

# Efficient Mining of Generalized Association Rules with Non-uniform Minimum Support

Ming-Cheng Tseng<sup>1</sup> and Wen-Yang Lin<sup>2,\*</sup>

<sup>1</sup> Institute of Information Engineering  
I-Shou University  
Kaohsiung 840, Taiwan  
clark.tseng@msa.hinet.net

<sup>2</sup> Department of Computer Science and Information Engineering  
National University of Kaohsiung  
Kaohsiung 811, Taiwan  
wylin@nuk.edu.tw

\*(Corresponding author)

**Abstract.** Mining generalized association rules between items in the presence of taxonomies has been recognized as an important model in data mining. Earlier work on generalized association rules confined the minimum supports to be uniformly specified for all items or for items within the same taxonomy level. This constraint on minimum support would restrain an expert from discovering some deviations or exceptions that are more interesting but much less supported than general trends. In this paper, we extended the scope of mining generalized association rules in the presence of taxonomies to allow any form of user-specified multiple minimum supports. We discuss the problems of using classic Apriori itemset generation and presented two algorithms, MMS\_Cumulate and MMS\_Stratify, for discovering the generalized frequent itemsets. Empirical evaluation showed that these two algorithms are very effective and have good linear scale-up characteristics.

**Keywords:** Data mining, generalized association rules, multiple minimum supports, taxonomy.

## 1. Introduction

Mining association rules from a large business database, such as transaction records, has been an important topic in the area of data mining. This problem is motivated by applications known as market basket analysis to find relationships between items purchased by customers, that is, what kinds of products tend to be purchased together [1]. Such information is useful in many aspects of market management, such as store layout planning, target marketing, and understanding customer behavior.

An association rule is an expression of the form  $A \Rightarrow B$ , where  $A$  and  $B$  are sets of items. Such a rule reveals that transactions in the database containing items in  $A$  tend to contain items in  $B$ , and the probability, measured as the fraction of transactions containing  $A$  also containing  $B$ , is called the

*confidence* of the rule. The *support* of the rule is the fraction of the transactions that contain all items in both *A* and *B*.

For example, an association rule,

Desktop PC  $\Rightarrow$  Ink-jet printer (*sup* = 30%, *conf* = 60%),

says that 30% (support) of customers purchase both Desktop PC and Ink-jet printer together, and 60% (confidence) of customers who purchase Desktop PC also purchase Ink-jet printer.

For an association rule to hold, the support and the confidence of the rule should satisfy a user-specified minimum support called *minsup* and minimum confidence called *minconf*, respectively. The problem of mining association rules is to discover all association rules that satisfy *minsup* and *minconf*. This task is usually decomposed into two steps:

1. Frequent itemset generation: generate all itemsets that exceed the *minsup*.
2. Rule construction: construct all association rules that satisfy *minconf* from the frequent itemsets in Step 1.

Intuitively, to discover frequent itemsets, each transaction has to be inspected to generate the supports of all combinations of items, which, however, will suffer for lots of I/O operations as well as computations. Therefore, most early work was focused on deriving efficient algorithms for finding frequent itemsets [2][4][9][10]. The most well-known is Apriori [2], which relies on the observation that an itemset can be frequent if and only if all of its subsets are frequent and thus a level-wise inspection proceeding from frequent 1-itemsets to the maximal frequent itemset can avoid large numbers of I/O accesses.

Despite the great achievement in improving the efficiency of mining algorithms, the existing association rule models used in all of these studies incur some problems. First, in many applications, there are taxonomies (hierarchies), explicitly or implicitly, over the items. It may be more useful to find association at different levels of the taxonomy [5][11] than only at the primitive concept level.

Second, the frequencies of items are not uniform. Some items occur very frequently in the transactions while others rarely appear. A uniform minimum support assumption would hinder the discovery of some deviations or exceptions that are more interesting but much less supported than general trends. Furthermore, a single minimum support also ignores the fact that support requirement varies at different levels when mining association rules in the presence of taxonomy.

These observations lead us to the investigation of mining generalized association rules across different levels of taxonomy with non-uniform minimum supports. The methods we propose not only can discover associations that span different hierarchy levels but also have high potential in producing rare but informative item rules.

The remainder of this paper is organized as follows. The problem of mining generalized association rules with multiple minimum supports is formalized in Section 2. In Section 3, we explain the proposed algorithms for finding frequent itemsets. The evaluation of the proposed algorithms on IBM synthetic data and Microsoft foodmart2000 is described in Section 4. A review of related work is

given in Section 5. Finally, our conclusions are stated in Section 6.

## 2. Problem Statement

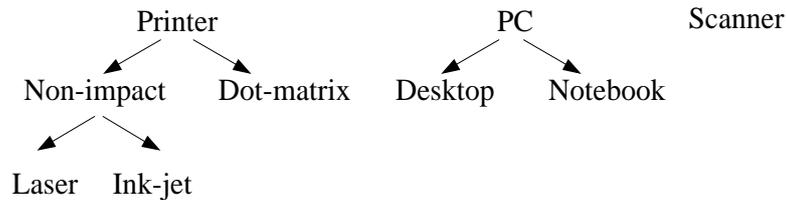
In this section, the basic concept behind generalized association rules in [11] is introduced. We then extend this model to incorporate multiple minimum supports.

### 2.1 Mining generalized association rules

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of items and  $D = \{t_1, t_2, \dots, t_n\}$  be a set of transactions, where each transaction  $t_i = \langle tid, A \rangle$  has a unique identifier  $tid$  and a set of items  $A$  ( $A \subseteq I$ ). To study the mining of generalized association rules from  $D$ , we assume that the taxonomy of items,  $T$ , is available and denoted as a directed acyclic graph on  $I \cup J$ , where  $J = \{j_1, j_2, \dots, j_p\}$  represents the set of generalized items derived from  $I$ . An edge in  $T$  denotes an *is-a* relationship, that is, if there is an edge from  $j$  to  $i$  we call  $j$  a parent (generalization) of  $i$  and  $i$  a child of  $j$ . The meanings of *ancestor* and *descendant* follow from the transitive-closure of the *is-a* relationship. Figure 1 illustrates a taxonomy constructed for  $I = \{\text{Laser printer, Ink-jet printer, Dot matrix printer, Desktop PC, Notebook, Scanner}\}$  and  $J = \{\text{Non-impact printer, Printer, PC}\}$ .

Note that the above transactions and taxonomy definitions imply that all leaves in  $T$  are formed from  $I$  while others are from  $J$  but only real items in  $I$  can appear in the transactions.

**Definition 1.** Given a transaction  $t = \langle tid, A \rangle$ , we say an itemset  $B$  is in  $t$  if every item in  $B$  is in  $A$  or is an ancestor of some items in  $A$ . An itemset  $B$  has support  $s$ , denoted as  $s = \text{sup}(B)$ , in the transaction set  $D$  if  $s\%$  of transactions in  $D$  contain  $B$ .



**Fig. 1.** An example of taxonomy.

**Definition 2.** Given a set of transactions  $D$  and a taxonomy  $T$ , a generalized association rule is an implication of the following form

$$A \Rightarrow B,$$

where  $A, B \subset I \cup J$ ,  $A \cap B = \emptyset$ , and no item in  $B$  is an ancestor of any item in  $A$ . The support of this rule,  $\text{sup}(A \Rightarrow B)$ , is equal to the support of  $A \cup B$ . The confidence of the rule,  $\text{conf}(A \Rightarrow B)$ , is the ratio of  $\text{sup}(A \cup B)$  versus  $\text{sup}(A)$ , i.e., the fraction of transactions in  $D$  containing  $A$  that also contain

B.

According to the definition of generalized association rules, an itemset is composed not simply of items in leaves of the taxonomy but also of generalized items in higher levels of the hierarchy. This is why we must take the ancestors of items in a transaction into account while determining the support of an itemset. In addition, the condition in Definition 2 that no item in  $B$  is an ancestor of any item in  $A$  is essential; otherwise, a rule of the form,  $a \Rightarrow ancestor(a)$ , always has 100% confidence and is trivial.

**Definition 3.** *The problem of mining generalized association rules is that, given a set of transactions  $D$  and a taxonomy  $T$ , find all generalized association rules that have support and confidence greater than a user-specified minimum support ( $minsup$ ) and minimum confidence ( $minconf$ ), respectively.*

## 2.2 Multiple-support specification

To allow the user to specify different minimum supports for different items, we have to extend the uniform support used in generalized association rules. The definition of generalized association rules remains the same but the minimum support is changed. Following the concept in [7], we assume that the user can specify different minimum supports for different items in the taxonomy.

**Definition 4.** *Let  $ms(a)$  denote the minimum support of an item  $a$  in  $I \cup J$ . An itemset  $A = \{a_1, a_2, \dots, a_k\}$ , where  $a_i \in I \cup J$  for  $1 \leq i \leq k$ , is frequent if the support of  $A$  is larger than the lowest value of minimum support of items in  $A$ , i.e.,*

$$sup(A) \geq \min_{a_i \in A} ms(a_i),$$

**Definition 5.** *A generalized association rule  $A \Rightarrow B$  is strong if the support satisfies the following condition*

$$sup(A \Rightarrow B) \geq \min_{a_i \in A \cup B} ms(a_i),$$

and

$$conf(A \Rightarrow B) \geq minconf.$$

**Definition 6.** *Given a set of transactions  $D$ , a taxonomy  $T$  composed of items  $a_1, a_2, \dots, a_n$ , and a minimum confidence,  $minconf$ , the problem of mining generalized association rules with multiple user-specified minimum item supports  $ms(a_1), ms(a_2), \dots, ms(a_n)$  associated with each item in  $T$  is to find all generalized association rules that are strong.*

The idea of mining generalized association rules with multiple minimum supports is better illustrated with an example.

**Example 1.** Suppose that a shopping transaction database  $D$  in Table 1 consists of items  $I = \{\text{Laser printer, Ink-jet printer, Dot matrix printer, Desktop PC, Notebook, Scanner}\}$  and the taxonomy  $T$  is as shown in Figure 1. Let the minimum support ( $ms$ ) associated with each item in the taxonomy be as follows:

$$\begin{array}{lll} ms(\text{Printer}) = 80\% & ms(\text{Non-impact}) = 65\% & ms(\text{Dot matrix}) = 70\% \\ ms(\text{Laser}) = 25\% & ms(\text{Ink-jet}) = 60\% & ms(\text{Scanner}) = 15\% \\ ms(\text{PC}) = 35\% & ms(\text{Desktop}) = 25\% & ms(\text{Notebook}) = 25\% \end{array}$$

Let  $minconf$  be 60%. The support of the following generalized association rule,

$$\text{PC, Laser} \Rightarrow \text{Dot matrix} \text{ (} sup = 16.7\%, conf = 50\% \text{),}$$

is less than  $\min\{ms(\text{PC}), ms(\text{Laser}), ms(\text{Dot matrix})\} = 25\%$ , which makes this rule fail. However, another rule,

$$\text{PC} \Rightarrow \text{Laser} \text{ (} sup = 33.3\%, conf = 66.7\% \text{),}$$

holds because the support satisfies  $\min\{ms(\text{PC}), ms(\text{Laser})\} = 25\%$ , and the confidence is larger than  $minconf$ . Table 2 lists the frequent itemsets and resulting strong rules.

**Table 1.** A transaction database ( $D$ ).

$tid$	Items Purchased
11	Notebook, Laser printer
12	Scanner, Dot-matrix printer
13	Dot-matrix printer, Ink-jet printer
14	Notebook, Dot-matrix printer, Laser printer
15	Scanner
16	Desktop computer

One of the difficulties in applying association rules mining to real-world applications is the setting of support constraint. The situation becomes worse when non-uniform, multiple item support specifications are allowed. The simplest method is to leave the work to the users. This approach has the most flexibility but places the users in a dilemma: How to specify the most appropriate support constraint, either uniform or non-uniform, to discover interesting patterns without suffering from combinatorial explosion and missing some less-supported but perceptive rules.

A more instructive method is using the frequencies (or supports) of items within the database [7], defined as follows:

$$ms(a) = \begin{cases} \alpha \times sup(a), & \text{if } \alpha \times sup(a) > minsup \\ minsup, & \text{otherwise} \end{cases} \quad (1)$$

where the parameter  $\alpha$  ( $0 \leq \alpha \leq 1$ ) is employed to control how the minimum support of an item  $a$ ,  $ms(a)$ , is related to its actual frequency in the database. If  $\alpha = 0$ , the specification degenerates to the uniform case. As the formula indicates, users still have to specify the value of  $minsup$  and an additional parameter  $\alpha$ . The problem thus remains tangling.

**Table 2.** Frequent itemsets and association rules generated for Example 1.

Itemset	min $ms$ (%)	Support (%)
{ Scanner }	15	33.3
{ PC }	35	50.0
{ Notebook }	25	33.3
{ Laser }	25	33.3
{ Scanner, Printer }	15	16.7
{ Scanner, Dot matrix }	15	16.7
{ Laser, PC }	25	33.3
{ Notebook, Printer }	25	33.3
{ Notebook, Non-impact }	25	33.3
{ Notebook, Laser }	25	33.3
Rules		
PC $\Rightarrow$ Laser ( $sup = 33.3\%$ , $conf = 66.7\%$ )		
Laser $\Rightarrow$ PC ( $sup = 33.3\%$ , $conf = 100\%$ )		
Notebook $\Rightarrow$ Printer ( $sup = 33.3\%$ , $conf = 100\%$ )		
Notebook $\Rightarrow$ Non-impact ( $sup = 33.3\%$ , $conf = 100\%$ )		
Notebook $\Rightarrow$ Laser ( $sup = 33.3\%$ , $conf = 100\%$ )		
Laser $\Rightarrow$ Notebook ( $sup = 33.3\%$ , $conf = 100\%$ )		

To solve this problem, we have proposed in [6] an approach for minimum support specification without consulting users. The basic idea of our approach is to “push” the confidence and lift measure (or called *positive correlation* [3][8]) into the support constraint to prune the spurious frequent itemsets that fail to generate interesting associations as early as possible. First, let us show how the constraint is specified to reduce the frequent itemsets that fail in generating strong associations.

**Lemma 1.** Let  $A \cup B$  be a frequent itemset,  $A \cap B = \emptyset$ , and without loss of generality, let  $a \in A$  and  $a$  be the one with the smallest minimum support, i.e.,  $ms(a) = \min_{a_i \in A \cup B} ms(a_i)$ . If the minimum support of  $a$  is set as  $ms(a) = sup(a) \times minconf$ , then the association rule  $A \Rightarrow B$  is strong, i.e.,  $sup(A \cup B) / sup(A) \geq minconf$ .

**Proof.** Since  $A \cup B$  is frequent,  $sup(A \cup B) \geq ms(a) = sup(a) \times minconf$ . Furthermore,  $sup(a) \geq sup(A)$  since  $a \in A$ . It follows that  $sup(A \cup B) / sup(A) \geq minconf$ . ■

According to Lemma 1, the minimum support of an item  $a$ , for  $a \in I \cup J$ , is specified as follows:

$$ms(a) = sup(a) \times minconf. \quad (2)$$

Note that Lemma 1 does not imply that the rule  $B \Rightarrow A$  is strong. This is because the confidence measure is not symmetric over the antecedence and consequence. Therefore, Eq. 2 does not guarantee that all rules generated from the frequent itemsets will be strong.

Next we consider how to specify the constraint to generate interesting associations.

**Definition 7.** An association rule is interesting if it is strong and

$$lift(A \Rightarrow B) = \frac{sup(A \cup B)}{sup(A)sup(B)} = \frac{conf(A \Rightarrow B)}{sup(B)} > 1.$$

In the above definition, we have introduced an extra constraint, *lift*, which is employed to measure the deviation of the rule from correlation. For example, consider the transaction database in Table 1. For a minimum support of 30% and minimum confidence of 50%, the following association rule is discovered:

$$\text{Scanner} \Rightarrow \text{Printer} \quad (sup = 33.3\%, conf = 50\%).$$

One may conclude that this rule is interesting because of its high support and high confidence. However, note that the support of the generalized item **Printer** is 66.7%. This means that a customer who is known to purchase **Scanner** is less likely to buy **Printer** (by 16.7%) than a customer about whom we have no information. In other words, buying **Scanner** and purchasing **Printer** is negatively correlated, indicated by  $lift = 50/66.7 = 0.75 < 1$ .

**Theorem 1.** Let  $I$  be a set of items and the minimum support of each item be specified below

$$ms(a_i) = sup(a_i) \times \max_{a_j \in I \cup J - \{a_i\}} sup(a_j). \quad (3)$$

Then any strong association rule  $A \Rightarrow B$ , for  $A, B \subseteq I$  and  $A \cap B = \emptyset$ , is interesting.

**Proof.** Since  $A \Rightarrow B$  is strong,  $sup(A \cup B) \geq \min_{a \in A \cup B} ms(a)$ . Specifically, let  $a = a_i$ . Then, we have

$$sup(A \cup B) \geq ms(a_i) = sup(a_i) \times \max_{a_j \in I \cup J - \{a_i\}} sup(a_j).$$

Since  $A \cap B = \emptyset$ ,  $a_i$  belongs to either  $A$  or  $B$ . Without loss of generality, let  $a_i \in A$ . It is easy to show that

$$sup(A) \leq sup(a_i) \text{ and } sup(B) \leq \max_{a_j \in I \cup J - \{a_i\}} sup(a_j).$$

The lemma then follows. ■

Note that the support constraint specified in Eq. 3 only provides a sufficient condition for obtaining interesting association rules from frequent itemsets. There may exist some itemsets that are infrequent with respect to this constraint but can generate positive lift associations. To construct all associations without missing any positive lift rule, we should refine the support specification. The intuition is to set the minimum support of an item  $a$  to accommodate all frequent itemsets that consist of  $a$  as the smallest supported item and are capable of generating at least one positive lift association rule.

Without loss of generality, we assume that  $a \in A$  and  $a$  is the one with the smallest minimum support over itemset  $A \cup B$ , i.e.,  $ms(a) = \min_{a_i \in A \cup B} ms(a_i)$ , and  $b \in B$ ,  $ms(b) = \min_{a_i \in B} ms(a_i)$ . The following conditions hold,

$$sup(A \cup B) \geq ms(a), sup(A) \leq sup(a), \text{ and } sup(B) \leq sup(b).$$

Thus, to make  $lift(A \Rightarrow B) > 1$ , we would specify  $ms(a) = sup(a) \times sup(b)$ . Note that  $b$  can be any item in the item set  $I$  except  $a$ , and  $sup(b) \geq sup(a)$ . What we need is the smallest qualified item, i.e.,  $b = \min\{a_i \mid a_i \in I - \{a\} \text{ and } sup(a_i) \geq sup(a)\}$ . Let  $I = \{a_1, a_2, \dots, a_n\}$ , and  $sup(a_i) \leq sup(a_{i+1})$ ,  $1 \leq i \leq n - 1$ . The minimum item support with respect to nonnegative lift can be specified as follows:

$$ms(a_i) = \begin{cases} sup(a_i) \times sup(a_{i+1}), & \text{if } 1 \leq i \leq n - 1 \\ sup(a_i), & \text{if } i = n \end{cases} \quad (4)$$

Now we have two separate support settings: The first is based on the confidence measure and the second is based on lift. To prune the spurious frequent itemsets so as to make most of the generated rules become interesting, we combine these two specifications as shown below, which we call the *confidence-lift support constraint (CLS)*.

$$ms(a_i) = \begin{cases} sup(a_i) \times \max\{minconf, sup(a_{i+1})\}, & \text{if } 1 \leq i \leq n - 1 \\ sup(a_i), & \text{if } i = n \end{cases} \quad (5)$$

**Example 2.** Let  $minconf = 50\%$ . The first two columns of Table 3 show the sorted list of all items, primitive or generalized, along with their supports. Then, according to Eq. 5, it is not hard to derive the minimum item supports, as shown in the last column. For example,  $ms(\text{Desktop}) = sup(\text{Desktop}) \times \max\{minconf, sup(\text{lnk-jet})\} = 1/6 \times 1/2 = 8.3\%$ ,  $ms(\text{PC}) = sup(\text{PC}) \times \max\{minconf, sup(\text{Printer})\} = 1/2 \times 2/3 = 33.3\%$ , and  $ms(\text{Printer}) = sup(\text{Printer}) = 66.7\%$  since Printer is the last item.

### 3. Methods for Generating Frequent Itemsets with Multiple Minimum

# Supports

## 3.1 Algorithm basics

Intuitively, the process of mining generalized association rules with multiple minimum supports is the same as that used in traditional association rules mining: First all frequent itemsets are discovered, and then from these itemsets rules that have large confidence are generated. Since the second phase is straightforward after all frequent itemsets have been found, we concentrate on algorithms for finding all frequent itemsets. We propose two methods, called MMS\_Cumulate and MMS\_Stratify, which are generalization of the Cumulate and Stratify algorithms presented in [11]; and MMS stands for Multiple Minimum Supports.

**Table 3.** Sorted list of items along with their supports and minimum supports.

Item	Support (%)	ms (%)
Desktop	16.7	8.3
Ink-jet	16.7	8.3
Laser	33.3	16.7
Notebook	33.3	16.7
Scanner	33.3	16.7
Dot-matrix	50.0	25.0
Non-impact	50.0	25.0
PC	50.0	33.3
Printer	66.7	66.7

Let  $k$ -itemset denote an itemset with  $k$  items. Our algorithm follows the level-wise approach widely used in most efficient algorithms to generate all frequent  $k$ -itemsets. First, scan the entire database  $D$  and count the occurrence of each item to generate the set of all frequent 1-itemsets ( $L_1$ ). In each subsequent step  $k$ ,  $k \geq 2$ , the set of frequent  $k$ -itemsets,  $L_k$ , is generated as follows: 1) Generate a set of candidate  $k$ -itemsets,  $C_k$ , from  $L_{k-1}$ , using the apriori-gen procedure described in [2]; and 2) Scan the database  $D$ , count the occurrence of each itemset in  $C_k$ , and prune those with less support. The resulting set is  $L_k$ .

The effectiveness of this approach relies heavily on a *downward closure property* (also called *Apriori property* [2]): if a  $k$ -itemset is frequent, then all of its subsets are frequent or, contrapositively, if any subset of a  $k$ -itemset is not frequent, then neither is the  $k$ -itemset. Hence, we can preprune some less supported  $k$ -itemsets in the course of examining  $(k-1)$ -itemsets. The downward closure property, however, may fail in the case of multiple minimum supports. For example, consider four items  $a$ ,  $b$ ,  $c$ , and  $d$  that have minimum supports specified as  $ms(a) = 15\%$ ,  $ms(b) = 20\%$ ,  $ms(c) = 4\%$ , and  $ms(d) = 6\%$ . Clearly, a 2-itemset  $\{a, b\}$  with 10% support is discarded for  $10\% < \min\{ms(a), ms(b)\}$ .

According to the downward closure, the 3-itemsets  $\{a, b, c\}$  and  $\{a, b, d\}$  will be pruned even though their supports may be larger than  $ms(c)$  and  $ms(d)$ , respectively.

To solve this problem, Liu et al. [7] proposed a concept called *sorted closure property*, which assumes that all items within an itemset are sorted in increasing order of their minimum supports. Since this important property has not been clearly defined, we provide a formalization. Hereafter, to distinguish from the traditional itemset, a sorted  $k$ -itemset is denoted as  $\langle a_1, a_2, \dots, a_k \rangle$ .

**Lemma 2. (Sorted closure)** *If a sorted  $k$ -itemset  $\langle a_1, a_2, \dots, a_k \rangle$ , for  $k \geq 2$  and  $ms(a_1) \leq ms(a_2) \leq \dots \leq ms(a_k)$ , is frequent, then all of its sorted subsets with  $k - 1$  items are frequent, except for the subset  $\langle a_2, a_3, \dots, a_k \rangle$ .*

**Proof.** The  $k$ -itemset  $\langle a_1, a_2, \dots, a_k \rangle$  has  $k$  subsets with  $k-1$  items, which can be divided into two groups with or without  $a_1$  included, i.e.,

$$\begin{aligned} \text{group 1: } & \langle a_1, a_2, \dots, a_{k-1} \rangle, \langle a_1, a_2, \dots, a_{k-2}, a_k \rangle, \dots, \langle a_1, a_3, \dots, a_k \rangle \\ \text{group 2: } & \langle a_2, a_3, \dots, a_k \rangle \end{aligned}$$

Note that all of the itemsets in group 1 have the same lowest minimum item support as that of  $\langle a_1, a_2, \dots, a_k \rangle$ , i.e.,  $ms(a_1)$ , while  $\langle a_2, a_3, \dots, a_k \rangle$  does not, which is  $ms(a_2)$ . Since  $ms(a_2) \geq ms(a_1)$ , the lemma follows. ■

Again, let  $L_k$  and  $C_k$  represent the set of frequent  $k$ -itemsets and candidate  $k$ -itemsets, respectively. We assume that any itemset in  $L_k$  or  $C_k$  is sorted in increasing order of the minimum item supports. The result in Lemma 2 reveals the obstacle in using the apriori-gen procedure for generating frequent itemsets.

**Lemma 3.** *For  $k = 2$ , the procedure  $apriori\text{-}gen(L_1)$  fails to generate all candidate 2-itemsets in  $C_2$ .*

**Proof.** Note that if a sorted candidate 2-itemset  $\langle a, b \rangle$  is generated from  $L_1$ , then both items  $a$  and  $b$  should be included in  $L_1$ ; that is, each one should occur more frequently than the corresponding minimum support  $ms(a)$  and  $ms(b)$ . Clearly, the case  $ms(a) \leq sup(a) < sup(b) < ms(b)$  fails to generate  $\langle a, b \rangle$  in  $C_2$  even  $sup(\langle a, b \rangle) \geq ms(a)$ . The lemma then follows. ■

To solve this problem, [7] suggested using a sorted itemset, called *frontier set*,  $F = \langle a_j, a_{j_1}, a_{j_2}, \dots, a_{j_l} \rangle$ , to generate the set of candidate 2-itemsets, where

$$\begin{aligned} a_j &= \min_{a_i \in I \cup J} \{a_i \mid sup(a_i) \geq ms(a_i)\}, \\ ms(a_j) &\leq ms(a_{j_1}) \leq ms(a_{j_2}) \leq \dots \leq ms(a_{j_l}), \\ sup(a_{j_i}) &\geq ms(a_j), \text{ for } 1 \leq i \leq l. \end{aligned}$$

The procedure,  $C_2\text{-gen}(F)$ , using  $F$  to generate  $C_2$  is shown in Figure 2.

```

for each item  $a \in F$  in the same order do
  if  $\text{sup}(a) \geq \text{ms}(a)$  then
    for each item  $b \in F$  that is after  $a$  do
      if  $\text{sup}(b) \geq \text{ms}(a)$  then
        insert  $\langle a, b \rangle$  into  $C_2$ ;

```

**Fig. 2.** Procedure  $C_2\text{-gen}(F)$ .

**Example 3.** Continuing with Example 1, we change  $\text{ms}(\text{Scanner})$  from 15% to 20%. The resulting  $F$  is shown in Table 4. We observe that  $\text{ms}(\text{Scanner})$  is the smallest of all items, and **Scanner** could join with any item whose support is greater than or equal to  $\text{ms}(\text{Scanner}) = 20\%$  to become a candidate 2-itemset without losing any 2-itemsets. The 2-itemsets  $\langle \text{Scanner}, \text{Desktop} \rangle$  and  $\langle \text{Scanner}, \text{Ink-jet} \rangle$  could not become candidates because  $\text{sup}(\text{Desktop})$  or  $\text{sup}(\text{Ink-jet})$  is less than  $\text{ms}(\text{Scanner})$ , and the supports of  $\langle \text{Scanner}, \text{Desktop} \rangle$  and  $\langle \text{Scanner}, \text{Ink-jet} \rangle$  could not be greater than  $\text{sup}(\text{Desktop})$  and  $\text{sup}(\text{Ink-jet})$ , respectively, according to the downward closure property. Therefore, we keep items whose support is greater than or equal to  $\text{ms}(\text{Scanner})$  in  $F$ , and discard **Desktop** and **Ink-jet**.

**Table 4.** The generated frontier set  $F$ .

Item	Sorted $\text{ms}$ %	Support %	$F$
Scanner	20	33.3	Scanner
Laser	25	33.3	Laser
Desktop	25	<u>16.7</u>	Notebook
Notebook	25	33.3	PC
PC	35	50.0	Non-impact
Ink-jet	60	<u>16.7</u>	Dot-matrix
Non-impact	65	50.0	Printer
Dot-matrix	70	50.0	
Printer	80	66.7	

**Lemma 4.** For  $k \geq 3$ , any  $k$ -itemset  $A = \langle a_1, a_2, \dots, a_k \rangle$  generated by procedure  $\text{apriori-gen}(L_{k-1})$  can be pruned if there exists one  $(k-1)$  subset of  $A$ , say  $\langle a_{i_1}, a_{i_2}, \dots, a_{i_{k-1}} \rangle$ , such that  $\langle a_{i_1}, a_{i_2}, \dots, a_{i_{k-1}} \rangle \notin L_{k-1}$  and  $a_{i_1} = a_1$  or  $\text{ms}(a_{i_1}) = \text{ms}(a_{i_2})$ .

**Proof.** It is straightforward from the contrapositive statement in Lemma 2. ■

The procedure for generating the set of candidate  $k$ -itemsets ( $k \geq 3$ ), called  $C_k\text{-gen}(L_{k-1})$ , is shown in Figure 3, which consists of two steps: (1) calling  $\text{apriori-gen}$  to produce candidate itemsets, and (2) pruning from  $C_k$  those itemsets that satisfy Lemma 4.

```

 $C_k = \text{apriori-gen}(L_{k-1});$  /* Joins  $L_{k-1}$  with  $L_{k-1}$  */
for each itemset  $A = \langle a_1, a_2, \dots, a_k \rangle \in C_k$  do
    for each  $(k-1)$ -itemset  $A' = \langle a_{i_1}, a_{i_2}, \dots, a_{i_{k-1}} \rangle$  of  $A$  do
        if  $a_1 = a_{i_1}$  or  $ms(a_{i_1}) = ms(a_{i_2})$  then
            if  $A' \notin L_{k-1}$  then delete  $A$  from  $C_k$ ;

```

**Fig. 3.** Procedure  $C_k\text{-gen}(L_{k-1})$ .

### 3.2 Algorithm MMS\_Cumulate

As stated in [11], the main problem arisen from incorporating taxonomy information into association rule mining is how to effectively compute the occurrences of an itemset  $A$  in the transaction database  $D$ . This involves checking for each item  $a \in A$  whether  $a$  or any of its descendants are contained in a transaction  $t$ . Intuitively, we can simplify the task by first adding the ancestors of all items in a transaction  $t$  into  $t$ . Then a transaction  $t$  contains  $A$  if and only if the extended transaction  $t^+$  contains  $A$ . Following the Cumulate algorithm in [11], our MMS\_Cumulate is deployed according to this simple concept with the following enhancements:

**Enhancement 1. Ancestors pre-computing:** instead of traversing the taxonomy  $T$  to determine the ancestors for each item, we pre-compute the ancestors of each item. The result is stored as a table called  $IA$ .

**Enhancement 2. Ancestors filtering:** only ancestors that are in one or more candidates of the current  $C_k$  are added into a transaction. That is, any ancestor in  $IA$  that is not present in any of the candidates in  $C_k$  is pruned.

**Enhancement 3. Itemset pruning:** in each  $C_k$ , any itemset that contains both an item and its ancestor is pruned. This is derived from the following observation. Note that the pruning should be performed for each  $C_k$  ( $k \geq 2$ ), instead of  $C_2$  only<sup>1</sup>.

**Lemma 5.** [11] *The support of an itemset  $A$  that contains both an item  $a$  and its ancestor  $\hat{a}$  is the same as the support for itemset  $A - \{\hat{a}\}$ .*

**Proof.** The proof is given in [11].

**Enhancement 4. Item pruning:** an item in a transaction  $t$  can be pruned if it is not present in any of the candidates in  $C_k$ , as justified by Lemma 4. Note that this should be performed after Enhancement 2.

---

<sup>1</sup> The statement of Lemma 2 in [11] is incorrect. For example, consider two itemsets  $\{a, b\}$  and  $\{a, c\}$  in  $L_2$ , and  $c$  is an ancestor of  $b$ . Note that  $b$  and  $c$  are not in the same itemset, but clearly  $\{a, b, c\}$  will be in  $C_3$ . This implies that we have to perform the pruning not only in  $C_2$  but also all subsequent  $C_k$ , for  $k \geq 3$ .

**Lemma 6.** For  $k \geq 2$ , an item  $a$  that is not present in any itemset of  $L_k$  will not be present in any itemset of  $C_{k+1}$ .

**Proof.** This is straightforward from the fact that  $C_{k+1}$  is derived from joining  $L_k$  with  $L_k$  for  $k \geq 2$ . ■

Indeed, Enhancement 2 is derived from Lemma 6 as well. Because an item may be a terminal or an interior node in the taxonomy graph and the transactions in database  $D$  are composed of terminal items only, we have to perform ancestor-filtering first and then item-pruning; otherwise, we will lose the case though some items are not frequent, in contrast to their ancestors. Figure 4 shows an overview of the MMS\_Cumulate algorithm. The procedure for generating  $F$  is shown in Figure 5. Procedure  $\text{subset}(C_k, t)$  follows the description in [2] except that items in transaction  $t$  are inspected in ascending ordering of minimum supports, rather than in lexicographic ordering.

```

Create IMS; /* the table of user-defined minimum support */
Create IA; /* the table of each item and its ancestors from taxonomy T */
SMS = sort(IMS); /* ascending sort according to ms(a) stored in IMS */
F = F-gen(SMS, D, IA);
L1 = {a ∈ F | sup(a) ≥ ms(a)};
for ( k = 2; Lk-1 ≠ ∅; k++) do
    if k = 2 then C2 = C2-gen(F);
    else Ck = Ck-gen(Lk-1);
    Delete any candidate in Ck that consists of an item and its ancestor;
    Delete any ancestor in IA that is not present in any of the candidates in Ck;
    Delete any item in F that is not present in any of the candidates in Ck;
    for each transaction t ∈ D do
        for each item a ∈ t do
            Add all ancestors of a in IA into t;
            Remove any duplicates from t;
            Delete any item in t that is not present in F;
            Ct = subset(Ck, t);
            for each candidate A ∈ Ct do
                Increase the count of A;
        end for
        Lk = {A ∈ Ck | sup(A) ≥ ms(A[1])}; /* A[1] denote the first item in A */
    end for
Result = ∪k Lk;

```

**Fig. 4.** Algorithm MMS\_Cumulate.

```

for each transaction  $t \in D$  do
    for each item  $a \in t$  do
        Add all ancestors of  $a$  in  $IA$  into  $t$ ;
        Remove any duplicates from  $t$ ;
    for each item  $a \in t$  do
        Increase the count of  $a$ ;
    end for

for each item  $a$  in  $SMS$  in the same order do
    if  $sup(a) \geq ms(a)$  then
        Insert  $a$  into  $F$ ;
        break;
    end if
end for

for each item  $b$  in  $SMS$  that is after  $a$  in the same order do
    if  $sup(b) \geq ms(a)$  then insert  $b$  into  $F$ ;

```

**Fig. 5.** Procedure  $F\text{-gen}(SMS, D, IA)$ .

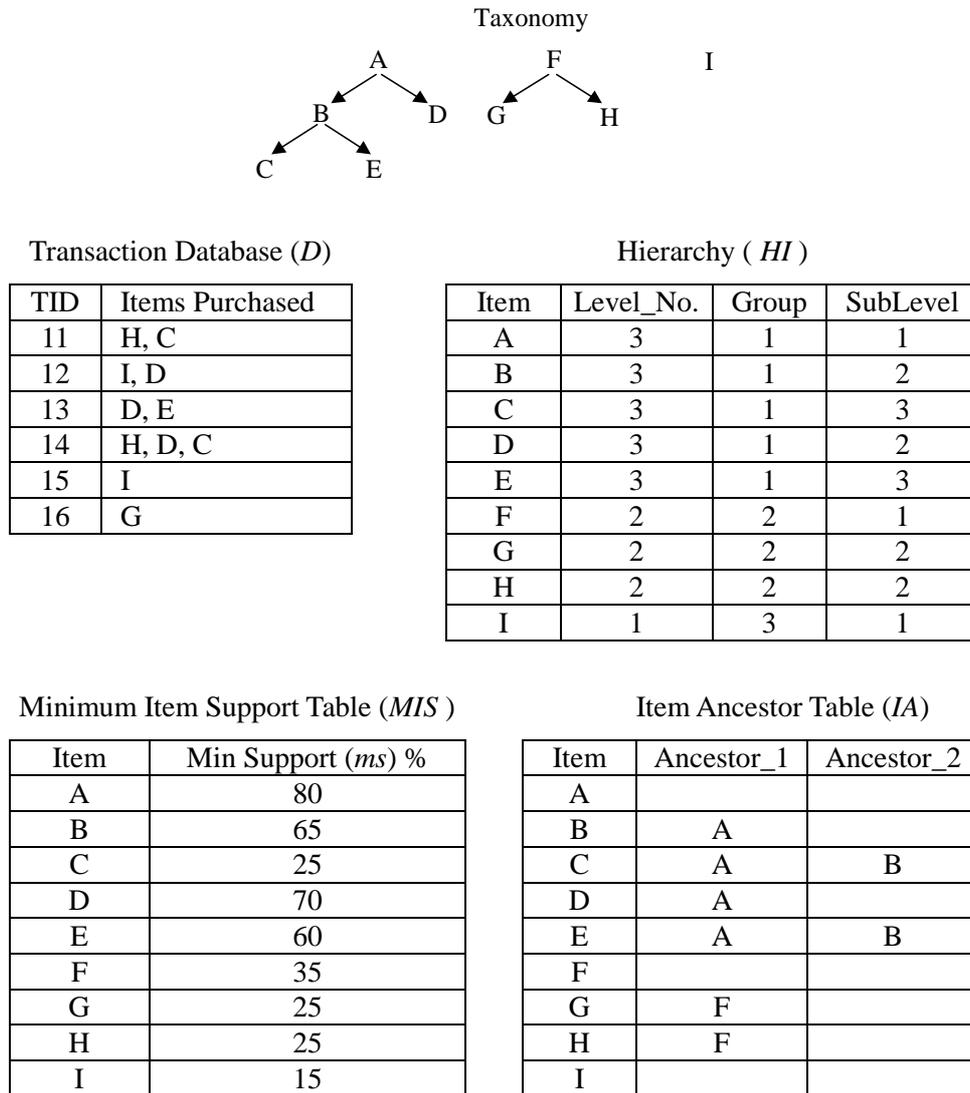
Tables 5 and 6 show the process of applying  $MMS\_Cumulate$  to the example shown in Figure 6. For simplicity, item “A” stands for “Printer”, “B” for “Non-impact printer”, “C” for “Laser printer”, “D” for “Dot-matrix printer”, “E” for “Ink-jet printer”, “F” for “PC”, “G” for “Desktop PC”, “H” for “Notebook”, and “I” for “Scanner” in the taxonomy.

### 3.3 Algorithm $MMS\_Stratify$

The stratification concept is introduced in [11]. It refers to a level-wise counting strategy from the top level of the taxonomy down to the lowest level, hoping that candidates containing items at higher levels will not have minimum support, thus there is no need to count candidates that include items at lower levels. However, this counting strategy may fail in the case of non-uniform minimum supports.

**Example 4.** Let  $\{\text{Printer, PC}\}$ ,  $\{\text{Printer, Desktop}\}$ , and  $\{\text{Printer, Notebook}\}$  be candidate itemsets to be counted. The taxonomy and minimum supports are defined as in Example 1. Using the level-wise strategy, we count first  $\{\text{Printer, PC}\}$  and assume that it is not frequent, i.e.,  $sup(\{\text{Printer, PC}\}) < 0.35$ . Because the minimum supports of  $\{\text{Printer, Desktop}\}$ , 0.25, and  $\{\text{Printer, Notebook}\}$ , also 0.25, are less than that of  $\{\text{Printer, PC}\}$ , we cannot assure that the occurrences of  $\{\text{Printer, Desktop}\}$  and  $\{\text{Printer, Notebook}\}$ , though less than  $\{\text{Printer, PC}\}$ , are also less than their minimum supports. In this case, we still have to count  $\{\text{Printer, Desktop}\}$  and  $\{\text{Printer, Notebook}\}$

even though {Printer, PC} does not have minimum support.



**Fig. 6.** An example of mining generalized association rules.

**Table 5.** Summary for Itemsets, Counts and Supports of *D* in Figure 6.

1-itemset	Counts	sup(%)	2-itemset	Counts	sup(%)	3-itemset	Counts	sup(%)
I	2	33.3	I, A	1	16.7	F, B, D	1	16.7
F	3	50.0	I, D	1	16.7	C, F, D	1	16.7
G	1	16.7	F, A	2	33.3	H, B, D	1	16.7
H	2	33.3	F, B	2	33.3	H, C, D	1	16.7
A	4	66.7	F, D	1	16.7			
B	3	50.0	C, F	2	33.3			
D	3	50.0	C, D	1	16.7			
C	2	33.3	H, A	2	33.3			
E	1	16.7	H, B	2	33.3			
			H, C	2	33.3			
			H, D	1	16.7			
			B, D	2	33.3			
			E, D	1	16.7			

**Table 6.** Running summary of MMS\_Cumulate on Figure 6.

Item	$ms(\%)$	$sup(\%)$	$F$	$L_1$	$C_2$	$L_2$	$C_3 \& L_3$
I	15	33.3	I	I	I, C	I, D	$\emptyset$
C	25	33.3	C	C	I, G	I, A	
G	25	16.7	G	H	I, H	C, H	
H	25	33.3	H	F	I, F	C, F	
F	35	50.0	F		I, E	H, B	
E	60	16.7	E		I, B	H, A	
B	65	50.0	B		I, D		
D	70	50.0	D		I, A		
A	80	66.7	A		C, H		
					C, F		
					C, D		
					H, B		
					H, D		
					H, A		
					F, B		
					F, D		
					F, A		

The following observation inspires us to deploy the MMS\_Stratify algorithm.

**Lemma 7.** Consider two  $k$ -itemsets  $\langle a_1, a_2, \dots, a_k \rangle$  and  $\langle a_1, a_2, \dots, \hat{a}_k \rangle$ , where  $\hat{a}_k$  is an ancestor of  $a_k$ .

If  $\langle a_1, a_2, \dots, \hat{a}_k \rangle$  is not frequent, then neither is  $\langle a_1, a_2, \dots, a_k \rangle$ .

**Proof.** Note that  $ms(\langle a_1, a_2, \dots, a_k \rangle) = ms(\langle a_1, a_2, \dots, \hat{a}_k \rangle) = ms(a_1)$ . Because  $sup(a_k) \leq sup(\hat{a}_k)$ , if  $\langle a_1, a_2, \dots, \hat{a}_k \rangle$  is not frequent we can derive

$$sup(\langle a_1, a_2, \dots, a_k \rangle) \leq sup(\langle a_1, a_2, \dots, \hat{a}_k \rangle) < ms(\langle a_1, a_2, \dots, \hat{a}_k \rangle) = ms(\langle a_1, a_2, \dots, a_k \rangle),$$

which completes the proof. ■

Lemma 7 implies that if a sorted candidate itemset in a higher level of the taxonomy is not frequent, then neither are all of its descendants that differ from the itemset only in the last item. Note that we do not make any relative assumption about the minimum supports of the item  $a_k$  and its ancestor  $\hat{a}_k$ . This means that the claim in Lemma 7 applies to any specifications of  $ms(a_k) = ms(\hat{a}_k)$  (corresponding to uniform case),  $ms(a_k) < ms(\hat{a}_k)$ , or  $ms(a_k) > ms(\hat{a}_k)$  (not ordinary case). As will become clear later, this makes our counting strategy applicable to any user-specified minimum item support specification.

We first divide  $C_k$ , according to the ancestor-descendant relationship claimed in Lemma 7, into two disjoint subsets, called *top candidate set*  $TC_k$  and *residual candidate set*  $RC_k$ , defined below.

**Definition 8.** Consider a set,  $S_k$ , of candidates in  $C_k$  induced by the schema  $\langle a_1, a_2, \dots, a_{k-1}, * \rangle$ , where

'\*' denotes don't care. A candidate  $k$ -itemset  $A = \langle a_1, a_2, \dots, a_{k-1}, a_k \rangle$  is a top candidate if none of the candidates in  $S_k$  is an ancestor of  $A$ . That is,

$$TC_k = \{A \mid A \in C_k, \neg(\exists \bar{A} \in C_k \text{ and } A[i] = \bar{A}[i], 1 \leq i \leq k-1, \bar{A}[k] \text{ is an ancestor of } A[k])\},$$

and

$$RC_k = C_k - TC_k.$$

**Example 5.** Assume that the candidate 2-itemset  $C_2$  for Example 1 consists of  $\langle \text{Scanner}, \text{PC} \rangle$ ,  $\langle \text{Scanner}, \text{Desktop} \rangle$ ,  $\langle \text{Scanner}, \text{Notebook} \rangle$ ,  $\langle \text{Notebook}, \text{Laser} \rangle$ ,  $\langle \text{Notebook}, \text{Non-impact} \rangle$ , and  $\langle \text{Notebook}, \text{Dot matrix} \rangle$ , and the supports of the items in higher levels are larger than those in lower levels. Then  $TC_2 = \{\langle \text{Scanner}, \text{PC} \rangle, \langle \text{Notebook}, \text{Non-impact} \rangle, \langle \text{Notebook}, \text{Dot matrix} \rangle\}$  and  $RC_2 = \{\langle \text{Scanner}, \text{Desktop} \rangle, \langle \text{Scanner}, \text{Notebook} \rangle, \langle \text{Notebook}, \text{Laser} \rangle\}$ .

Our approach is that, in each pass  $k$ , rather than counting all candidates in  $C_k$  as in `MMS_Cumulate`, we count the supports for candidates in  $TC_k$ . Then those that do not have minimum supports are deleted along with their descendants in  $RC_k$ . If  $RC_k$  is not empty, we then perform an extra scan over the transaction database  $D$  to count the remaining candidates in  $RC_k$ . Again, the less frequent candidates are eliminated. The resulting  $TC_k$  and  $RC_k$ , called  $TL_k$  and  $RL_k$  respectively, form the set of frequent  $k$ -itemsets  $L_k$ . An overview of the `MMS_Stratify` algorithm is described in Figure 7. The enhancements used in `MMS_Cumulate` apply to this algorithm as well. The procedures for generating  $TC_k$  and  $RC_k$  are given in Figures 8 and 9 respectively, where  $HI$  denotes the `hierarchy_item` relation. Table 7 shows the progressing result of applying `MMS_Stratify` to the example in Figure 6.

**Example 6.** Let us continue with Example 5. If the support of the top level candidate does not pass its minimum support  $ms(\text{Scanner})$ , we do not need to count the remaining descendant candidates  $\langle \text{Scanner}, \text{Desktop} \rangle$ ,  $\langle \text{Scanner}, \text{Notebook} \rangle$ . On the contrary, if  $\langle \text{Scanner}, \text{Desktop} \rangle$  is a frequent itemset, we should perform another pass over  $D$  to count  $\langle \text{Scanner}, \text{Desktop} \rangle$  and  $\langle \text{Scanner}, \text{Notebook} \rangle$  to determine whether they are frequent or not.

### 3.4 Analytical comparison of `MMS_Cumulate` and `MMS_Stratify`

In this subsection, we will compare the proposed two algorithms in terms of their complexity. Rather than deriving the computation cost to accomplish the whole task, we confine ourselves to the main step: Given the current set of frequent  $k$ -itemset  $L_k$ , what is the cost to generate the set of frequent  $k+1$ -itemsets  $L_{k+1}$ ?

Recall that for Apriori-like algorithms, the set of candidate  $k+1$ -itemsets  $C_{k+1}$  are formed from performing itemset joining over  $L_k$ , followed by some pruning strategies. Then the support of each

```

Create IMS;
Create HI; /* the table of each item with Hierarchy Level, Sublevel, and Group */
Create IA;
SMS = sort(IMS);
F = F-gen(SMS, D, IA);
L1 = {a ∈ F | sup(a) ≥ ms(a)};
for (k = 2; Lk-1 ≠ ∅; k++) do
    if k = 2 then C2 = C2-gen(F)
    else Ck = Ck-gen(Lk-1);
    Delete any candidate in Ck that consists of an item and its ancestor;
    Delete any ancestor in IA that is not present in any of the candidates in Ck;
    Delete any item in F that is not present in any of the candidates in Ck;
    TCk = TCk-gen(Ck, HI); /* Using Ck, HI to find top Ck */
    for each transaction t ∈ D do
        for each item a ∈ t do
            Add all ancestors of a in IA into t;
            Remove any duplicates from t;
            Delete any item in t that is not present in F;
            Ct = subset(TCk, t);
            for each candidate A ∈ Ct do
                Increase the count of A;
        end for
        TLk = {A ∈ TCk | sup(A) ≥ ms(A[1])};
        RCk = RCk-gen(Ck, TCk, TLk); /* Using Ck, TLk to find residual Ck */
        if RCk ≠ ∅ then
            Delete any ancestor in IA that is not present in any of the candidates in RCk;
            Delete any item in F that is not present in any of the candidates in RCk;
            for each transaction t ∈ D do
                for each item a ∈ t do
                    Add all ancestors of a in IA into t;
                    Remove any duplicates from t;
                    Delete any item in t that is not present in F;
                    Ct = subset(RCk, t);
                    for each candidate A ∈ Ct do
                        Increase the count of A;
                    end for
                RLk = {A ∈ RCk | sup(A) ≥ ms(A[1])};
            end if
            Lk = TLk ∪ RLk;
        end for
    end for
Result = ∪k Lk;

```

**Fig. 7.** Algorithm MMS\_Stratify.

**for** each itemset  $A \in C_k$  **do**  
Sort  $C_k$  according to  $HI$  of  $A[k]$  preserving the ordering of  $A[1]A[2]...A[k-1]$ ;  
**for** itemset  $A \in C_k$  in the same order **do**  
 $S_k = \{ \bar{A} \mid \bar{A} \in C_k \text{ and } A[i] = \bar{A}[i], 1 \leq i \leq k-1 \}$ ;  
**if**  $A$  is not marked and none of the candidates in  $S_k$  is an ancestor of  $A[k]$  **then**  
Insert  $A$  into  $TC_k$ ;  
Mark  $A$  and all of its descendants in  $S_k$ ;  
**end if**  
**end for**

**Fig. 8.** Procedure  $TC_k$ -gen( $C_k, HI$ ).

**for** each itemset  $A \in TC_k$  **do**  
**if**  $A \notin TL_k$  **then**  
 $S_k = \{ \bar{A} \mid \bar{A} \in C_k \text{ and } A[i] = \bar{A}[i], 1 \leq i \leq k-1 \}$ ;  
Insert all of its descendants in  $S_k$  into  $RC_k$ ;  
**end if**

**Fig. 9.** Procedure  $RC_k$ -gen( $C_k, TC_k, TL_k$ ).

**Table 7.** Running summary of MMS\_Stratify on Figure 6.

Item	$ms(\%)$	$sup(\%)$	$F$	$L_1$	$C_2$	Sort $C_2$	$TC_2$	$TL_2$	$RC_2$	$RL_2$	$L_2$	$C_3 \& L_3$
I	15	33.3	I	I	I, G	I, A	I, A	I, A	I, B	I, D	I, D	$\emptyset$
G	25	16.7	G	H	I, H	I, B	I, F	H, A	I, D	H, B	I, A	
H	25	33.3	H	C	I, C	I, D	H, A	C, F	I, C	H, C	H, C	
C	25	33.3	C	F	I, F	I, C	C, D		I, E		H, B	
F	35	50.0	F		I, E	I, E	C, F		H, B		H, A	
E	60	16.7	E		I, B	I, F	F, A		H, D		C, F	
B	65	50.0	B		I, D	I, G			H, C			
D	70	50.0	D		I, A	I, H						
A	80	66.7	A		H, C	H, A						
					H, B	H, B						
					H, D	H, D						
					H, A	H, C						
					C, F	C, D						
					C, D	C, F						
					F, B	F, A						
					F, D	F, B						
					F, A	F, D						

candidate itemset is counted by scanning the transaction database. In this context, the primary computation involves database scanning and support counting of the candidate itemsets. Let  $\alpha$  be the

ratio of the cost for scanning a transaction from the database to that for counting the support of an itemset. The cost for the step of concern is  $|C_{k+1}| + \alpha |D|$  for MMS\_Cumulate, while for MMS\_Stratify it is  $|TC_{k+1}| + |RC_{k+1}^*| + 2\alpha |D|$ , where  $RC_{k+1}^*$  denotes, after performing the stratification pruning, the set of remaining candidates in  $RC_{k+1}$ . Let  $k_{\max}$  be the maximal cardinality of frequent itemsets. Then the total cost difference of the two algorithms will be

$$\begin{aligned} & \sum_{k=2, k_{\max}} [(|C_k| + \alpha |D|) - (|TC_k| + |RC_k^*| + 2\alpha |D|)] \\ &= \sum_{k=2, k_{\max}} (|RC_k| - |RC_k^*|) - (k_{\max} - 1)\alpha |D| \end{aligned} \quad (6)$$

Note that  $|RC_k| - |RC_k^*|$  denotes the number of candidates in  $RC_k$  pruned by the stratification strategy. Thus, the result in Eq. 6 means that the superiority of MMS\_Stratify over MMS\_Cumulate depends on whether the cost reduced by the stratification pruning can compensate for that spent on extra scanning of the database, which will be seen later in the experiments.

## 4. Experiments

In this section, we evaluate the performance of algorithms, MMS\_Cumulate and MMS\_Stratify, using two synthetic datasets, named Synth1 and Synth2, generated by the IBM data generator [2, 11], and Microsoft foodmart2000 database (Foodmart for short), a sample supermarket data warehouse provided in MS SQL 2000. The parameter settings are shown in Table 8. The data for Foodmart is drawn from sales\_fact\_1997, sales\_fact\_1998 and sales\_fact\_dec\_1998 in foodmart2000. The corresponding item taxonomy consists of three levels: There are 1560 primitive items in the first level (product), 110 generalized items in the second level (product\_subcategory), and 47 generalized items in the top level (product\_category). All experiments were performed on an Intel Pentium-IV 2.80GHz with 2GB RAM, running on Windows 2000.

**Table 8.** Parameter settings for synthetic dataset and foodmart2000.

Parameter		Default value		
		Synth1	Synth2	Foodmart
$ DB $	Number of original transactions	100,000	200,000	62,568
$ t $	Average size of transactions	12	16	12
$ I $	Average size of the maximal potentially frequent itemsets	3	3	3
$N$	Number of items including generalized items	132	231	1,717
$R$	Number of groups	30	30	47
$L$	Number of levels	3	3	3
$f$	Fanout	5	5	14

We first compared the execution times of MMS\_Cumulate and MMS\_Stratify with the Cumulate and Stratify algorithms presented in [11] for different uniform minimum supports, ranging from 0.5%

to 3.5% for Synth1 and Synth2, respectively, and 0.05% to 0.35% for Foodmart. The results are shown in Figures 10 to 12. It can be observed that in uniform support constraint, MMS\_Cumulate behaves like its uniform counterpart, Cumulate. However, the situation is somewhat different for MMS\_Stratify and Stratify, particular in the real dataset Foodmart, where MMS\_Stratify overwhelms Stratify. This is because the definition of ancestor-descendant relationship in MMS\_Stratify is more rigorous than that in Stratify. Consequently, during the execution of Stratify, many itemset groups that satisfy the relationship are identified but most of them are frequent. Therefore, as the smaller the support constraint is, the larger the amount of top itemsets identified are frequent, and the longer the amount of time spent on the stratification pruning is wasted. This also explains why Cumulate and MMS\_Cumulate perform better than Stratify and MMS\_Stratify.

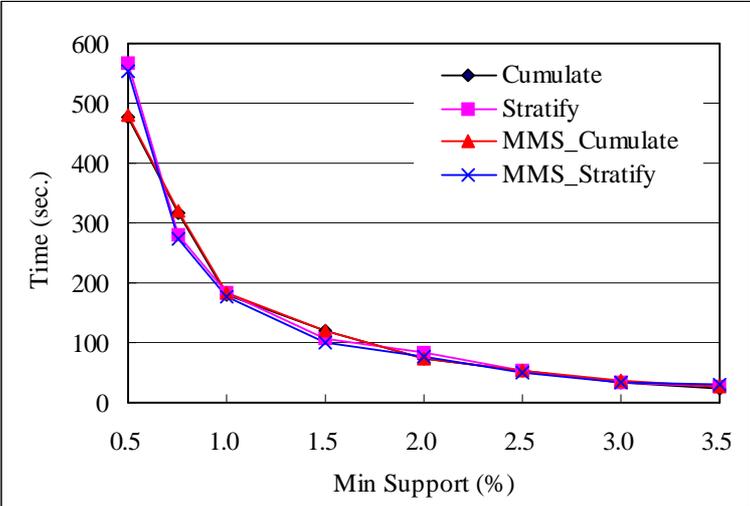


Fig. 10. Execution times for various minimum supports for Synth1.

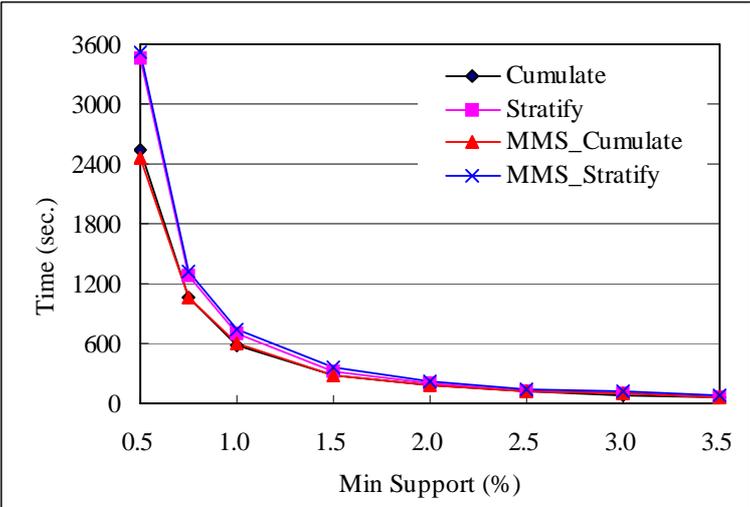
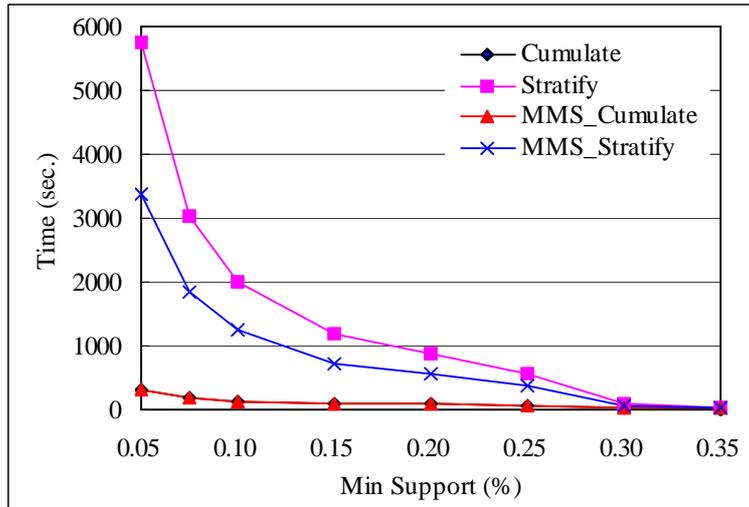
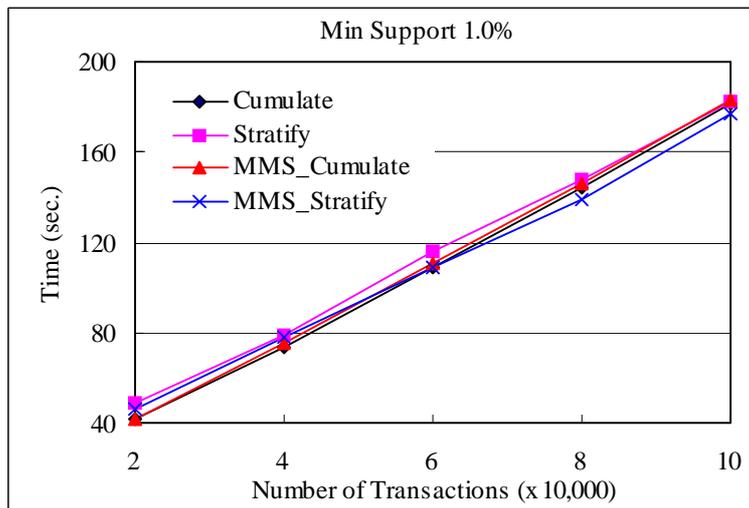


Fig. 11. Execution times for various minimum supports for Synth2.

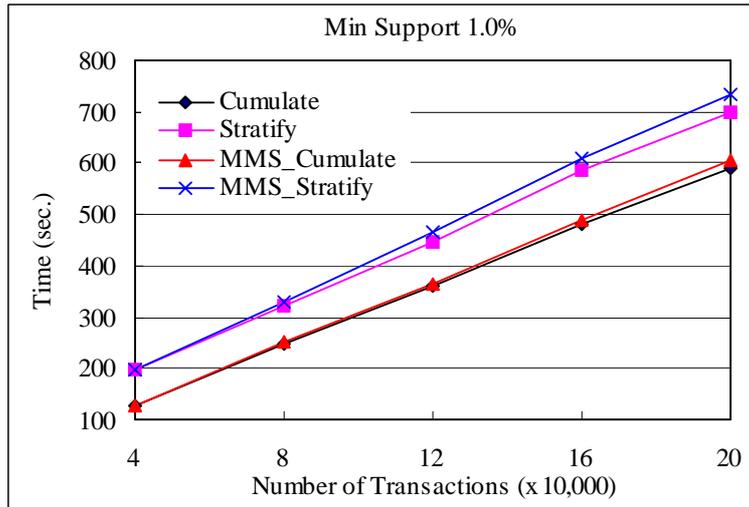


**Fig. 12.** Execution times for various minimum supports for Foodmart.

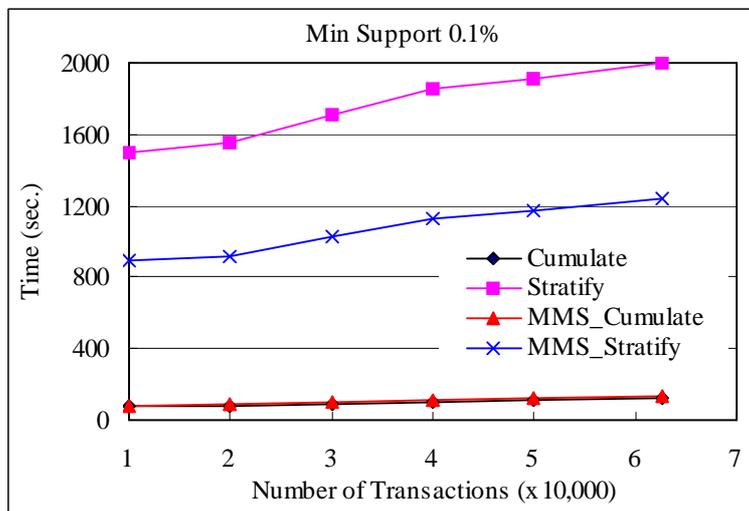
We then compared the scalability of the algorithms under varying transaction sizes at uniform  $minsup = 1.0\%$  for Synth1 and Synth2, and  $minsup = 0.1\%$  for Foodmart. The results are shown in Figures 13 to 15. As can be seen, all algorithms exhibit good linear scalability. Note that the picture shown in Figure 13 is somewhat different from the others. In particular, MMS\_Stratify takes the lead when the transaction size is over 60,000. This is because in this case MMS\_Stratify can prune much more candidates through the restricted ancestor-descendant relationship to lessen the work spent on database scanning for support counting.



**Fig. 13.** Transactions scale-up for uniform minimum support with  $minsup = 1.0\%$  for Synth1.



**Fig. 14.** Transactions scale-up for uniform minimum support with  $minsup = 1.0\%$  for Synth2.



**Fig. 15.** Transactions scale-up for uniform minimum support with  $minsup = 0.1\%$  for Foodmart.

The efficiency of MMS\_Cumulate and MMS\_Stratify were then compared for multiple minimum supports. Three different specifications discussed in Section 3.2 were examined. They are 1) the support of each item was assigned randomly (*random specification*); 2) the support of each item was assigned according to Eq. 1 (*normal specification*); and 3) the support of each item is assigned using Eq. 5 (*CLS specification*). Note that all of these specifications satisfied the ordinary case that the minimum support of an item  $a$  is not larger than that of any of its ancestors  $\hat{a}$ , i.e.,  $ms(a) \leq ms(\hat{a})$ . This assumption conforms to the fact that the support of an item in the database is less than that of its ancestors. In order to speed up the mining time for *normal specification* and *CLS specification*, when  $ms(a)$  is less than 0.5%, we set  $ms(a) = 0.5\%$  for Synth1 and Synth2, and when  $ms(a)$  is less than 0.05%, we set  $ms(a) = 0.05\%$  for Foodmart.

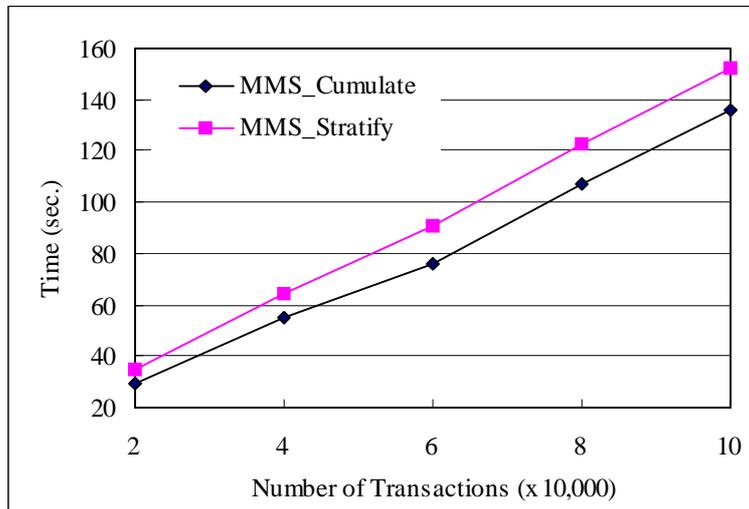
For random specification, minimum supports ranging from 0.1% to 6.0% for Synth1 and Synth2 and from 0.011% to 6.0% for Foodmart were specified to items randomly, with items in higher levels

of the taxonomy receiving larger values. The results are shown in Figures 16 to 18. Algorithm MMS\_Cumulate performs better than MMS\_Stratify, with the gap increasing as the number of transactions increases. The results also display the scalability of the algorithms. Both MMS\_Cumulate and MMS\_Stratify exhibit linear scale-up with the number of transactions.

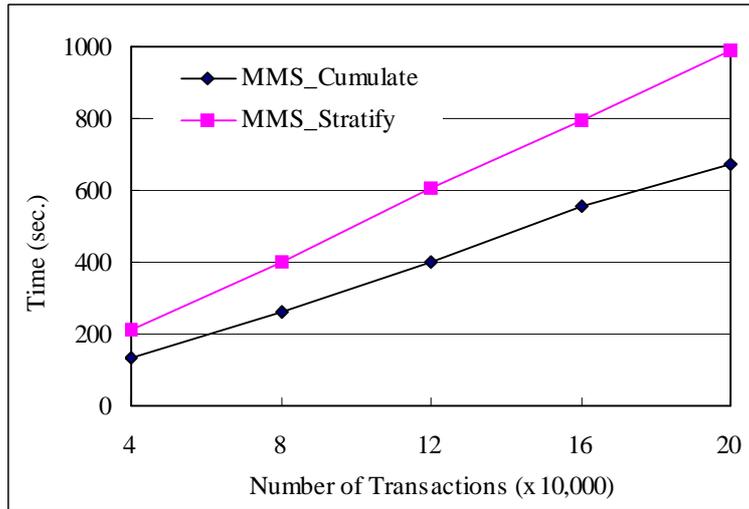
Figures 19 to 21 show the results for the normal specification. Overall, MMS\_Cumulate performs better than MMS\_Stratify, but the gap decreases as  $\alpha$  is getting larger. However, in Figure 19, MMS\_Stratify performs slightly worse than MMS\_Cumulate for  $\alpha < 0.3$  but overwhelms MMS\_Cumulate when  $\alpha$  is larger than 0.3. This is because we have set  $ms(a) = 0.5\%$  when  $ms(a)$  is less than 0.5%, causing many top itemsets to be infrequent. Therefore, MMS\_Stratify can prune much more low-level candidates.

For CLS specification, similar results can be observed, as shown in Figures 22, 23 and 24.

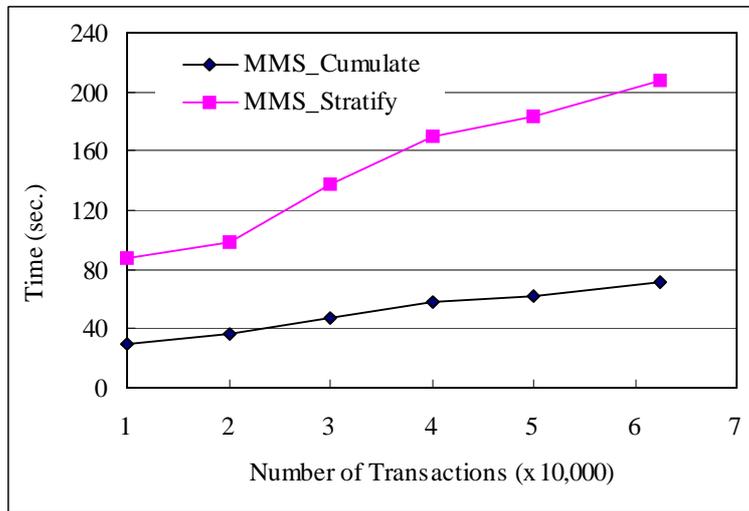
Finally, we examined the capability of the proposed methods in finding informative rules with small and non-uniform supports. To this end, we chose some items of high values but with relative small supports in Foodmart, as shown in Table 9, and tried to find informative rules composed of these items. Note that the item “Cereal” is a generalized item while the others are primitive items. We have used the following setting. The minimum supports were set as  $ms(\text{Pleasant Canned Yams}) = 0.005\%$ ,  $ms(\text{Gorilla Mild Cheddar Cheese}) = 0.25\%$ ,  $ms(\text{Carrington Ice Cream}) = 0.25\%$ ,  $ms(\text{CDR Apple Preserves}) = 0.002\%$ ,  $ms(\text{Giant Small Brown Eggs}) = 0.25\%$ ,  $ms(\text{Cereal}) = 3\%$ , and was randomly set within 0.2% to 6% for other items. The minimum confidence was set to be 60%.



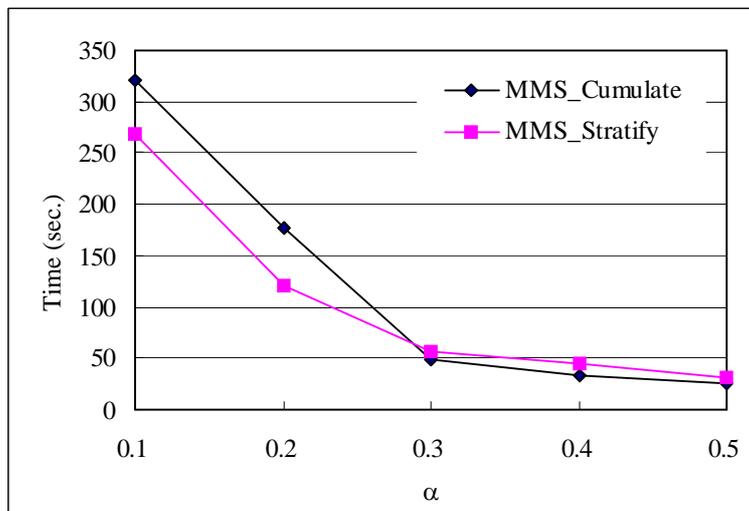
**Fig. 16.** Execution time for random support specification for Synth1.



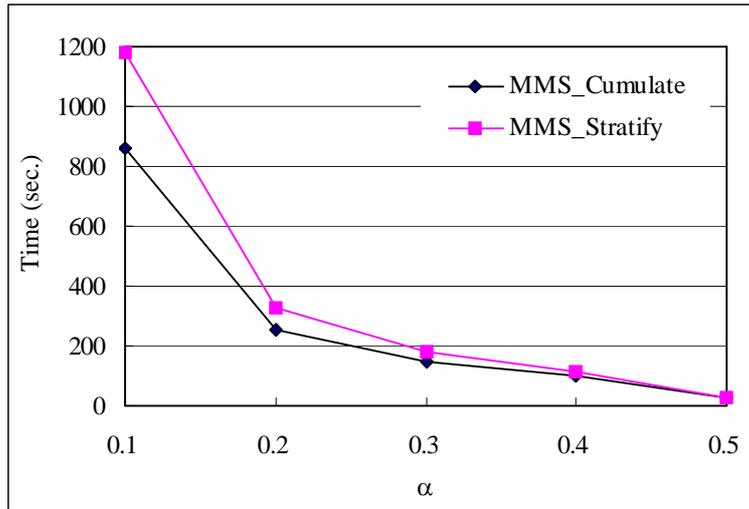
**Fig. 17.** Execution time for random support specification for Synth2.



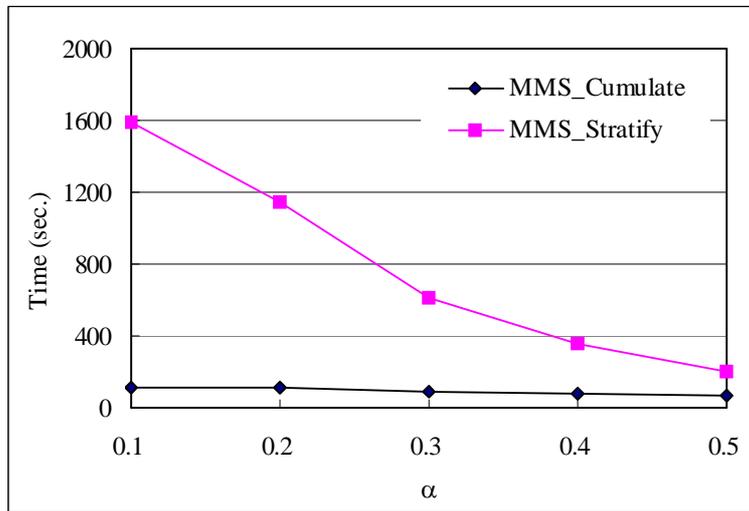
**Fig. 18.** Execution time for random support specification for Foodmart.



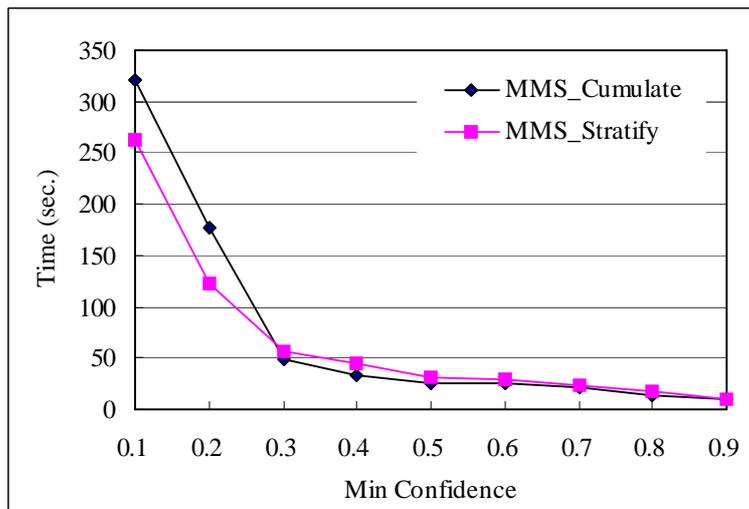
**Fig. 19.** Execution time for normal support specification for Synth1.



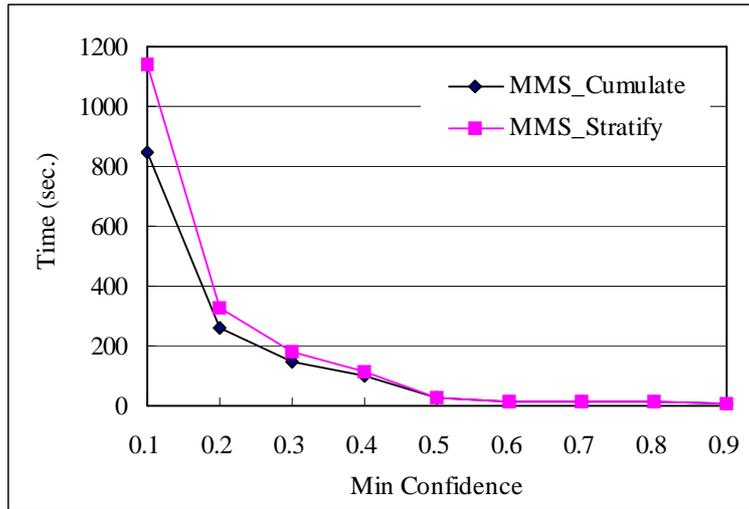
**Fig. 20.** Execution times for normal support specification for Synth2.



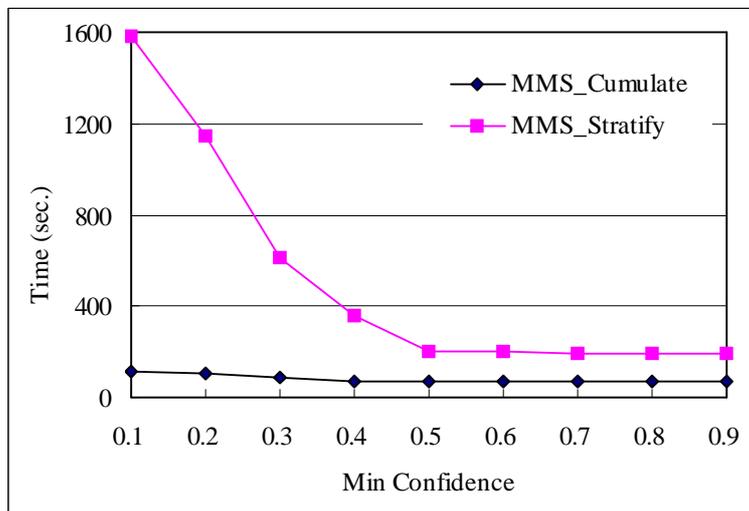
**Fig. 21.** Execution times for normal support specification for Foodmart.



**Fig. 22.** Execution time for CLS support specification for Synth1.



**Fig. 23.** Execution time for CLS support specification for Synth2.



**Fig. 24.** Execution time for CLS support specification for Foodmart.

**Table 9.** Selected high-value items with small supports from Foodmart.

Itemset	Support
Pleasant Canned Yams	0.237%
Carrington Ice Cream	0.272%
Gorilla Mild Cheddar Cheese	0.328%
CDR Apple Preserves	0.128%
Giant Small Brown Eggs	0.257%
Cereal	5.391%

Some interesting rules were discovered, but for simplicity, we only show four of them. The first two discovered rules were generated from the frequent itemset {Pleasant Canned Yams, Carrington Ice Cream, Gorilla Mild Cheddar Cheese} as shown below:

Pleasant Canned Yams, Carrington Ice Cream  $\Rightarrow$  Gorilla Mild Cheddar Cheese ( $sup = 0.008\%$ ,  $conf = 83.3\%$ ,  $lift = 306.7$ )

and

Pleasant Canned Yams, Gorilla Mild Cheddar Cheese  $\Rightarrow$  Carrington Ice Cream ( $sup = 0.008\%$ ,  $conf = 100\%$ ,  $lift = 305.2$ ).

Another two were generated from the frequent itemset {CDR Apple Preserves, Giant Small Brown Eggs, Cereal}:

CDR Apple Preserves, Giant Small Brown Eggs  $\Rightarrow$  Cereal ( $sup = 0.003\%$ ,  $conf = 66.7\%$ ,  $lift = 12.4$ )

and

CDR Apple Preserves, Cereal  $\Rightarrow$  Giant Small Brown Eggs ( $sup = 0.003\%$ ,  $conf = 50\%$ ,  $lift = 194.3$ ).

All of these rules exhibit high confidence and positive implication. In reality, these rules reveal that the promotion of some item combinations, e.g., Pleasant Canned Yams and Gorilla Mild Cheddar Cheese, is very likely to raise the sales of some particular items, e.g., Carrington Ice Cream. Simple though this example is, it has illustrated the ability of our methods in finding very rare but informative rules.

## 5. Related Work

The problem of mining association rules in the presence of taxonomy information was addressed first in [5] and [11]. In [11], the problem is named “mining generalized association rules,” which aims to find associations between items at any level of the taxonomy under the *minsup* and *minconf* constraints. Their work, however, did not recognize the varied support requirements inherent in items at different hierarchy levels.

In [5], the problem was stated somewhat different from that in [11]. They generalized the uniform minimum support constraint into a form of level-wise assignment, i.e., items at the same level receive the same minimum support. The objective was mining associations level-by-level in a fixed hierarchy. That is, only associations between items at the same level were examined progressively from the top level to the bottom.

Another form of association rules involving mining with multiple minimum supports was proposed in [7]. Their method allows users to specify different minimum support for different items and can find rules involving both frequent and rare items. However, their model considers no taxonomy at all and hence fails to find associations between items at different hierarchy levels.

To our knowledge, [5] is the only work considering both aspects of item taxonomy and multiple supports. However, their intention was quite different from ours. First, although several variants were proposed, all of them follow a level-wise, progressively deepening strategy that performs a top-down traversing of the taxonomy to generate all frequent itemsets. An Apriori-like algorithm is applied at each level, which leads to  $p$  database scans, where  $p = \sum_l k_l$  and  $k_l$  is the maximum  $k$ -itemset at level  $l$ . This is quite a large overhead compared with our algorithm, which requires only  $\max_l k_l$  times. Second, the minimum supports are specified uniform at each taxonomy level, that is, items at the same taxonomy level receive the same minimum support. This restrains the flexibility and power of association rules. Furthermore, together with the progressively deepening strategy, their approaches would fail to discover all frequent itemsets, especially those involving level-crossing associations. Let us illustrate this with an example, and for self-explanatory demonstration, a generic description of their approaches is given in Figure 25.

```

D: a taxonomy-information-encoded transaction database;
minsup[l]: the minimum support threshold for each concept level l;
for (l = 1; L[l, 1] ≠ ∅ and l < max_level; l++) do
    L[l, 1] = the frequent 1-itemsets at level l;
    for (k = 2; L[l, k - 1] ≠ ∅; k++) do
        Ck = apriori-gen(L[l, k - 1]);
        for each transaction t ∈ D do
            Ct = subset(Ck, t);
            for each candidate A ∈ Ct do increase the count of A;
        endfor
        L[l, k] = {A ∈ Ck | sup(A) ≥ minsup[l]};
    endfor
    LL[l] = ∪k L[l, k]; /* LL[l]: the set of frequent itemsets at level l */
end for
Result = ∪l LL[l];

```

**Fig. 25.** A generic description of multi-level association mining algorithms presented in [5].

**Example 7.** Consider the example used in [5], as shown in Figure 26, where the minimum support is set to be 4 at level 1, and 3 at levels 2 and 3. Each item is encoded as a sequence of digits, representing its positions in the taxonomy. For example, the item ‘White Old Mills Bread’ is encoded as ‘211’ in which the first digit, ‘2’, represents ‘bread’ at level-1, the second ‘1’ for ‘White’ at level 2, and the third ‘2’ for ‘Old Mills’ at level 3. To discover all frequent itemsets, the proposed algorithms first apply the Apriori algorithm to  $T$ , generating all level-1 frequent itemsets. The result is

$$L[1, 1] = \{\{1 **\}, \{2 **\}\},$$

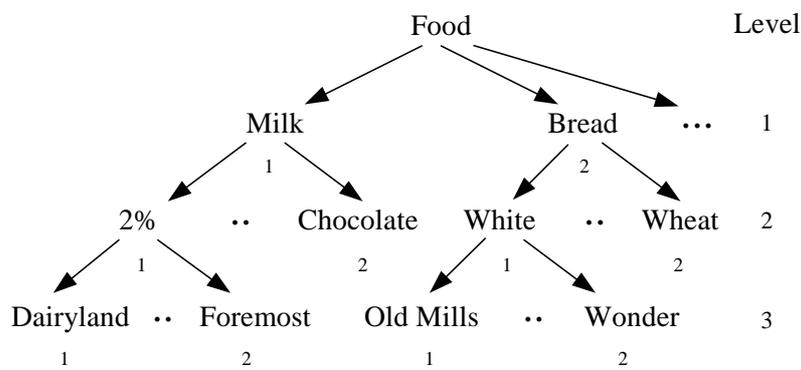
and

$$L[1, 2] = \{\{1 **, 2 **\}\}.$$

According to the level-wise deepening paradigm, only descendants of the frequent itemsets at level-1 are inspected to generate frequent itemsets at level-2. The resulting level-2, frequent 1-itemset is

$$L[2, 1] = \{\{11 *\}, \{12 *\}, \{21 *\}, \{22 *\}\}.$$

Note that  $\{32 *\}$  and  $\{41 *\}$  are missed in  $L[2, 1]$  though they are frequent. For the same reason,  $\{323\}$  is missed in  $L[3, 1]$  and so are level-crossing frequent itemsets  $\{111, 12 *\}, \{12 *, 221\}, \{11 *, 12 *, 221\}$ .



Encoded table  $\mathcal{D}$

<i>tid</i>	Items Purchased
11	{111, 121, 211, 221}
12	{111, 211, 222, 323}
13	{112, 122, 221, 411}
14	{111, 121}
15	{111, 122, 211, 221, 413}
16	{211, 323, 524}
17	{323, 411, 524, 713}

**Fig. 26.** The example of taxonomy and encoded transaction table in [5].

## 6. Conclusions

We have investigated in this paper the problem of mining generalized association rules in the presence of taxonomy and multiple minimum support specification. The classic Apriori itemset generation works in the presence of taxonomy but fails in the case of non-uniform minimum supports. We presented two algorithms, MMS\_Cumulate and MMS\_Stratify, for discovering these generalized frequent itemsets. Empirical evaluation showed that these two algorithms are very effective and have good linear scale-up characteristic. Between the two algorithms, MMS\_Stratify performed slightly

better than MMS\_Cumulate, with the gap increasing with the problem size, such as the number of transactions and/or candidate itemsets. As for the specification for non-uniform, multiple item supports, we also presented a confidence-lift specification, which is beneficial for discovering less-supported but perceptive rules without suffering from combinatorial explosion.

## References

- [1] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases, in: Proc. 1993 ACM-SIGMOD Int. Conf. on Management of Data, (Washington, D.C., 1993) 207-216.
- [2] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in: Proc. 20th Int. Conf. on Very Large Data Bases, (Santiago, Chile, 1994) 487-499.
- [3] S. Brin, R. Motwani, and C. Silverstein, Beyond market baskets: generalizing association rules to correlations, in: Proc. 1997 ACM-SIGMOD Int. Conf. on Management of Data, (1997) 207-216.
- [4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market-basket data," in: Proc. 1997 ACM-SIGMOD Int. Conf. on Management of Data, (1997) 207-216.
- [5] J. Han and Y. Fu, Discovery of multiple-level association rules from large databases, in: Proc. 21st Int. Conf. on Very Large Data Bases, (Zurich, Switzerland, 1995) 420-431.
- [6] W. Y. Lin, M. C. Tseng, and J. H. Su, "A confidence-lift support specification for interesting associations mining," in: Proc. 6th Pacific Area Conference on Knowledge Discovery and Data Mining (PAKDD-2002), Taipei, Taiwan, R.O.C., May 2002.
- [7] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in: Proc. 1999 Int. Conf. on Knowledge Discovery and Data Mining, (San Deige, CA, 1999) 337-341.
- [8] B. Lin, W. Hsu, and Y. Ma, Pruning and summarizing the discovered association, in: Proc. 1999 ACM-SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. (San Diego, CA, 1999) 125-134.
- [9] J. S. Park, M. S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," in: Proc. 1995 ACM-SIGMOD Int. Conf. on Management of Data, (San Jose, CA 1995) 175-186.
- [10] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases," in: Proc. 21st Int. Conf. on Very Large Data Bases, (Zurich, Switzerland, 1995) 432-444.
- [11] R. Srikant and R. Agrawal, "Mining generalized association rules," Future Generation Computer Systems, Volume 13, Issues 2-3, November 1997, 161-180.