database T and the sorted items M to produce the seeds for generating the set of candidate large itemsets of length 2, i.e., C_2. init-pass has two steps:

1. It makes a pass over the data to record the actual support count of each item in M.
2. It then follows the sorted order to find the first item i in M that meets MIS(i). i is inserted into F. For each subsequent item j in M after i, if j.count $\geq$ MIS(i) then j is also inserted into F (j.count means the count of item j).

Note that for simplicity, we use the term's support and count interchangeably (actually, support = count/$|T|$, where $|T|$ is the size of the database T).

Example 4: Let us follow Example 3 and the given MIS values of the four items. Assume our database has 100 transactions (not limited to the four items). After making one pass over the data, we obtain the following support counts: 3.count = 6, 4.count = 3, 1.count = 9 and 2.count = 25. Then, (in sorted order)

F = {3, 1, 2}, and L_1 = {<3>, <2>}

Item 4 is not in F because 4.count < MIS(3) (= 5%), and <1> is not in L_1 because 1.count < MIS(1) (=10%).

Large 1-itemsets (L_1) are obtained from F (line 3). It is easy to show that all large 1-itemsets are in L_1.

For each subsequent pass, say pass k, the algorithm performs 3 major operations. First, the large itemsets in L_{k-1} found in the (k-1)*th* pass are used to generate the candidate itemsets C_k using the *condidate-gen* function (line 7). It then scans the database and updates various support counts of the candidates in C_k (line 9-13). After that, those new large itemsets are identified to form L_k (line 14).

However, there is a special case, i.e., when k = 2 (line 5), for which the candidate itemsets generation function is different. Both candidate generation functions level2-candidate-gen and candidate-gen are described below.

3.3 Candidate generation

level2-candidate-gen takes as arguments F (not L_1) and MS, and returns a superset of the set of all large 2-itemsets. The algorithm is as follows:

```
for each item f in F in the same order do
    if f.count ≥ MIS(f) then
        for each item h in F that is after f do
            if h.count ≥ MIS(f) then
                insert <f, h> into C2
```

Example 5: Let us continue with Example 4. We obtain,

$$C_2 = \{<3, 1>, <3, 2>\}$$

<1, 2> is not a candidate 2-itemset because the support count of item 1 is only 9 (or 9%), which is less than MIS(1) (= 10%). Hence, <1, 2> cannot be large.

Note that we must use F rather than L_1 because L_1 does not contain those items that may satisfy the MIS of an earlier item (in the sorted order) but not the MIS of itself (see the difference between F and L_1 in Example 4). Using F, the problem discussed in Section 3.1 is solved for C_2.

Correctness of level2-candidate-gen: We need to show that $C_2 \supseteq L_2$. Clearly, any item of a large 2-itemset must at least have the minimum item support of its first item $c[1]$ (i.e., MIS($c[1]$)). Hence, if we combine each item f in F, where f.count ≥ MIS(f), with each possible item h after f and h.count ≥ MIS(f), we would obtain a superset of the itemsets in L_2. level2-candidate-gen does exactly that.

Let us now present the candidate-gen function. It performs a similar task as the apriori-gen function in Apriori algorithm. candidate-gen takes as argument L_{k-1} ($k > 2$) the set of all large (k-1)-itemsets, and returns a superset of the set of all large k-itemsets. It has two steps, the join step and the prune step. The join step is the same as that in apriori-gen function. The prune step is, however, different. The join step is given below. It joins L_{k-1} with L_{k-1}:

```
insert into Ck
select p.item1, p.item2, …, p.itemk-1, q.itemk-1
from Lk-1 p, Lk–1 q
where p.item1 = q.item1, …, p.itemk-2 = q.itemk-2,
      p.itemk-1< q.itemk-1
```

Basically, it joins any two itemsets in L_{k-1} whose first k-2 items are the same, but the last items are different.

After the join step, there may still be candidate itemsets in C_k that are impossible to be large. The prune step removes these itemsets. This step is given below (next page):

```
1   for each itemset c ∈ C_k do
2       for each (k-1)-subset s of c do
3           if (c[1] ∈ s) or (MIS(c[2]) = MIS(c[1])) then
4               if (s ∉ L_{k-1}) then
5                   delete c from C_k;
```

It checks each itemset c in C_k (line 1) to see whether it can be deleted by finding its (k-1)-subsets in L_{k-1}. For each (k-1)-subset s in c (there are k (k-1)-subsets altogether), if s is not in L_{k-1}, c can be deleted. However, there is an exception, which is when s does not include $c[1]$ (there is only one such s). This means that the first item of c, which also has the lowest MIS value, is not in s. Then, even if s is not in L_{k-1}, we cannot delete c because we cannot be sure that s does not satisfy MIS(c[1]), although we know that it does not satisfy MIS(c[2]), unless MIS(c[2]) = MIS(c[1]) (line 3).

Example 6: Let L_3 be {<1, 2, 3>, <1, 2, 5>, <1, 3, 4>, <1, 3, 5>, <1, 4, 5>, <1, 4, 6>, <2, 3, 5>}. Items in each itemset are in the sorted order. After the join step, C_4 will be

{<1, 2, 3, 5>, <1, 3, 4, 5>, <1, 4, 5, 6>}

The prune step deletes the itemset <1, 4, 5, 6> because the itemset <1, 5, 6> is not in L_{k-1}. We are then left with C_4 = {<1, 2, 3, 5>, <1, 3, 4, 5>}. <1, 3, 4, 5> is not deleted although <3, 4, 5> is not in L_3 because the minimum support for <3, 4, 5> is MIS(3), which may be higher than MIS(1). Although <3, 4, 5> does not satisfy MIS(3), we cannot be sure that it does not satisfy MIS(1) either. However, if we know MIS(3) = MIS(1), then <1, 3, 4, 5> can also be deleted.

Correctness of candidate-gen: Again, we need to show that $C_k \supseteq L_k$. Clearly, any subset of a large k-itemset P must at least have the minimum item support of P's first item c[1] (i.e., MIS(c[1])). Hence, if we extend each itemset in L_{k-1} with all possible items and then delete those whose (k-1)-subsets do not meet MIS(c[1]), we would be left with a superset of the itemsets in L_k.

The join step is equivalent to extending L_{k-1} with each item in the database and then deleting those itemsets for which the (k-1)-itemset obtained by deleting the (k-1)th item is not in L_{k-1}. The condition $p.item_{k-1} < q.item_{k-1}$ simply ensures that no duplicates are generated. Thus, after the join step, $C_k \supseteq L_k$. By similar reasoning, the prune step, where we delete from C_k each itemset c whose (k-1)-subsets containing c[1] (which has the lowest MIS value among the items in c) are not in L_{k-1}, also does not delete any itemset that could be in L_k. When MIS(c[2]) = MIS(c[1]), even if s does not contain c[1] but $s \notin L_{k-1}$, deleting c does not remove any itemset that could be in L_k.

The problem discussed in Section 3.1 is solved for C_k (k > 2) because due to the sorting we do not need to extend a large (k-1)-itemset with any item that has a lower MIS value, but only an item with a higher (or equal) MIS value.

3.4 Subset function

The subset function checks to see which itemsets in C_k are in transaction t. Itemsets in C_k are stored in a tree similar to that in [4]. Each tree node contains an item (except the root node). By depth-first traversing of the tree against t, we can find if an itemset is in t. At each node, we check whether the item in the node is in t. If so, we go down the tree. If not, we backtrack. When a leaf node is reached, we know that the itemset represented by the path is in t.

This method for finding C_t is different from that in [4]. The method in [4] uses each item in t to traverse the tree. In our extended model, this, however, requires the items in each transaction t to be sorted according to their MIS values in ascending order in order to retain the downward closure property. This computation can be quite substantial if the database is large and resides on hard disk. Most databases for association rule mining are very large. We have implemented both techniques and experimented with a number of databases. Even without considering sorting of each transaction t, our implementation has similar performance as that in [4]. When sorting is included for the method in [4], our method, which requires no sorting of each transaction but only a sorting of all the items in I (which is much smaller in number), performs better on large databases.

4 Evaluation

The section evaluates the extended model. We show that the model allows us to find rules with very low supports (involving rare items) yet without generating a huge number of meaningless rules with frequent items. We first give the experiment results using synthetic data. We then present the results on our real-life data sets.

4.1 Experiments with synthetic data

The synthetic test data is generated with the data generator in [4]. It generates data that mimic the transactions in the retailing environment. The data generator is widely used for evaluating association rule mining algorithms.

For our experiments, we need a method to assign MIS values to the items in the data set. We use the actual frequencies (or the supports) of the items in the data as the basis for MIS assignments. Specifically, we use the following formulas:

$$MIS(i) = \begin{cases} M(i) & M(i) > LS \\ LS & Otherwise \end{cases}$$

$$M(i) = \beta f(i)$$

$f(i)$ is the actual frequency (or the support expressed in percentage of the data set size) of item i in the data. LS is the lowest minimum item support. β ($0 \leq \beta \leq 1$) is a parameter that controls how the MIS values for items should be related to their frequencies. Thus, to set MIS values for items we use two parameters, β and LS. If $\beta = 0$, we have only one minimum support, LS, which is the same as the traditional association rule mining. If $\beta = 1$ and $f(i) \geq LS$, $f(i)$ is the MIS value for i.

Example 7: Consider three items, 1, 2 and 3 in a data set, where $f(1) = 1\%$, $f(2) = 3\%$ and $f(3) = 10\%$. If we use $LS = 2\%$ and $\beta = 0.3$, then MIS(1) = 2%, MIS(2) = 2% and MIS(3) = 3%.

For our experiments, we generated a number of data sets to test our model. Here, we use the results from one data set to illustrate. The others are similar and thus omitted. This data set is generated with 1000 items, and 10 items per transaction on average [4]. The number of transaction is 100,000. The standard deviation of the item frequencies of the data set is 1.14% (the mean is 1.17% expressed in percentage of the total data set size). This shows that the frequencies of the items do not vary a great deal. (The synthetic data generator is designed for generating data used by mining algorithms with only one min-sup.) For our experiment, we use three very low LS values, 0.1%, 0.2%, and 0.3%. Figure 1 shows the number of large itemsets found. The three thick lines give the numbers of large itemsets found using the existing approach of a single minsup at 0.1%, 0.2% and 0.3% respectively. To show how β affects the number of large itemsets found by our method, we let $\beta = 1/\alpha$ and vary α from 1 to 20. Figure 2 gives the corresponding numbers of candidate itemsets in the experiment. Again the three thick lines give the number of candidate itemsets using the existing approach of a single minsup at 0.1%, 0.2% and 0.3% respectively

We see from Figure 1 that the number of large itemsets is significantly reduced by our method when α is not too large. When α becomes larger, the number of large itemsets found by our method gets closer to that found by the single minsup method. The reason is because when α becomes larger more and more many items' MIS values reach LS. From our application experiences, the user is usually satisfied with the large itemsets found at $\alpha = 4$. At $\alpha = 4$ and $LS = 0.2\%$, for example, the number of large itemsets found by our method is less than 61% of that found by the single minsup method. From Figure 2, we see that the corresponding numbers of candidate itemsets are also much less. The execution times are roughly the same (hence are not shown here) because database scan dominates the computation in this experiment. Below, we will see that for our real-life data set, the reductions in both the number of large itemsets found and the number of candidate itemsets used are much more remarkable because the item frequencies in our real-life data set vary a great deal. The execution times also drop drastically because the data set is small and the computation time is dominated by the itemsets generation.

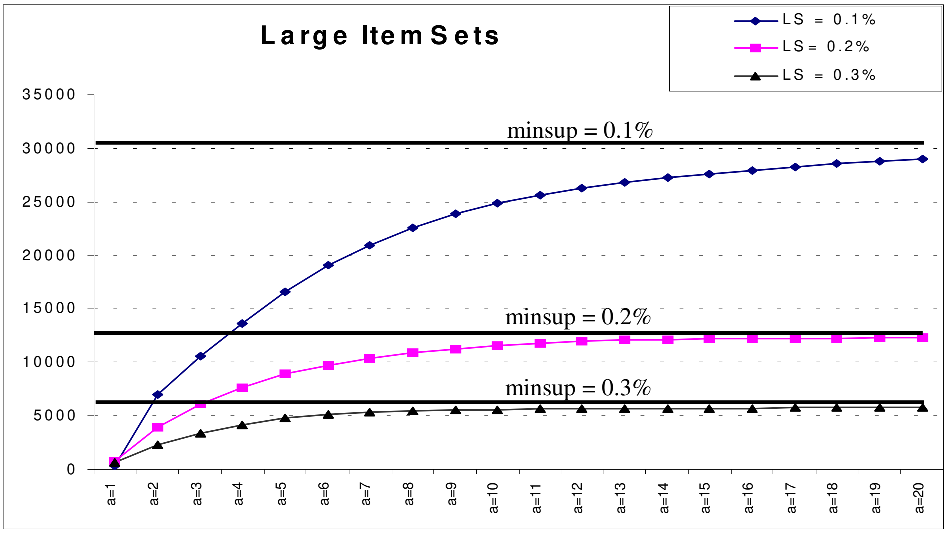

Figure 1. Number of large itemsets found

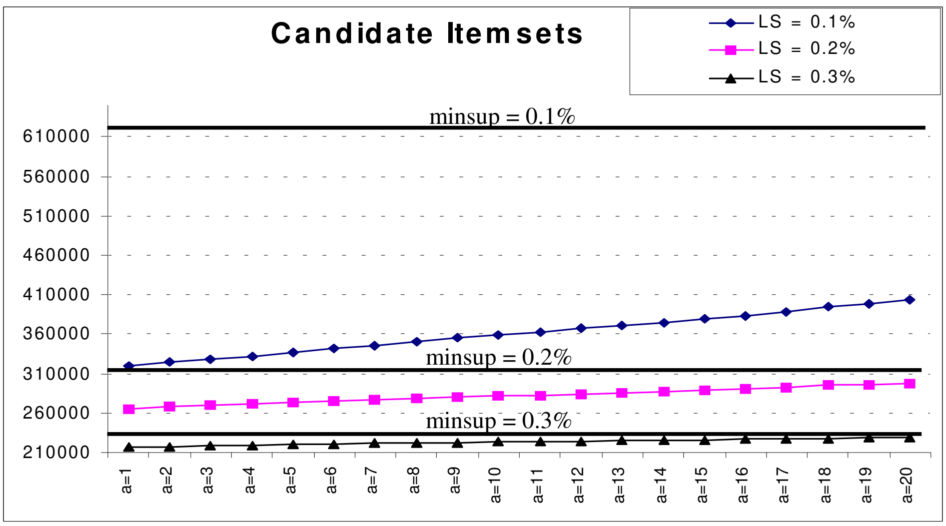

Figure 2. Number of candidate itemsets

4.2 Application to real-life data

We tested the algorithm using 3 real-life data sets. Here, we only describe one application. The results with the other two are quite similar.

Due to confidentiality agreement, we are unable to provide the details of the application. Here, we only give the characteristics of the data. The data set has 55 items and 700 transactions. Each transaction has 14-16 items. Some items can appear in 500 transactions, while some may only appear in 30 transactions. The standard deviation of item frequencies in the data is 25.4% (the mean is 24.3%).

For this application, the user sets LS = 1%. The results are shown in Figure 3, which include both the numbers of candidate itemsets and large itemsets found. The two thick lines show the number of candidate itemsets and the number of large itemsets found respectively by the single minsup (= 1%) method. Our new method reduces the numbers dramatically. For this application, the user is happy with the large itemsets found at $\alpha = 4$. The number of large itemsets found by our method at $\alpha = 4$ is only 8.5% of that found by the existing single minsup method. The drop in the number of candidate itemsets is even more drastic.

Figure 4 shows the execution time comparison in percentage. The execution time used by the single minsup method is set to 100%. We can see that the proposed method also reduces the execution time significantly (since this data set is small, the itemsets generation dominates the whole computation).

Note that for the real-life applications, the user can also assign MIS values manually rather than using the formulas in Section 4.1.

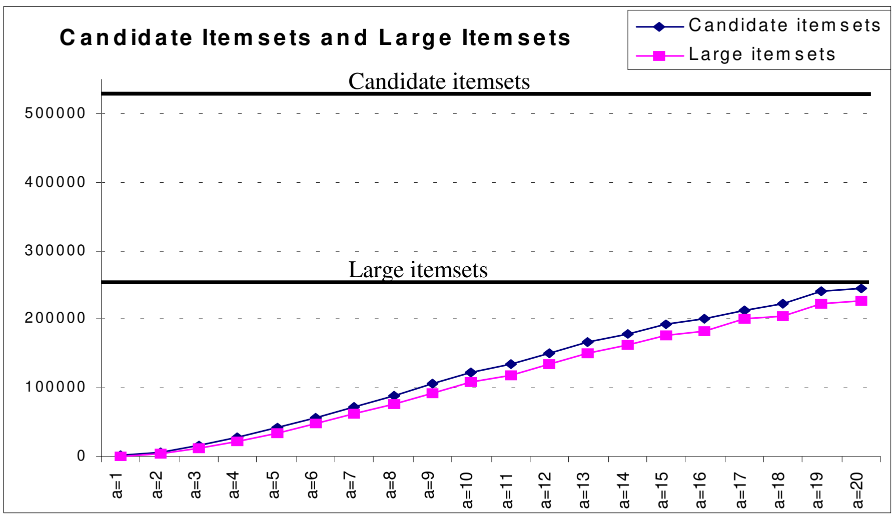

Figure 3. Numbers of candidate itemsets and large itemsets.

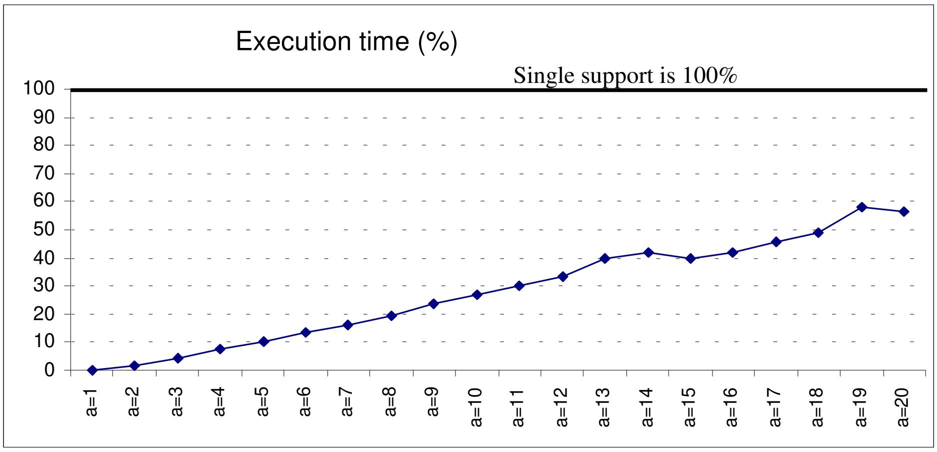

Figure 4. Comparison of execution times in percentage

5 Related Work

Association rule mining has been studied extensively in the past. However, existing research has been focused on improving the performance of mining algorithms [e.g., 2, 4, 7, 16, 19, 21, 23, 24, 28]; mining generalized (or multiple-level) rules [11, 26]; devising parallel algorithms [e.g., 3]; mining quantitative rules [e.g., 10, 23, 14]; finding constrained association rules [27, 21], etc. The model used in all these works is the same, i.e., with only one user-specified minimum support threshold.

Although multiple-level association rule mining in [11] can use different minimum supports at different levels of hierarchy, at the same level it uses only one minsup. For example, we have the taxonomy: *milk* and *cheese* are *Dairy_product*; and *pork* and *beef* are *Meat*. At the level of *Dairy_product* and *Meat*, association rules can have one minsup, and at the level of *milk*, *cheese*, *pork* and *beef*, there can be a different minsup. This model is essentially the same as the original model in [2] because each level has its own association rules involving items of that level. Our proposed model is more flexible as we can assign a MIS value for each item. [25] presents a generalized multiple-level association mining technique, where an association rule can involve items at any level of hierarchy. However, this model still uses only one minsup. Although we did not describe our extended model in the generalized association mining context, it is easy to see that our model applies there.

It is also easy to see that our algorithm MSapriori for finding large itemsets is a generalization of the Apriori algorithm [4] for the single minsup mining. That is, when all MIS values are the same as *LS*, it reduces to the Apriori algorithm. A key idea of the algorithm MSapriori is the sorting of items in *I* according to their MIS values in order to retain the downward closure property. Although we still use level-wise search, each step of our algorithm is different from that of algorithm Apriori, from initialization, candidate itemsets generation to pruning of candidate itemsets. The proposed algorithm also uses a different method to traverse the data structure storing candidate itemsets to avoid sorting of each transaction in the database, which is computationally expensive for large databases.

6 Conclusion

This paper argues that a single minsup is insufficient for association rule mining since it cannot reflect the natures and frequency differences of the items in the database. In real-life applications, such differences can be very large. It is neither satisfactory to set the minsup too high, nor is it satisfactory to set it too low. This paper proposes a more flexible and powerful model. It allows the user to specify multiple minimum item supports. This model enables us to found rare item rules yet without producing a huge number of meaningless rules with frequent items. The effectiveness of the new model is shown experimentally and practically.

References

[1] Aggarwal, C., and Yu, P. "Online generation of association rules." *ICDE-98*, 1998, pp. 402-411.

[2] Agrawal, R., Imielinski, T., Swami, A. “Mining association rules between sets of items in large databases.” *SIGMOD-1993*, 1993, pp. 207-216.

[3] Agrawal, R. and Shafer, J. C. “Parallel mining of association rules: Design, implementation, and experience.” *IEEE TKDE*, 8, 1996, pp. 962-969.

[4] Agrawal, R. and Srikant, R. “Fast algorithms for mining association rules.” *VLDB-94*, 1994.

[5] Bayardo, R., Agrawal, R, and Gunopulos, D. "Constraint-based rule mining in large, dense databases." To appear in *ICDE-99*, 1999.

[6] Brin, S. Motwani, R. and Silverstein, R. “Beyond market basket: generalizing association rules to correlations.” *SIGMOD-97*, 1997, 265-276.

[7] Brin, S. Motwani, R. Ullman, J. and Tsur, S. “Dynamic Itemset counting and implication rules for market basket data.” *SIGMOD-97*, 1997, pp. 255-264.

[8] Cheung, D. W., Han, J, V. Ng, and Wong, C.Y. “Maintenance of discovered association rules in large databases: an incremental updating technique.” *ICDE-96*, 1996, pp. 106-114.

[9] Fayyad, U., Piatesky-Shapiro, G. and Smyth, P. “From data mining to knowledge discovery: an overview,” In: *Advances in knowledge discovery and data mining*, U. Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusamy, (eds.), AAAI/MIT Press, 1996, pp. 1-34.

[10] Fukuda, T, Morimoto, Y. Morishita, S. and Tokuyama, T. “Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization.” *SIGMOD-96*, 1996.

[11] Han, J. and Fu, Y. “Discovery of multiple-level association rules from large databases.” *VLDB-95*, 1995.

[12] Klemetinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo, A.I. “Finding interesting rules from large sets of discovered association rules.” *CIKM-94*. 1994, pp. 401-407.

[13] Lee, W., Stolfo, S. J., and Mok, K. W. “Mining audit data to build intrusion detection models.” *KDD-98*.

[14] Lent, B, Swami, A., and Widom, J. “Clustering association rules.” *ICDE-97*, 1997.

[15] Mannila, H. "Database methods for data mining.” *KDD-98* tutorial, 1998.

[16] Mannila, H., Toivonen, H. and Verkamo, A. I. “Efficient algorithms for discovering association rules.” In AAAI workshop on knowledge discovery in databases, 1994.

[17] Mannila, H., Toivonen, H. and Verkamo, A. I. “Discovery of frequent episodes in sequence.” *Data Mining and Knowledge Discovery*, 1(3): 259-289, 1997.

[18] Moore, D.S. “Tests for chi-squared type.” In: R. B. D’Agostino and M. A. Stephens (eds), *Googness-of-Fit Techniques*, Marcel Dekker, New York, 1996, pp. 63-95.

[19] Meo, R., Psaila, G. and Ceri, S. “A new SQL-like operator for mining association rules.” *VLDB-96*, 1996, pp. 122-133.

[20] Megiddo, M. and Srikant, R. “Discovering predictive association rules.” *KDD-98*, 1998.

[21] Ng. R. T. Lakshmanan, L. Han, J. 1998. “Exploratory mining and pruning optimizations of constrained association rules.” *SIGMOD-98*, 1998.

[22] Park, J. S. Chen, M. S. and Yu, P. S. 1995. “An effective hash based algorithm for mining association rules.” *SIGMOD-95*, 1995, pp. 175-186.

[23] Rastogi, R. and Shim, K. 1998. “Mining optimized association rules with categorical

and numberic attributes.” *ICDE –98*, 1998.

[24] Savasere, A. Omiecinski, E. and Navathe, S. “An efficient algorithm for mining association rules in large databases.” *VLDB-95*, 1995, pp. 432-444.

[25] Silverstein, C., Brin, S., Motiwani, R., and Ullman, J. “Scalable techniques for mining causal structures.” *VLDB-98*, 1998.

[26] Srikant, R. and Agrawal, R. “Mining generalized association rules.” *VLDB-1995*, 1995.

[27] Srikant, R., Vu, Q. and Agrawal, R. “Mining association rules with item constraints.” *KDD-97*, 1997, pp. 67-73.

[28] Toivonen, H. “Sampling large databases for association rules.” *VLDB-96*, pp. 134-145.

[29] Tsur, D., Ullman, J. D., Abitboul, S. Clifton, C. Motwani, R. and Nestorov, S. “Query flocks: a generalization of association rule mining.” http://www-db.standford.edu/~ullman/pub/qflocks.ps, Oct. 1997.