Maintenance of Generalized Association Rules with Multiple Minimum Supports

Ming-Cheng Tseng[1] and Wen-Yang Lin[2]

[1](corresponding author)

Institute of Information Engineering, I-Shou University, Kaohsiung 840, Taiwan

Tel: +886-7-3635377

E-mail: clark.tseng@msa.hinet.net

[2]Dept. of Information Management, I-Shou University, Kaohsiung 840, Taiwan

E-mail: wylin@isu.edu.tw

Abstract

Mining generalized association rules among items in the presence of taxonomy 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 items within the same taxonomy level. This constraint would restrain an expert from discovering more interesting but much less supported association rules. In our previous work, we have addressed this problem and proposed two algorithms, MMS_Cumulate and MMS_Stratify. In this paper, we examined the problem of maintaining the discovered multi-supported, generalized association rules when new transactions are added into the original database. We proposed two algorithms, UD_Cumulate and UD_Stratify, which can incrementally update the discovered generalized association rules with non-uniform support specification and are capable of

effectively reducing the number of candidate sets and database re-scanning. Empirical evaluation showed that UD_Cumulate and UD_Stratify are 2-6 times faster than running MMS_Cumulate or MMS_Stratify on the updated database afresh.

Keywords: Generalized association rules, maintenance, multiple minimum supports, sorted closure, taxonomy.

1. Introduction

Mining association rules from a large database of business data, such as transaction records, has been a popular topic within the area of data mining [1, 2, 11, 13]. This problem is originally motivated by applications known as market basket analysis to find relationships among items purchased by customers, that is, the kinds of products tend to be purchased together. For example, an association rule,

$$\text{Desktop} \Rightarrow \text{Ink-jet}\ (\text{Support}=30\%, \text{Confidence}=60\%),$$

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. Such information is useful in many aspects of market management, such as store layout planning, target marketing, and understanding the behavior customers.

In many applications, there are taxonomies (hierarchies), explicitly or implicitly, over the items. In some applications, it may be more useful to find associations at different levels of the taxonomy than only at the primitive concept level [7, 14]. For example, consider Figure 1, the taxonomy of items from which the previous association rule is derived can be represented as

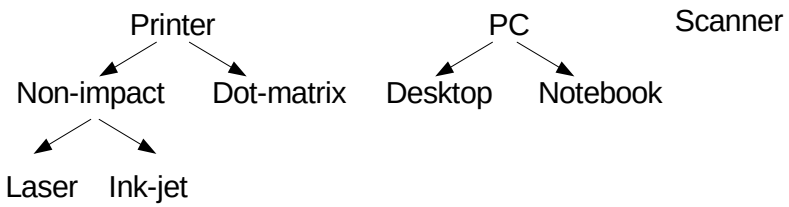


Figure 1. An example of taxonomy *T*.

It is likely to happen that the association rule

$$\text{Desktop} \Rightarrow \text{Ink-jet} \ (\text{Support}=30\%, \text{Confidence}=60\%)$$

does not hold when the minimum support is set to 40%, but the following association rule may be valid

$$\text{PC} \Rightarrow \text{Printer}.$$

Besides, note that in reality the frequencies of items are not uniform. Some items occur very frequently in the transactions while others rarely appear. In this case, 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.

To meet such situations, we have investigated the problem of mining generalized association rules across different levels of taxonomy with non-uniform minimum supports [16]. We proposed two efficient algorithms, MMS_Cumulate and MMS_Stratify, which not only can discover associations that span different hierarchy levels but also have high potential to produce rare but informative item rules.

The proposed approaches, however, are not effective to the situation for frequently updating the large source database. In this case, adopting the mining approach tends to re-applying the whole process on the updated database to reflect correctly the most recent associations among items. This is not cost-effective and is unacceptable in general. To be more realistic and cost-effective, it is better to perform the association mining algorithms to generate the initial association rules, and then upon updating the source database, apply an incremental maintenance method to re-build the discovered rules. The major challenge here is to deploy an efficient maintenance algorithm to fa-

cilitate the whole mining process. This problem is nontrivial because updates may invalidate some of the discovered association rules, turn previous weak rules into strong ones and import new, undiscovered rules.

In this paper, we addressed the issues for developing efficient maintenance methods and proposed two algorithms, UD_Cumulate and UD_Stratify. Our algorithms can incrementally update the generalized association rules with non-uniform support specification and is capable of effectively reducing the number of candidate sets and database re-scanning. The performance study showed that UD_Cumulate and UD_Stratify are 2-6 times faster than running MMS_Cumulate or MMS_Stratify on the updated database afresh.

The remainder of the paper is organized as follows. The problem of maintaining generalized association rules with multiple minimum supports is formalized in Section 2. In Section 3, we explain the proposed algorithms for updating frequent itemsets with multiple minimum supports. The evaluation of the proposed algorithms on IBM synthetic data 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

2.1 Mining multi-supported, generalized association rules

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items and $DB = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where each transaction $t_i = \langle \text{tid}, A \rangle$ has a unique identifier *tid* and a set of items A ($A \subseteq I$). We assume that the taxonomy of items, T, is available and is denoted as a directed acyclic graph on $I \cup J$, where $J = \{j_1, j_2, \ldots, j_p\}$ 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. For example, in Figure 1 I = {Laser printer, Ink-jet printer, Dot matrix printer, Desktop PC, Notebook, Scanner} and J = {Non-impact printer, Printer, Personal computer}.

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 item in A. An itemset B has support s, denoted as $s = sup(B)$, in the transaction set DB if s% of transactions in DB contain B.

Definition 2. Given a set of transactions DB and a taxonomy T, a generalized association rule is an implication of the form $A \Rightarrow B$, where $A, B \subset I \cup J$, $A \cap B = \varnothing$, and no item in B is an ancestor of any item in A. The support of this rule, $sup(A \Rightarrow B)$, is equal to the support of $A \cup B$. The confidence of the rule, $conf(A \Rightarrow B)$, is the ratio of $sup(A \cup B)$ and $sup(A)$.

The condition in Definition 2 that no item in A is an ancestor of any item in B is essential; otherwise, a rule of the form, $a \Rightarrow ancestor(a)$, always has 100% confidence and is trivial.

Definition 3. Let $ms(a)$ denote the minimum support of an item a in $I \cup J$. An itemset $A = \{a_1, a_2, \ldots, a_k\}$, where $a_i \in I \cup J$, 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 4. A multi-supported, generalized association rule $A \Rightarrow B$ is *strong* if

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

and

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

Definition 5. Given a set of transactions DB, a taxonomy T, the user-specified minimum supports for all items in T, $\{ms(a_1), ms(a_2), \ldots, ms(a_n)\}$, and the *minconf*, the problem of mining multi-supported, generalized association rules is to find all association rules that are strong.

Example 1. Suppose that a shopping transaction database DB in Table 1 consists of items I = {Laser printer, Ink-jet printer, Dot matrix printer, Desktop PC, Notebook, Scanner} and taxonomy T as shown in Figure 1. Let the minimum confidence (*minconf*) be 60% and the minimum support (*ms*) associated with each item in the taxonomy be as follows:

ms(Printer) = 80%	*ms*(Non-impact) = 65%	*ms*(Dot matrix) = 70%
ms(Laser) = 25%	*ms*(Ink-jet) = 60%	*ms*(Scanner) = 15%
ms(PC) = 35%	*ms*(Desktop) = 25%	*ms*(Notebook) = 25%

The following generalized association rule,

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

fails because its support is less than min{*ms*(PC), *ms*(Laser), *ms*(Dot matrix)} = 25%. But another rule,

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

holds because both its support and confidence are larger than min{*ms*(PC), *ms*(Laser)} = 25% and *minconf*, respectively. Table 2 lists the frequent itemsets and the resulting strong rules for this example.

Table 1. A transaction database (*DB*).

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

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

Itemsets	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 ⇒ Laser (*sup* = 33.3%, *conf* = 66.7%)
Laser ⇒ PC (*sup* = 33.3%, *conf* = 100%)
Notebook ⇒ Printer (*sup* = 33.3%, *conf* = 100%)
Notebook ⇒ Non-impact (*sup* = 33.3%, *conf* = 100%)
Notebook ⇒ Laser (*sup* = 33.3%, *conf* = 100%)

As shown in [1], the task of mining association rules is usually decomposed into two steps:

1. Itemset generation: find all frequent itemsets those have support exceeding a threshold minimum support.
2. Rule construction: from the set of frequent itemsets, construct all association rules that have a confidence exceeding a threshold minimum confidence.

Since the solution to the second subproblem is straightforward, the problem can be reduced to finding the set of frequent itemsets that satisfy the specified minimum support.

2.2 Maintaining multi-supported, generalized association rules

In real business applications, the database grows over time. This implies that if the updated database is processed afresh, the previously discovered associations might be invalid and some undiscovered associations should be generated. That is, the discovered association rules must be updated to reflect the new circumstance. Analogous to mining associations, this problem can be reduced to updating the frequent itemsets.

Definition 6. Let *DB* denote the original database, *db* the incremental database, *UD* the updated database containing *db* and *DB*, i.e., *UD* = *db* + *DB*, *T* the taxonomy of items, and L^{DB} the set of frequent itemsets in *DB*. The problem of updating the frequent itemsets with taxonomy and multiple supports is to find

$$L^{UD} = \{A | \, sup_{UD}(A) \geq \min_{a_i \in A} ms(a_i)\},$$

given the knowledge of *DB*, *T*, *db*, L^{DB}, and $sup_{DB}(A) \forall A \in L^{DB}$.

Example 2. Consider Example 1 again, and suppose that the incremental transaction database *db* is shown in Table 3. Table 4 lists the frequent itemsets and the resulting strong rules. Comparing Table 4 to Table 2, we observe that two old frequent 2-itemsets in Table 2, {Scanner, Printer} and {Scanner, Dot matrix}, are discarded, while three new frequent itemsets, {Desktop}, {PC, Printer}, and {PC, Non-impact}, are added into Table 4, and two new rules PC $\Rightarrow$ Printer and PC $\Rightarrow$ Non-impact are found.

Table 3. An incremental database (*db*).

TID	Items Purchased
17	Desktop computer, Laser printer
18	Laser printer

3. Methods for updating frequent itemsets for multi-supported, generalized association rules

3.1 Preliminary

As stated in the pioneering work [4], the primary challenge of devising effective association rules maintenance algorithm is how to reuse the original frequent itemsets and avoid the possibility of re-scanning the original database *DB*.

Table 4. Frequent itemsets and association rules generated for Example 2.

Itemsets	min *ms* (%)	Support (%)
{Scanner}	15	25.0
{Desktop}	25	25.0
{Notebook}	25	25.0
{PC}	35	50.0
{Laser}	25	50.0
{PC, Printer}	35	37.5
{PC, Non-impact}	35	37.5

{Laser, PC}	25	37.5
{Notebook, Printer}	25	25.0
{Notebook, Non-impact}	25	25.0
{Notebook, Laser}	25	25.0
Rules		
PC ⇒ Printer (*sup* = 37.5%, *conf* = 75%)		
PC ⇒ Non-impact (*sup* = 37.5%, *conf* = 75%)		
PC ⇒ Laser (*sup* = 37.5%, *conf* = 75%)		
Laser ⇒ PC (*sup* = 37.5%, *conf* = 75%)		
Notebook ⇒ Printer (*sup* = 25%, *conf* = 100%)		
Notebook ⇒ Non-impact (*sup* = 25%, *conf* = 100%)		
Notebook ⇒ Laser (*sup* = 25%, *conf* = 100%)		

Let $|DB|$ denote the number of transaction records in the original database *DB*, $|db|$ be the number of transaction records in the incremental database *db*, and $|UD|$ be the number of transaction records in the whole updated database *UD* containing *db* and *DB*. For a sorted *k*-itemset $A = \langle a_1, a_2, \ldots, a_k \rangle$ with $ms(a_1) \le ms(a_2) \le \ldots \le ms(a_k)$, we define its support counts in *db* as $A.count_{db}$, in *DB* as $A.count_{DB}$ and in *UD* as $A.count_{UD}$. Note that $|UD| = |db| + |DB|$ and $A.count_{UD} = A.count_{db} + A.count_{DB}$. After scanning the incremental database *db*, we have $A.count_{db}(A)$ and can proceed further according to the following conditions.

(1) If *A* is a frequent itemset both in *db* and *DB*, i.e., $count_{db}(A) \ge ms(a_1) \times |db|$ and $count_{DB}(A) \ge ms(a_1) \times |DB|$, then *A* is a frequent itemset in *UD*. There is no need to compute because $count_{UD}(A) \ge ms(a_1) \times |UD|$.

(2) If *A* is not a frequent itemset in *db* but is frequent in *DB*, then a simple calculation can determine whether *A* is frequent or not in *UD*.

(3) If *A* is a frequent itemset in *db* but not frequent in *DB*, then *A* is an undetermined itemset in *UD*. Since $count_{DB}(A)$ is not available, we must re-scan *DB* to compute $count_{UD}(A)$ to decide whether *A* is frequent or not in *UD*.

(4) If A is neither a frequent itemset in db nor in DB, then A is not frequent in UD. There is no need for further computation.

These four cases are depicted in Figure 2. Note that only case 3 yields the essence of re-scanning the original database DB.

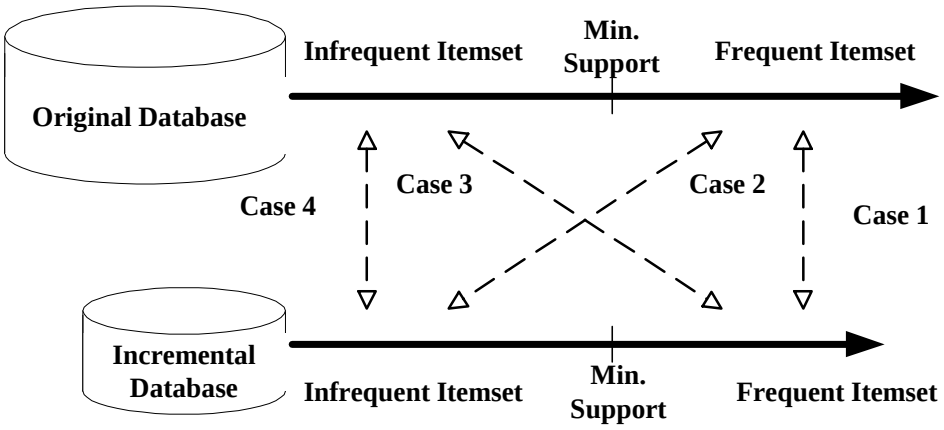


Figure 2. Four cases in the incremental frequent itemset maintenance [8].

Let k-itemset denote an itemset with k items. The basic process of updating generalized association rules with multiple minimum supports is similar to previous work [4, 6] on updating association rules with uniform support and follows the level-wise approach widely used in most efficient algorithms to generate all frequent k-itemsets. First, count all 1-itemsets in db and then determine whether to re-scan the original database DB from these itemsets and finally create frequent 1-itemsets L_1. From frequent 1-itemsets, generate candidate 2-itemsets C_2 and repeat the above procedure until no frequent k-itemsets L_k are created.

The above paradigm, however, has to be modified to incorporate taxonomy information and multiple minimum supports. First, note that in the presence of taxonomy an itemset can be composed of items, primitive or generalized, in the taxonomy. To calculate the occurrence of each itemset, the current scanned transaction t is extended to include the generalized items of all its component items.

Secondly, note that the well-known apriori pruning technique based on the concept of *downward closure* does not work for multiple support specification. 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% of 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, we have adopted the *sorted closure property* [9] in our previous work for mining generalized association rules with multiple minimum supports. Hereafter, to distinguish from the traditional itemset, a sorted k-itemset denoted as $\langle a_1, a_2, \ldots, a_k\rangle$ is used.

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

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 1 reveals the obstacle in using the apriori-gen in generating frequent itemsets.

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

For example, consider a sorted candidate 2-itemset $\langle a, b\rangle$. It is easy to find if we want to generate this itemset from L_1, both items a and b should be included in L_1; that is, each one should be occurring more frequently than the corresponding minimum support $ms(a)$ and $ms(b)$. Clearly, the case $ms(a) \leq sup(b) < ms(b)$ fails to generate $\langle a, b\rangle$ in C_2 even $sup(\langle a, b\rangle) \geq ms(a)$.

For the above reason, all items within an itemset are sorted in the increasing order of their minimum supports, and a sorted itemset, called frontier set, $F = \langle a_j, a_{j_1}, a_{j_2}, \ldots, a_{j_l}\rangle$, is facilitated to generate the set of candidate 2-itemsets, where $a_j = \min_{a_i \in I \cup J} \{a_i | sup(a_i) \geq ms(a_i)\}$, $ms(a_j) \leq ms(a_{j_1}) \leq ms(a_{j_2}) \leq \ldots \leq ms(a_{j_l})$, $sup(a_{j_i}) \geq ms(a_j)$, $1 \leq i \leq l$.

Example 3. Continuing with Example 1, we change *ms*(Scanner) from 15% to 20%. The resulting *F* is shown in Table 5. From Table 5, we observe that *ms*(Scanner) is the smallest of all items, and Scanner could join with any item whose support is greater than or equal to *ms*(Scanner) = 20% to become a C_2 candidate without losing any 2-itemsets. The 2-itemsets ⟨Scanner, Desktop⟩ and ⟨Scanner, Ink-jet⟩ could not become candidates because *sup*(Desktop) or *sup*(Ink-jet) is less than *ms*(Scanner), and the supports of ⟨Scanner, Desktop⟩ and ⟨Scanner, Ink-jet⟩ could not be greater than *sup*(Desktop) and *sup*(Ink-jet), respectively, according to the *downward closure property*. Therefore, we keep items whose support is greater than or equal to *ms*(Scanner) in *F*, and discard Desktop and Ink-jet.

For C_k, $k \geq 3$, the candidate generation is not changed. Please refer to [16] for details on the mining of multi-support, generalized association rules.

3.2 Algorithm UD_Cumulate

The basic process of UD_Cumulate algorithm is proceeded as follows. First, count all 1-itemsets in *db* including generalized items. Second, combine the itemset counts in *db* and *DB*, and create the frequent 1-itemsets L_1 according to the four cases. Next, create the frontier set F^{UD} and use it to generate candidate 2-itemsets C_2. Then, generate the frequent 2-itemsets L_2 following the same procedure for L_1. Finally, for $k \geq 3$, repeat the above procedure until no frequent *k*-itemsets L_k are created, except that the candidate *k*-itemsets C_k are generated from L_{k-1}. In each iteration *k* for C_k generation, an additional pruning technique as described below is performed.

Table 5. Generating frontier set *F*.

Item	Sorted *ms* %	Support %	*F*
Scanner	20	33.3	Scanner

Laser	25	33.3	Laser
Desktop	25	16.7	Notebook
Notebook	25	33.3	PC
PC	35	50.0	Non-impact
Ink-jet	60	16.7	Dot-matrix
Non-impact	65	50.0	Printer
Dot-matrix	70	50.0	
Printer	80	66.7	

Lemma 3. For any sorted k-itemset $A = \langle a_1, a_2, \ldots, a_k \rangle$, if $k \geq 2$, A can be pruned if A is not a frequent itemset in DB and there exists a subset of A, say $\langle a_{i_1}, a_{i_2}, \ldots, a_{i_{k-1}} \rangle$, such that $sup_{db}(\langle a_{i_1}, a_{i_2}, \ldots, a_{i_{k-1}} \rangle) < ms(a_1)$ in db, and $a_{i_1} = a_1$ or $ms(a_1) = ms(a_2)$.

Proof. Note that if A is not frequent in DB and db, then A will not be frequent in UD. Since we know that A is not frequent in DB, all we have to do is to make sure A is not frequent in db. If we can find A is not frequent in db, then we sure A is not frequent in UD. ■

Example 4. Assume that ms(Scanner) = 30%, ms(Notebook) = 30%, ms(Ink-jet) = 50%, $|DB| = 1000$, $|db| = 100$, ⟨Scanner, Notebook, Ink-jet⟩ is not frequent in DB. Let us consider the three subsets of ⟨Scanner, Notebook, Ink-jet⟩, and assume that sup_{db}(⟨Scanner, Notebook⟩), sup_{db}(⟨Scanner, Ink-jet⟩) or sup_{db}(⟨Notebook, Ink-jet⟩) is not frequent in db. Since ⟨Scanner, Notebook, Ink-jet⟩ is infrequent in DB, $count_{DB}$(⟨Scanner, Notebook, Ink-jet⟩) < 30% × 1000 = 300. Now, if ⟨Scanner, Notebook⟩ or ⟨Scanner, Ink-jet⟩ is not frequent in db, then sup_{db}(⟨Scanner, Notebook, Ink-jet⟩) ≤ sup_{db}(⟨Scanner, Notebook⟩) < ms(Scanner), or sup_{db}(⟨Scanner, Notebook, Ink-jet⟩) ≤ sup_{db}(⟨Scanner, Ink-jet⟩) < ms(Scanner) in db according to the *downward closure property*, and so $count_{db}$(⟨Scanner, Notebook, Ink-jet⟩) < 100

× 30% = 30. Hence, $count_{UD}$(⟨Scanner, Notebook, Ink-jet⟩) = $count_{db}$(⟨Scanner, Notebook, Ink-jet⟩) + $count_{DB}$(⟨Scanner, Notebook, Ink-jet⟩) < 30 + 300 = 330. It follows that sup_{UD}(⟨Scanner, Notebook, Ink-jet⟩) = $count_{UD}$(⟨Scanner, Notebook, Ink-jet⟩)/|*UD*| < 330/(1000+100) = 30% = *ms*(Scanner). Therefore, ⟨Scanner, Notebook, Ink-jet⟩ is not a frequent 3-itemset in *UD*. On the other hand, if ⟨Notebook, Ink-jet⟩ is not frequent in *db*, then it is true that sup_{db}(⟨Scanner, Notebook, Ink-jet⟩) ≤ sup_{db}(⟨Notebook, Ink-jet⟩) < *ms*(Notebook) = 30% = *ms*(Scanner). Hence, ⟨Scanner, Notebook, Ink-jet⟩ is not frequent in *db*. Finally, we can derive that ⟨Scanner, Notebook, Ink-jet⟩ is not a frequent 3-itemset in *UD*.

The procedure for generating F^{UD} is described in Figure 3.

Example 5. Let us continue with Example 2. Tables 6 and 7 show the progressing results for generating F^{UD}. First, scan *db* to find the counts of all 1-itemsets in *db*, then scan *DB* to find the counts of all 1-itemsets that are not in L_1^{DB}. The result is shown in Table 6. Next, calculate and combine the counts of items calculated in the first two steps and finally generate F^{UD}, and the result is shown in Table 7.

---

1. **for** each transaction $t \in db$ **do** /* Scan *db* and calculate $count_{db}(a)$ */
2. **for** each item $a \in t$ **do**
3. Add all ancestors of *a* in *IA* into *t*;
4. Remove any duplicates from *t*;
5. **for** each item $a \in t$ **do**
6. $count_{db}(a)$++;
7. **end for**
8. **for** each item $a \in L_1^{DB}$ **do** /* Cases 1 & 2 */
9. $count_{UD}(a) = count_{DB}(a) + count_{db}(a)$;
10. **if** $sup_{UD}(a) \geq ms(a)$ **then**
11. $L_1^{UD} = L_1^{UD} \cup \{a\}$;
12. **end for**
13. **for** each transaction $t \in DB$ **do** /* Scan *DB* and calculate $count_{DB}(a)$
14. for $a \notin L_1^{DB}$ */
15. **for** each item $a \in t$ **do**

16. Add all ancestors of a in IA into t;
17. Remove any duplicates from t;
18. **for** each item $a \in t$ and $a \notin L_1^{DB}$ **do**
19. $count_{DB}(a)$++;
20. **end for**
21. **for** each item $a \notin L_1^{DB}$ **do** /* Cases 3 & 4 */
22. $count_{UD}(a) = count_{DB}(a) + count_{db}(a)$;
23. **if** $sup_{UD}(a) \geq ms(a)$ **then**
24. $L_1^{UD} = L_1^{UD} \cup \{a\}$;
25. **end for**
26. **for** each item a in SMS in the same order **do**
27. **if** $a \in L_1^{UD}$ **then**
28. Insert a into F^{UD};
29. **break**;
30. **end if**
31. **for** each item b in SMS that is after a in the same order **do**
32. **if** $sup_{UD}(b) \geq ms(a)$ **then**
33. Insert b into F^{UD};

Figure 3. Procedure F^{UD}-gen(SMS, DB, db, L_1^{DB}, IA).

Table 6. The result for scanning *db* and *DB*.

L_1^{DB}			Scan *db*			Scan *DB* not in L_1^{DB}		
1-itemset	Count	Sup %	1-itemset	Count	Sup %	1-itemset	Count	Sup %
Scanner	2	33.3	Laser	2	100	Desktop	1	16.7
Laser	2	33.3	Desktop	1	50.0	Dot-matrix	3	50.0
Notebook	2	33.3	PC	1	50.0	Non-impact	3	50.0
PC	3	50.0	Non-impact	2	100	Ink-jet	1	16.7
			Printer	2	100	Printer	4	66.7

Table 7. Generating frontier set F^{UD}.

C_1	Count	Sorted *ms*%	Sup %	F^{UD}
Scanner	2	15	25.0	Scanner
Laser	4	25	50.0	Laser
Desktop	2	25	25.0	Desktop
Notebook	2	25	25.0	Notebook
PC	4	35	50.0	PC
Ink-jet	1	60	12.5	Non-impact
Non-impact	5	65	62.5	Dot-matrix
Dot-matrix	3	70	37.5	Printer
Printer	6	80	75.0	

Figure 4 shows an overview of the UD_Cumulate algorithm. As an illustration, let us consider the example shown in Figure 5. For convenience, all the itemsets and frequent itemsets in *DB*, *db* and *UD* are first shown in Tables 8 to 13. A simple picture of running the UD_Cumulate algorithm on the example in Figure 5 is shown in Figure 6. First, scan *db* and sort all the items in *db*. Second, load L_1^{DB}, and check whether those items in *db* are in L_1^{DB}, if so, we can calculate their supports straightforward. Otherwise, we must re-scan the original database *DB* to determine whether candidate 1-itemsets are frequent or not. At the same time, we generate the frontier set F^{UD} and use it to generate candidate 2-itemsets C_2. After that, the process is almost the same as that for generating L_1, except that, if candidate 2-itemsets are not frequent in *DB*, we can use $count_{db}(C_1)$ to prune C_2, and determine whether candidate 2-itemsets C_2 are frequent or not in *db*.

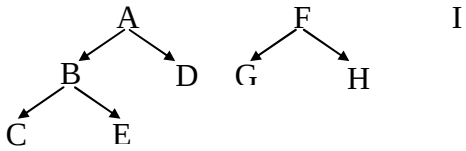
---

1. Create *IMS*; /* The table of user-defined minimum support */
2. Create *IA*; / * The table of each item and its ancestors from taxonomy *T* */
3. *SMS* = sort(*IMS*); /* Ascending sort according to *ms*(*a*) stored in *IMS* */
4. $F^{UD} = F^{UD}$-gen(*SMS*, *DB*, *db*, L_1^{DB}, *IA*);
5. $L_1 = \{a \in F^{UD} \mid sup(a) \geq ms(a)\}$;
6. **for** ($k = 2$; $L_{k-1} \neq \varnothing$; k++) **do**
7. **if** $k = 2$ **then** $C_2 = C_2$-gen(F^{UD});
8. **else** C_k = apriori-gen(L_{k-1});
9. Delete any candidate in C_k that consists of an item and its ancestor;
10. Delete any candidate in C_k that satisfies Lemma 3;
11. Delete any ancestor in *IA* that is not present in any of the candidates in
12. C_k;
13. Delete any item in F^{UD} that is not present in any of the candidates in C_k;
14. Cal-count(*db*, F^{UD}, C_k); /* Scan *db* and calculate counts of C_k */
15. **for** each candidate $A \in L_k^{DB}$ **do** /* Cases 1 & 2 */
16. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
17. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then** /* A[1] denotes the first item with the smallest minimum support in sorted A */
18. $L_k^{UD} = L_k^{UD} \cup \{A\}$;
19. **end for**
20. **for** each candidate $A \in C_k - L_k^{DB}$ **do** /* Cases 3 & 4 */

21. **if** $count_{db}(A) < ms(A[1]) \times |db|$ **then**
22. Delete any candidate in $C_k - L_k^{DB}$;
23. Cal-count(DB, F^{UD}, $C_k - L_k^{DB}$); /* Scan DB and calculate counts of $C_k - L_k^{DB}$ */
24. **for** each candidate $A \in C_k - L_k^{DB}$ **do**
25. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
26. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then**
27. $L_k^{UD} = L_k^{UD} \cup \{A\}$;
28. **end for**
29. Result = $\bigcup_k L_k^{UD}$;

Figure 4. Algorithm UD_Cumulate.

Taxonomy



Original Database (*DB*)

TID	Items Purchased
11	H, C
12	I, D
13	D, E
14	H, D, C
15	I
16	G

Hierarchy Table (*HI*)

Item	Level_No.	Group	SubLevel
A	3	1	1
B	3	1	2
C	3	1	3
D	3	1	2
E	3	1	3
F	2	2	1
G	2	2	2
H	2	2	2
I	1	3	1

Incremental Database (*db*)

TID	Items Purchased
17	G, C
18	C

Minimum Item Support Table (*MIS*)

Item	*minsup* %
A	80

Item Ancestor Table (*IA*)

Item	Ancestor_1	Ancestor_2
A		

B	65
C	25
D	70
E	60
F	35
G	25
H	25
I	15

B	A	
C	A	B
D	A	
E	A	B
F		
G	F	
H	F	
I		

Figure 5. An example of updating generalized association rules.

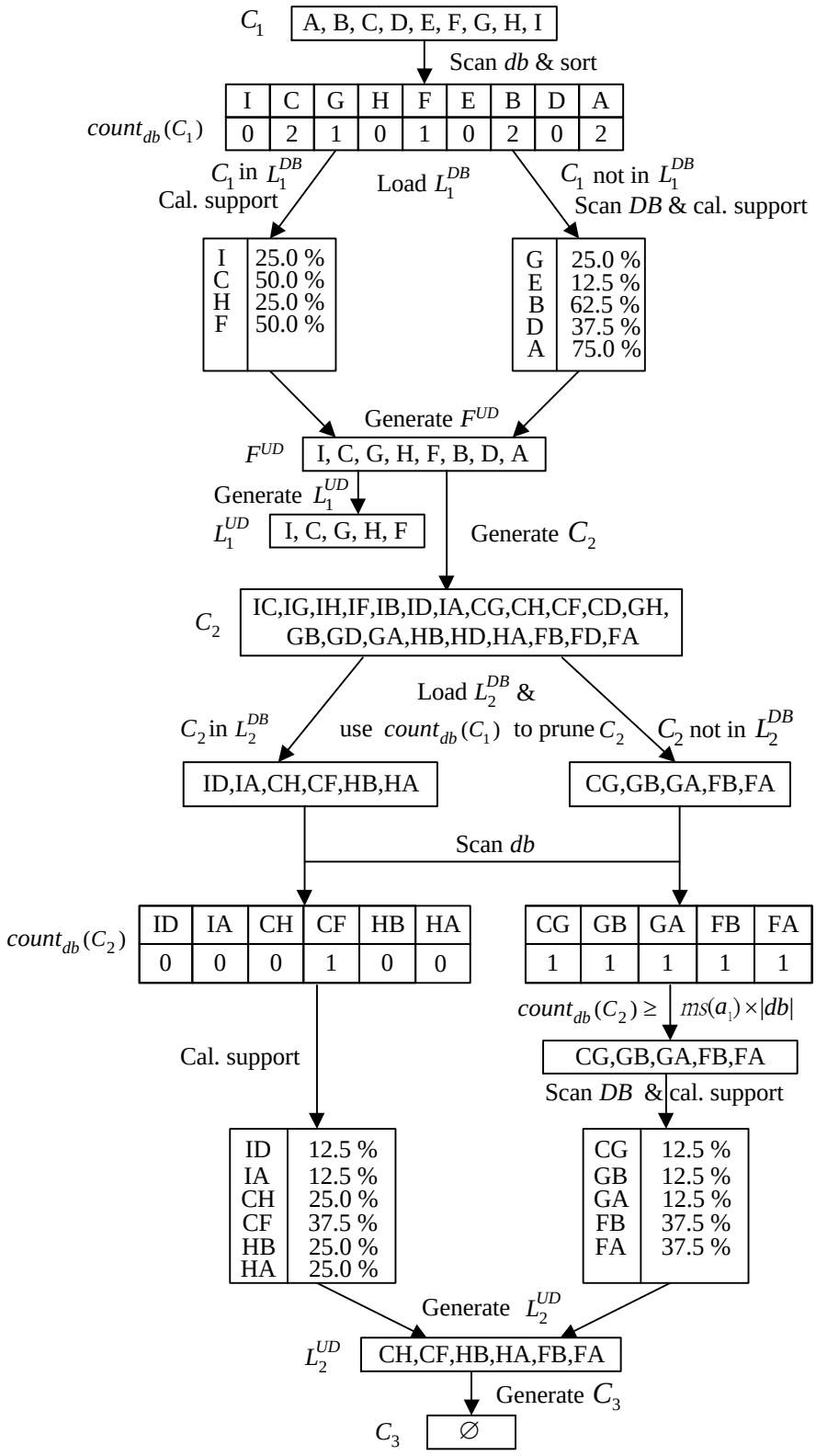


Figure 6. An example of UD_Cumulate.

Table 8. Summary for candidates, counts and supports of *DB* in Figure 5.

C_1	Count	Sup %	C_2	Count	Sup %	C_3	Count	Sup %
I	2	33.3	I, D	1	16.7	C, H, D	1	16.7
C	2	33.3	I, A	1	16.7	C, F, D	1	16.7
G	1	16.7	C, H	2	33.3	H, B, D	1	16.7
H	2	33.3	C, F	2	33.3	F, B, D	1	16.7
F	3	50.0	C, D	1	16.7			
E	1	16.7	H, B	2	33.3			
B	3	50.0	H, D	1	16.7			
D	3	50.0	H, A	2	33.3			
A	4	66.7	F, B	2	33.3			
			F, D	1	16.7			
			F, A	2	33.3			
			E, D	1	16.7			
			B, D	2	33.3			

Table 9. Summary for candidates, counts and supports of *db* in Figure 5.

C_1	Count	Sup %	C_2	Count	Sup %
C	2	100	C, G	1	50.0
G	1	50.0	C, F	1	50.0
F	1	50.0	G, B	1	50.0
B	2	100	G, A	1	50.0
A	2	100	F, B	1	50.0
			F, A	1	50.0

Table 10. Frontier set F^{UD} in Figure 5.

C_1	Count	Sorted *ms*%	Sup %	F^{UD}
I	2	15	25.0	I
C	4	25	50.0	C
G	2	25	25.0	G
H	2	25	25.0	H
F	4	35	50.0	F
E	1	60	12.5	B
B	5	65	62.5	D
D	3	70	37.5	A
A	6	80	75.0	

Table 11. Summary for candidates, counts and supports of *UD* in Figure 5.

C_1	Count	Sup %	C_2	Count	Sup %	C_3	Count	Sup %
I	2	25.0	I, D	1	12.5	C, H, D	1	12.5
C	4	50.0	I, A	1	12.5	C, F, D	1	12.5
G	2	25.0	C, G	1	12.5	H, B, D	1	12.5
H	2	25.0	C, H	2	25.0	F, B, D	1	12.5
F	4	50.0	C, F	3	37.5			
E	1	12.5	C, D	1	12.5			
B	5	62.5	G, B	1	12.5			
D	3	37.5	G, A	1	12.5			
A	6	75.0	H, B	2	25.0			
			H, D	1	12.5			
			H, A	2	25.0			
			F, B	3	37.5			
			F, D	1	12.5			
			F, A	3	37.5			
			E, D	1	12.5			
			B, D	2	25.0			

Table 12. Frequent itemsets summary of *DB* in Figure 5.

L_1^{DB}			L_2^{DB}			L_3^{DB}
1-itemset	Count	Sup %	2-itemset	Count	Sup %	3-itemset
I	2	33.3	I, D	1	16.7	∅
C	2	33.3	I, A	1	16.7	
H	2	33.3	C, H	2	33.3	
F	3	50.0	C, F	2	33.3	
			H, B	2	33.3	
			H, A	2	33.3	

Table 13. Frequent itemsets summary of *UD* in Figure 5.

L_1^{UD}			L_2^{UD}			L_3^{UD}
1-itemset	Count	Sup %	2-itemset	Count	Sup %	3-itemset
I	2	25.0	C, H	2	25.0	∅
C	4	50.0	C, F	3	37.5	
G	2	25.0	H, B	2	25.0	
H	2	25.0	H, A	2	25.0	
F	4	50.0	F, B	3	37.5	
			F, A	3	37.5	

3.3 Algorithm UD_Stratify

The UD_Stratify algorithm is based on the concept of stratification introduced in [14], which refers to a level-wise counting strategy from the top level of the taxonomy down to the lowest level, hoping that those candidates containing items at higher levels will not have minimum support, yielding no need to count candidates which include items at lower levels. However, this counting strategy may fail in the case of non-uniform minimum supports.

Example 6. Let {Printer, PC}, {Printer, Desktop}, and {Printer, Notebook} are the candidate itemsets to be counted. The taxonomy and minimum supports are defined as Example 1. Using the level-wise strategy, we first count {Printer, PC} and assume that it is not frequent, i.e., $sup(\{\text{Printer, PC}\}) < 0.35$. Since the minimum supports of {Printer, Desktop}, 0.25, and {Printer, Notebook}, also 0.25, are less than {Printer, PC}, we cannot assure that the occurrences of {Printer, Desktop} and {Printer, Notebook}, though less than {Printer, PC}, are also less than their minimum supports. In this case, we still have to count {Printer, Desktop} and {Printer, Notebook} even though {Printer, PC} does not have minimum support.

The following observation inspires us to the deployment of the UD_Stratify algorithm.

Lemma 4. Consider the two k-itemsets $\langle a_1, a_2, \ldots, a_k\rangle$ and $\langle a_1, a_2, \ldots, \hat{a}_k \rangle$, where $\hat{a}_k$ is an ancestor of a_k. If $\langle a_1, a_2, \ldots, \hat{a}_k \rangle$ is not frequent, then neither is $\langle a_1, a_2, \ldots, a_k\rangle$.

Lemma 4 implies that if a sorted candidate itemset in 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. We thus first divide C_k, according to the ancestor-descendant re-

lationship claimed in Lemma 4, into two disjoint subsets, called *top candidate set* TC_k and *residual candidate set* RC_k, as defined below.

Definition 7. Consider a set, S_k, of candidates in C_k induced by the schema $\langle a_1, a_2, \ldots, a_{k-1}, * \rangle$, where '*' denotes don't care. A candidate k-itemset $A = \langle a_1, a_2, \ldots, a_{k-1}, a_k \rangle$ is a top candidate if none of the candidates in S_k is an ancestor of A. That is,

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

and

$RC_k = C_k - TC_k$, if every itemset in TC_k is a frequent itemset.

Example 7. Assume that the candidate 2-itemset C_2 in Example 1 consists of ⟨Scanner, PC⟩, ⟨Scanner, Desktop⟩, ⟨Scanner, Notebook⟩, ⟨Notebook, Laser⟩, ⟨Notebook, Non-impact⟩, and ⟨Notebook, Dot matrix⟩, and the supports of the items in higher levels are larger than those in lower levels. Then TC_2 = {⟨Scanner, PC⟩, ⟨Notebook, Non-impact⟩, ⟨Notebook, Dot matrix⟩} and RC_2 = {⟨Scanner, Desktop⟩, ⟨Scanner, Notebook⟩, ⟨Notebook, Laser⟩}. If the support of the top level candidate does not pass its minimum support *ms*(Scanner), we do not need to count the remaining descendant candidates ⟨Scanner, Desktop⟩, ⟨Scanner, Notebook⟩. On the contrary, if ⟨Scanner, Desktop⟩ is a frequent itemset, we should perform another pass over *DB* to count ⟨Scanner, Desktop⟩ and ⟨Scanner, Notebook⟩ to determine whether they are frequent or not.

Our approach is that, for $k \ge 2$, rather than counting all candidates in C_k as in UD_Cumulate algorithm, we count the supports of candidates in TC_k, deleting as well as their descendants in RC_k the candidates that do not have minimum support. If RC_k is not empty, we then perform an extra scanning over the transaction database *DB* 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 UD_Stratify algorithm is described in Figure 7. The procedures for generating TC_k and RC_k are given in Figure 8 and Figure 9, respectively.

Figure 10 shows a picture of running the UD_Stratify algorithm on the example in Figure 5. The procedure for generating frontier set F^{UD} and L_1^{UD} is the same as that in the UD_Cumulate algorithm. After that, first, we sort C_2 to find the *top candidate set* TC_2, like UD_Cumulate algorithm doing, and generate TL_2. We then use TL_2 to prune C_2 and generate RC_2. Since RC_2 is not empty, we must do the same procedure as generating TL_2, and generate RL_2 and L_k^{UD} lastly.

1. Create *IMS*; /* The table of user-defined minimum support */
2. Create *HI*; /* The table of each item with Hierarchy Level, Sublevel, and Group */
3. Create *IA*; /* The table of each item and its ancestors from taxonomy *T* */
4. *SMS* = sort(*IMS*); /* Ascending sort according to *ms*(*a*) stored in *IMS* */
5. $F^{UD} = F^{UD}$-gen(*SMS*, *DB*, *db*, L_1^{DB}, *IA*);
6. $L_1 = \{a \in F^{UD} \mid sup(a) \geq ms(a)\}$;
7. **for** ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do**
8. **if** $k = 2$ **then** $C_2 = C_2$-gen(F)
9. **else** $C_k = C_k$-gen(L_{k-1});
10. Delete any candidate in C_k that consists of an item and its ancestor;
11. Delete any candidate in C_k that satisfies Lemma 3;
12. $TC_k = TC_k$-gen(C_k, *HI*); /* Using C_k, *HI* to find *top* C_k */
13. Delete any ancestor in *IA* that is not present in any of the candidates in TC_k;
14. Delete any item in F^{UD} that is not present in any of the candidates in C_k;
15. Cal-count(*db*, F^{UD}, TC_k); /* Scan *db* and calculate counts of TC_k */
16. **for** each candidate $A \in L_k^{DB}$ **do** /* Cases 1 & 2 */
17. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
18. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then** /* A[1] denotes the first item with the smallest minimum support in sorted A */
19. $TL_k = TL_k \cup \{A\}$;
20. **end for**
21. **for** each candidate $A \in TC_k - L_k^{DB}$ **do** /* Cases 3 & 4 */
22. **if** $count_{db}(A) < ms(A[1]) \times |db|$ **then**
23. Delete any candidate in $TC_k - L_k^{DB}$;
24. Cal-count(*DB*, F^{UD}, $TC_k - L_k^{DB}$); /* Scan *DB* and calculate counts of $TC_k - L_k^{DB}$ */
25. **for** each candidate $A \in TC_k - L_k^{DB}$ **do**
26. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
27. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then**
28. $TL_k = TL_k \cup \{A\}$;
29. **end for**
30. $RC_k = RC_k$-gen(C_k, TC_k, TL_k); /* Use C_k, TL_k to find residual C_k */
31. **If** $RC_k \neq \emptyset$ **then**
32. Delete any ancestor in *IA* that is not present in any of the candidates in RC_k;
33. Delete any item in F^{UD} that is not present in any of the candidates in RC_k;
34. Cal-count(*db*, F^{UD}, RC_k); /* Scan *db* and calculate counts of RC_k */
35. **for** each candidate $A \in L_k^{DB}$ **do** /* Cases 1 & 2 */
36. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
37. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then**
38. $RL_k = RL_k \cup \{A\}$;
39. **end for**

40. **for** each candidate $A \in RC_k - L_k^{DB}$ **do** /* Cases 3 & 4 */
41. **if** $count_{db}(A) < ms(A[1]) \times |db|$ **then**
42. Delete any candidate in $RC_k - L_k^{DB}$;
43. Cal-count(DB, F^{UD}, $RC_k - L_k^{DB}$); /* Scan DB and calculate counts of
44. $RC_k - L_k^{DB}$ */
45. **for** each candidate $A \in RC_k - L_k^{DB}$ **do** /* Cases 3 & 4 */
46. $count_{UD}(A) = count_{DB}(A) + count_{db}(A)$;
47. **if** $count_{UD}(A) \geq ms(A[1]) \times |UD|$ **then**
48. $RL_k = RL_k \cup \{A\}$;
49. **end for**
50. **end if**
51. $L_k^{UD} = TL_k \cup RL_k$;
52. **end for**
53. Result = $\bigcup_k L_k^{UD}$;

Figure 7. Algorithm UD_Stratify.

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

Figure 8. Procedure TC_k -gen(C_k, HI).

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

Figure 9. Procedure RC_k-gen(C_k , TC_k, TL_k).

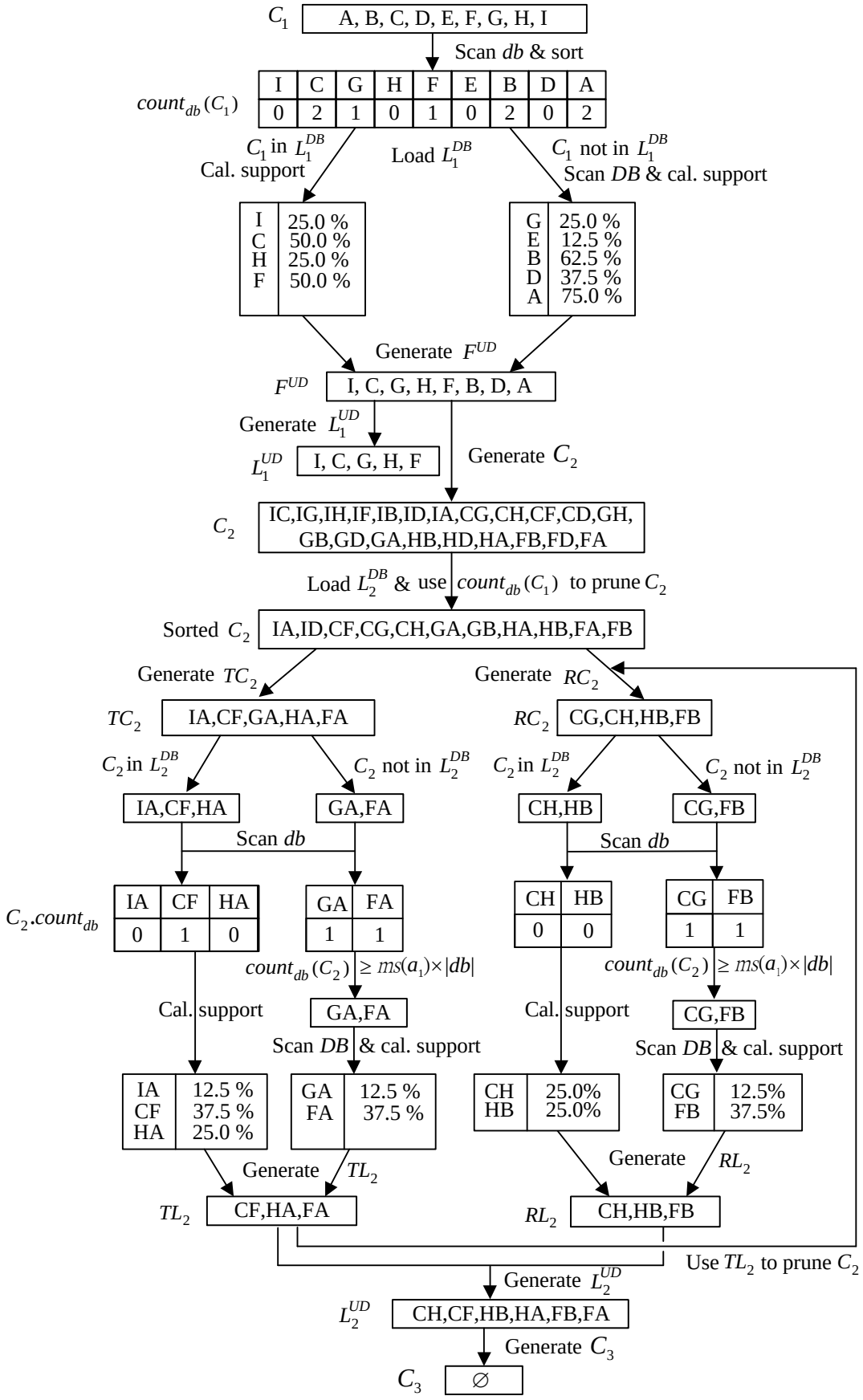


Figure 10. An example of UD_Stratify.

4. Experiments

In this section, we evaluate the performance of the two algorithms, UD_Cumulate and UD_Stratify, using the synthetic dataset generated by the IBM data generator [2]. We performed four experiments, changing a different parameter in each experiment. All the parameters except the one being varied were set to their default values, as shown in Table 14. All experiments were performed on an Intel Pentium-II 350 with 64MB RAM, running on Windows

Table 14. Default parameter settings for synthetic data generation.

Parameter		Default value
$\|DB\|$	Number of original transactions	100,000
$\|db\|$	Number of incremental transactions	10,000
$\|t\|$	Average size of transactions	5
N	Number of items	200
R	Number of groups	30
L	Number of levels	3
F	Fanout	5

Minimum Support: We use the following formula [9] to generate multiple minimum supports for each item a:

$$ms(a) = \begin{cases} \alpha \times sup_{DB}(a), & \alpha \times sup_{DB}(a) \geq 1.0 \\ 1.0, & \text{otherwise} \end{cases}$$

where $0.6 \geq \alpha \geq 0.1$ and $sup_{DB}(a)$ denotes the support of item a in the original database DB. The result is depicted in Figure 11. As shown in the figure, UD_Cumulate and UD_Stratify perform

significantly better than MMS_Stratify and MMS_Cumulate; the improvement ranges from 3 to 6 times and increases as α decreases. Besides, algorithm MMS_Stratify performs better than MMS_Cumulate for $\alpha \leq 0.2$, and algorithm UD_Cumulate performs slightly better than UD_Stratify.

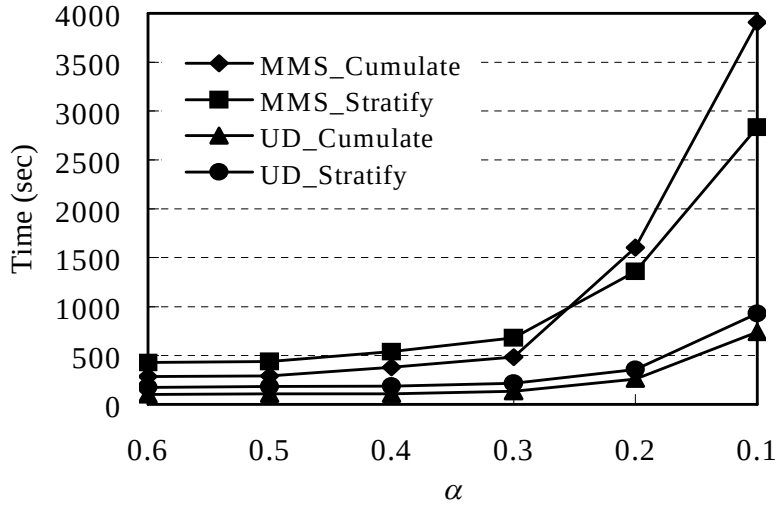


Figure 11. Performance comparison of MMS_Cumulate, MMS_Stratify, UD_Cumulate, and UD_Stratify for different α values.

Number of Incremental Transactions: We then compared the efficiency of these four algorithms under various sizes of incremental database. The number of transactions was varied from 10,000 to 85,000, and the minimum supports were specified with $\alpha = 0.6$. As shown in Figure 12, UD_Cumulate and UD_Stratify outperform MMS_Stratify and MMS_Cumulate, and algorithm UD_Cumulate performs better than UD_Stratify. The performance gap between UD_Stratify and MMS_Cumulate decreases as the incremental size increases. The reason is that at high α values, i.e., high minimum supports, the percentage of top candidates with generalized items becomes very small, and so UD_Stratify cannot prune significantly larger number of descendant itemsets to compensate for the cost of another database scan.

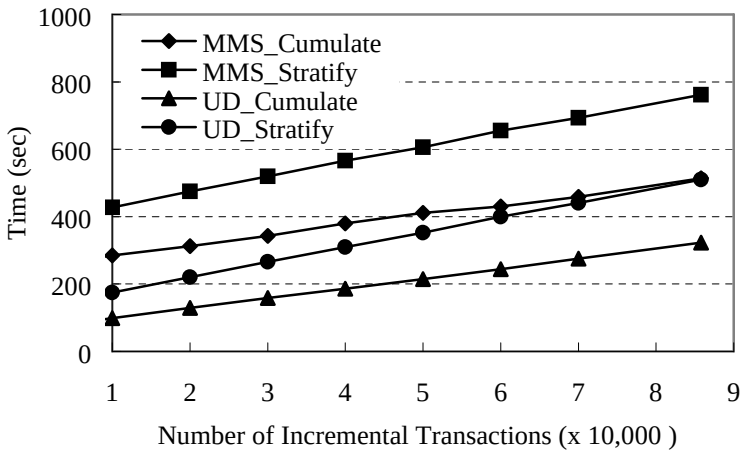


Figure 12. Performance comparison of MMS_Cumulate, MMS_Stratify, UD_Cumulate, and UD_Stratify for various sizes of incremental transactions.

Fanout: We changed the fanout from 3 to 9. Note that this corresponded to decreasing the number of levels. The results are shown in Figure 13. All algorithms performed faster as the fanout increased. This is because the cardinality as well as the number of generalized itemsets decreased upon increasing the number of levels. Also note that the performance gap between UD_Cumulate, UD_Stratify and MMS_Cumulate, MMS_Stratify decreased at high fanout since there were fewer candidate itemsets.

Number of Groups: We varied the number of groups from 5 to 20. As shown in Figure 14, the effect of increasing the number of groups is similar to that of increasing the fanout. The reason is that the number of items within a specific group decreases as the number of groups increases, so the probability of a generalized item decreases.

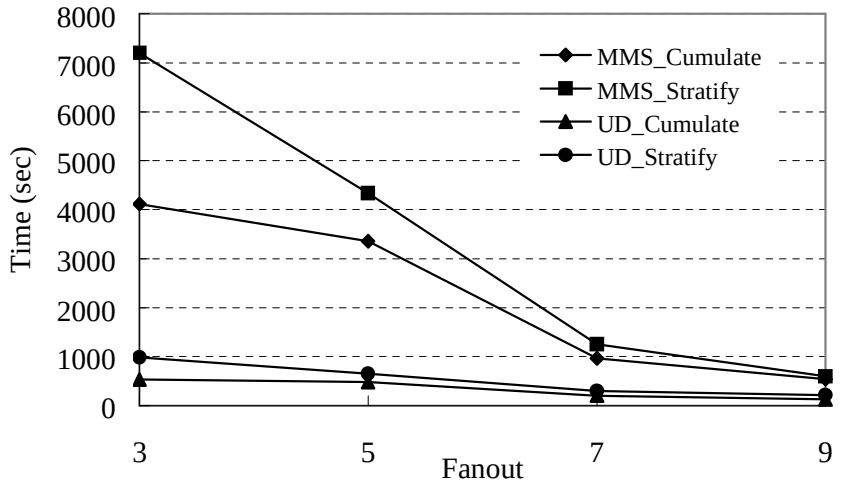


Figure 13. Performance comparison of MMS_Cumulate, MMS_Stratify, UD_Cumulate, and UD_Stratify for varying fanout.

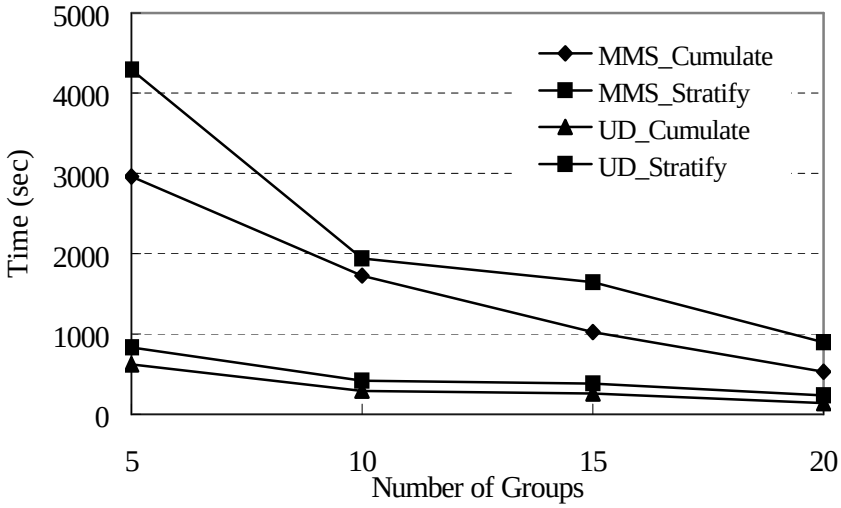


Figure 14. Performance comparison of MMS_Cumulate, MMS_Stratify, UD_Cumulate, and UD_Stratify for varying number of groups.

5. Related Work

The problem of incremental updating association rules was first addressed by Cheung et al. [4]. They coined the essence of updating the discovered association rules when new transaction records are added into the incremental database over time and proposed an algorithm called FUP (Fast UPdate). By making use of the discovered frequent itemsets, the proposed algorithm can dramatically reduce the efforts for computing the frequent itemsets in the updated database. They further examined the maintenance of multi-level association rules [5], and extended the model to incorporate the situations of deletion and modification [6]. All their approaches, however, did not recognize the varied support requirement inherent in items at different hierarchy levels.

Since then, a number of techniques have been proposed to improve the efficiency of incremental mining algorithm [8, 10, 12, 15]. In [12, 15], the authors proposed an incremental updating technique mainly based on the concept of *negative borders*. Empirical results showed that the approach could significantly save I/O access time for the maintenance of association rules.

In [10], Ng and Lam proposed an alternative incremental updating algorithm that incorporated the *dynamic counting* technique. Similar to its batch counterpart IDC [3], this algorithm can significantly reduce the number of database scans.

In [8], Hong et al. developed an incremental mining algorithm that was based on a novel concept of *pre-large itemsets*. According to two user-specified upper and lower support thresholds, their approach facilitated the pre-large itemsets (those itemsets having support larger than the lower support threshold) to refrain from rescanning the original database until the accumulated amount of inserted transactions exceeds a safety bound derived by pre-large concept. Their work, however, did not exploit association rules with generalized items, and did not consider multiple minimum supports.

To sum up, all previous works for incremental maintenance of association rules addressed in part the aspects discussed in this paper; no work, to our knowledge, has considered simultane-

ously both issues of taxonomy and non-uniform support specification. A summary of these works is described in Table 15.

Table 15. A summary of related work for incremental maintenance of association rules

Contributors	Model of incremental maintenance of association rules		
	Support specification	Exploiting taxonomy	Type of update
Cheung et al. [4]	uniform	no	insertion
Cheung et al. [5]	uniform	yes	insertion
Cheung et al. [6]	uniform	no	insertion, deletion and modification
Hong et al. [8]	uniform	no	insertion
Ng and Lam [10]	uniform	no	insertion
Sarda and Srinivas [12]	uniform	no	insertion
Thomas et al. [15]	uniform	no	insertion

6. Conclusions

In this paper, we have investigated the problem of maintaining association rules in the presence of taxonomy and multiple minimum supports. We presented two novel algorithms, UD_Cumulate and UD_Stratify, for maintaining multi-supported, generalized frequent itemsets. The proposed algorithms can incrementally update the discovered generalized association rules with non-uniform support specification and effectively reduce the number of candidate itemsets and database re-scanning. Empirical evaluation showed that these two algorithms not only were 2-6 times faster than running MMS_Cumulate or MMS_Stratify on the updated database afresh but also had good linear scale-up characteristic.

In the future, we will extend the maintenance of generalized association rules to a more general model that incorporates the situations of deletion and modification, and fuzzy taxonomic structures. We will also investigate the problem of on-line discovery and maintenance of multi-dimensional association rules from data warehouse data.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases, in: *Proc. of 1993 ACM-SIGMOD International Conference on Management of Data*, 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in: *Proc. of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487-499.

[3] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, Dynamic Itemset Counting and Implication Rules for Market-Basket Data, in: *Proc. of 1997 ACM-SIGMOD International Conference on Management of Data*, pp. 207-216, 1997.

[4] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental update technique, in: *Proc. of the 12th 1996 International Conference on Data Engineering*, 1996, pp.106-114.

[5] D.W. Cheung, V.T. Ng, B.W. Tam, Maintenance of discovered knowledge: a case in multi-level association rules, in: *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 307-310.

[6] D.W. Cheung, S.D. Lee, and B. Kao, A general incremental technique for maintaining discovered association rules, in: *Proc. of the 5th International Conference on Database Systems for Advanced Applications (DASFAA'97)*, 1997, pp. 185-194.

[7] J. Han and Y. Fu, Discovery of multiple-level association rules from large databases, in: *Proc. of the 21st International Conference on Very Large Data Bases*, 1995, pp. 420-431.

[8] T.P. Hong, C.Y. Wang, Y.H. Tao, Incremental data mining based on two support thresholds, in: *Proc. of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, 2000, pp.436-439.

[9] B. Liu, W. Hsu, and Y. Ma, Mining association rules with multiple minimum supports, in: *Proc. of the 5th International Conference Knowledge on Discovery and Data Mining*, 1999, pp. 337-341.

[10] K.K. Ng and W. Lam, Updating of association rules dynamically, in: *Proc. of 1999 International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, 2000, pp. 84-91.

[11] J.S. Park, M.S. Chen, and P.S. Yu, An effective hash-based algorithm for mining association rules, in: *Proc. of 1995 ACM-SIGMOD International Conference on Management of Data*, 1995, pp.175-186.

[12] N.L. Sarda and N.V. Srinivas, An adaptive algorithm for incremental mining of association rules, in: *Proc. of the 9th International Workshop on Database and Expert Systems Applications (DEXA'98)*, 1998, pp. 240-245.

[13] A. Savasere, E. Omiecinski, and S. Navathe, An efficient algorithm for mining association rules in large databases, in: *Proc. of the 21st International Conference on Very Large Data Bases*, 1995, pp. 432-444.

[14] R. Srikant and R. Agrawal, Mining generalized association rules, in: *Proc. of the 21st International Conference on Very Large Data Bases*, 1995, pp. 407-419.

[15] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, An efficient algorithm for the incremental updation of association rules in large databases, in: *Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1997, pp. 263-266.

[16] M.C. Tseng and W.Y. Lin, Mining generalized association rules with multiple minimum supports, in: *Proc. of the 3rd International Conference on Data Warehousing and Knowledge Discovery*, 2001, pp. 11-20.