JIS

Incremental Maintenance of Generalized Association Rules under Taxonomy Evolution

Ming-Cheng Tseng

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

Wen-Yang Lin[1]

Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan

Rong Jeng

Department of Information Management, I-Shou University, Kaohsiung 840, Taiwan

Abstract

Mining association rules from large databases of business data is an important topic in data mining. In many applications, there are explicit or implicit taxonomies (hierarchies) over the items, so it may be more useful to find associations at different levels of the taxonomy than only at the primitive concept level. Previous work on the mining of generalized association rules, however, assumed that the taxonomy of items are kept unchanged, disregarding the fact that the taxonomy might be updated as new transactions are added into the database over time. Under this circumstance, how to effectively update the discovered generalized association rules to reflect the database change with the taxonomy evolution is a crucial task. In this paper, we examine this problem and propose two novel algorithms, called IDTE and IDTE2, which can incrementally update the discovered generalized association rules when the taxonomy of items is evolved with new transactions. Empirical evaluations show that our algorithms can maintain their performance even in large amounts of incremental transactions and high degree of taxonomy evolution, and is faster than applying the contemporary generalized association mining algorithms to the whole updated database.

Keywords: Data mining; frequent itemsets; generalized association rules; incremental maintenance; taxonomy evolution

[1] *Correspondence to: Wen-Yang Lin, Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan. E-mail: wylin@nuk.edu.tw*

1. Introduction

Mining association rules from large databases of business data, such as transaction records, is a significant topic in the research on data mining [1][2]. An association rule is an expression of the form $X \Rightarrow Y$, where X and Y are sets of items. Such a rule reveals that transactions in the database containing items in X tend to also contain items in Y, and the probability, measured as the fraction of transactions containing X that also contain Y, is called the *confidence* of the rule. The *support* of the rule is the fraction of the transactions that contain all items in both X and Y. For an association rule to be valid, the rule should satisfy a user-specified minimum support, called *ms*, and minimum confidence, called *mc*, respectively. The problem of mining association rules is to discover all association rules that satisfy *ms* and *mc*.

In many applications, there are explicit or implicit taxonomies (hierarchies) over the items, so it may be more useful to find associations at different levels of the taxonomy than only at the primitive concept level [3][4]. For example, consider the taxonomy of items in Figure 1, where "PC" denotes "Personal Computer", "PDA" denotes "Personal Digital Assistance" and "IBM TP" is a kind of Notebook. It is likely that the association rule,

$$\text{Systemax V} \Rightarrow \text{HP LaserJet (Support = 20\%, Confidence = 100\%)},$$

does not hold when the minimum support is set to 30%, but the following association rule may be valid,

$$\text{Desktop PC} \Rightarrow \text{HP LaserJet}.$$

To the best of our knowledge, most work to date on mining generalized association rules required the taxonomy to be static [3][4][5][6], ignoring the fact that the taxonomy may change as time passes and new transactions are continuously added into the original database [7]. For example, items corresponding to new products must be added into the taxonomy, and their insertion would further introduce new classifications if they are of new types. On the other hand, items and/or their classifications will also be abandoned if they are no longer produced. All of these changes would reshape the taxonomy and in turn would invalidate previously discovered generalized association rules and/or introduce new ones, not to mention the changes caused by the transaction update to the database. Under these circumstances, it is important to find a method to effectively update the discovered generalized association rules.

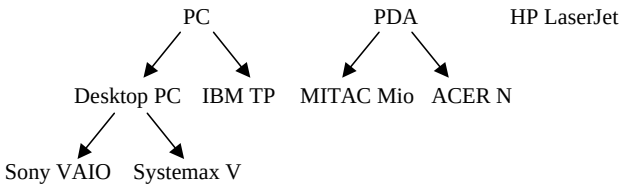


Figure 1. Example of taxonomy.

In this paper, we examine the problem of maintaining the discovered generalized association rules when new transactions are added to the original database along with taxonomy evolution. We give a formal problem description and clarify the situations for a taxonomy updating in Section 2.

A straightforward strategy to deal with this problem is to apply one of the generalized association mining methods to the whole database from scratch to find rules that reflect the most recent associations. This approach, however, has the following disadvantages:

1. The process of generating frequent itemsets is very time-consuming.
2. It is not cost-effective because the discovered frequent itemsets are not reused.

A more realistic and cost-effective alternative is to employ an association mining algorithm to generate the initial association rules, and when updates to the source database and taxonomy occur, apply an updating method to re-build the discovered rules. The challenge becomes that of developing an efficient updating algorithm to

facilitate the overall mining process. This problem is nontrivial because updates to the database and taxonomy not only can reshape the concept hierarchy and the form of generalized items, but also may invalidate some of the discovered association rules, thus turning previous weak rules into strong ones, and generating important new rules.

In this paper, we propose two algorithms, called IDTE (Incremental Database and Taxonomy Evolution) and IDTE2, for mining the generalized frequent itemsets. These algorithms are capable of effectively reducing the number of candidate sets and database rescanning, and so they can efficiently update the generalized association rules. A detailed description of the IDTE and IDTE2 algorithms is given in Section 3.

In Section 4, we use synthetic data to evaluate the performance of the proposed IDTE and IDTE2 with two leading generalized association mining algorithms, Cumulate and Stratify [4]. A remark on previous related work is presented in Section 5. Finally, we summarize our work and propose future investigations in Section 6.

2. Problem statement

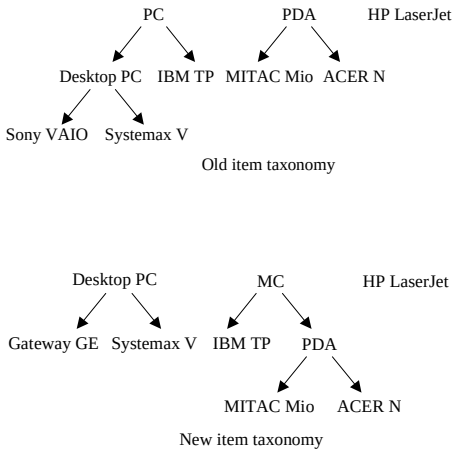
In real business applications the databases are changing over time, with new transactions (e.g., new types of items) continuously added and outdated transactions (e.g., abandoned items) deleted. Thus the taxonomy that represents the classification of items must also evolve to reflect such changes. This implies that if the updated database is processed afresh, some of 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.

2.1. Problem description

Consider the task of mining generalized frequent itemsets from a given transaction database *DB* with the item taxonomy *T*. In the literature, although different proposed methods have different strategies for implementation, the main process involves adding to each transaction the generalized items in the taxonomy [3][4][5][6]. For this reason, we can view the task as mining frequent itemsets from the extended database *ED*, the extended version of *DB*, by adding to each transaction the ancestors of each primitive item in *T*. We use L^{ED} to denote the set of newly discovered frequent itemsets.

Assume that the minimum support constraint *ms* is kept constant. Now let us consider the situation when new transactions in *db* are added to *DB*, and the taxonomy *T* is changed into a new one, $\overline{T}$. Following the previous paradigm, we can view the problem as follows. Let $\overline{ED}$ and $\overline{ed}$ denote the extended version of the original database *DB* and incremental database *db*, respectively, by adding to each transaction the generalized items in $\overline{T}$. Further, let $\overline{UE}$ be the updated extended database containing $\overline{ED}$ and $\overline{ed}$, i.e., $\overline{UE} = \overline{ED} + \overline{ed}$. The problem of updating L^{ED} when new transactions *db* are added to *DB*, and *T* is changed into $\overline{T}$, is equivalent to finding the set of frequent itemsets in $\overline{UE}$, denoted as $L^{\overline{UE}}$.

For illustration, consider an original database with the old item taxonomy and an incremental database with the new item taxonomy in Figure 2. Assume that primitive item “Sony VAIO” and generalized item “PC” are deleted from the “PC” group, and new primitive item “Gateway GE” is inserted into its top generalized item “MC”. The corresponding extended databases, including *ED*, *ed*, $\overline{ED}$, and $\overline{ed}$, are shown in Figure 3.



Original Database (*DB*)

TID	Items Purchased
1	Systemax V
2	IBM TP, HP LaserJet
3	Sony VAIO, Systemax V, MITAC Mio, ACER N
4	Sony VAIO, IBM TP
5	IBM TP, HP LaserJet
6	Systemax V
7	HP LaserJet
8	HP LaserJet

Incremental Database (*db*)

TID	Items Purchased
9	Gateway GE, HP LaserJet
10	Gateway GE, HP LaserJet

Figure 2. Example of original database DB with old item taxonomy and incremental database db with new item taxonomy.

Original Extended Database (*ED*)

TID	Primitive Items	Generalized Items
1	Systemax V	Desktop PC, PC
2	IBM TP, HP LaserJet	PC
3	Sony VAIO, Systemax V, MITAC Mio, ACER N	Desktop PC, PC, PDA
4	Sony VAIO, IBM TP	Desktop PC, PC
5	IBM TP, HP LaserJet	PC
6	Systemax V	Desktop PC, PC
7	HP LaserJet	
8	HP LaserJet	

Original Extended Incremental Database (*ed*)

TID	Primitive Items	Generalized Items
9	Gateway GE, HP LaserJet	
10	Gateway GE, HP LaserJet	

Updated Extended Database ($\overline{ED}$)

TID	Primitive Items	Generalized Items
1	Systemax V	Desktop PC
2	IBM TP, HP LaserJet	MC
3	Sony VAIO, Systemax V, MITAC Mio, ACER N	Desktop PC, PDA, MC
4	Sony VAIO, IBM TP	MC
5	IBM TP, HP LaserJet	MC
6	Systemax V	Desktop PC
7	HP LaserJet	
8	HP LaserJet	

Updated Extended Incremental Database ($\overline{ed}$)

TID	Primitive Items	Generalized Items
9	Gateway GE, HP LaserJet	Desktop PC
10	Gateway GE, HP LaserJet	Desktop PC

Figure 3. The corresponding extended databases: ED, ed, $\overline{ED}$, and $\overline{ed}$.

2.2. Situations for taxonomy evolution and frequent itemset update

By intuition, both the taxonomy evolution and its effect on previously discovered frequent itemsets are more complex than those for data updates because the taxonomy evolution induces item rearrangement: once an item, primitive or generalized, is reclassified into another category, all of its ancestors (generalized items) in both the old and the new taxonomy are affected. In this subsection, we describe different situations for taxonomy evolution, and clarify the essence of frequent itemset updates for each type of taxonomy evolution.

According to our observation, there are four basic types of item updates that will cause taxonomy evolution:

1. Item insertion: New items are added to the taxonomy.
2. Item deletion: Obsolete items are removed from the taxonomy.
3. Item renaming: Items are renamed for certain reasons, such as error correction, product promotion, etc.
4. Item reclassification: Items are reclassified into different categories.

Note that hereafter the term "item" refers to a *primitive* or a *generalized* item. In the following, we elaborate each type of evolution.

Type 1: Item insertion. There are different strategies to handle this type of update operation, depending on whether the inserted item is a primitive or a generalized item.

When the new inserted item is primitive, we do not have to process it until an incremental database update containing that item occurs. This is because the new item does appear in neither the original database, nor the discovered associations. However, if the new item is a generalization, then the insertion will affect the discovered associations since a new generalization often incurs some item reclassification.

Example 1. Figure 4 shows this type of taxonomy evolution. In Figure 4(a), a new primitive item "ASUS W" is inserted into the generalized item "PC". Because the new item "ASUS W" does not appear in the original transactions, and so is not in the original set of frequent itemsets, we do not have to process it until there is an incremental database update. Next, we consider the case of a new generalization insertion. In Figure 4(b), a generalized item "Desktop PC" is inserted as a child of the generalized item "PC", and items "Sony VAIO" and "Systemax V" are reclassified to the new generalization, "Desktop PC". Since items "Sony VAIO" and "Systemax V" already exist in the original database, we must process them to update the generalized frequent itemsets.

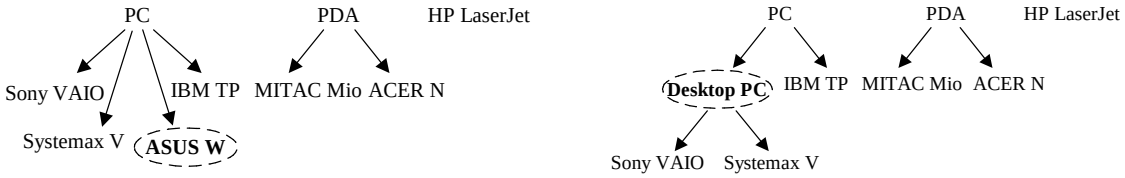


Figure 4. Example of taxonomy evolution caused by item insertion. The inserted new item is: (a) primitive; (b) generalized.

Type 2: Item deletion. Unlike the case of item insertion, the deletion of a primitive item from the taxonomy would incur a problem of inconsistency. In other words, if there is no transaction update to delete the occurrence of that item, then the refined item taxonomy will not conform to the database. An outdated item continues to appear in the transaction of interest. To simplify this problem, we assume that the updated extended database is always consistent with the evolution of the taxonomy. Additionally, the removal of a generalization may also lead to item reclassification. Therefore, we also have to deal with the situation caused by item deletion.

Example 2. Figure 5 shows this type of taxonomy evolution, where Figure 5(a) illustrates the deletion of the primitive item "Sony VAIO", and Figure 5(b) demonstrates the removal of the generalized item "Desktop PC" and the reclassification of "Sony VAIO" and "Systemax V" to "PC".

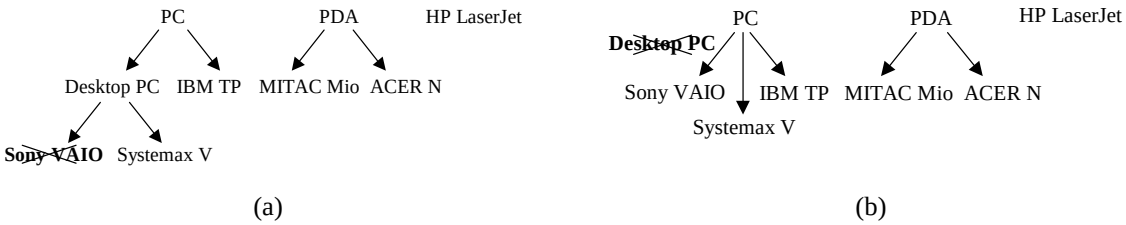


Figure 5. Example of taxonomy evolution caused by item deletion: (a) primitive; (b) generalized.

Type 3: Item renaming. Items may be renamed for many reasons such as error correction, product promotion, or just for novelty. When items are renamed, we do not have to process the database since the processing codes of the renamed items are the same. Instead, we just replace the discovered frequent itemsets and the association rules with the new names.

Example 3. Figure 6 shows this type of taxonomy evolution, where the generalized item "Desktop PC" is renamed to "Desktop PC Pro", and the primitive item "MITAC Mio" is renamed to "MITAC Mio Pro".

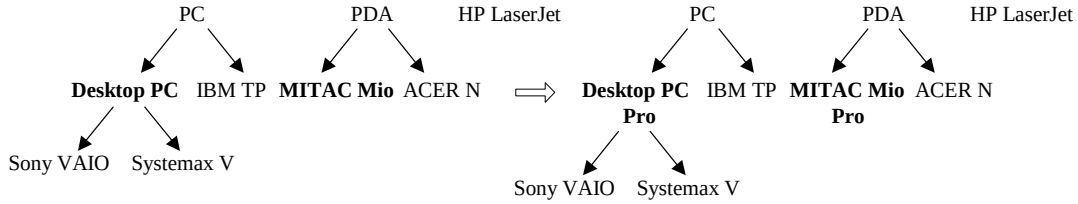


Figure 6. Example of taxonomy evolution caused by item renaming.

Type 4: Item reclassification. Among all types of taxonomy updates this is the most far-reaching operation. Once an item, primitive or generalized, is reclassified into another category, all of its ancestors (generalized items) in the old as well as the new taxonomy are affected. That is, the supports of these affected generalized items must be recounted, as are the frequent itemsets containing any one of the affected generalized items.

Example 4. Consider Figure 7. A new group "MC" denoting "Mobile Computer" is added along with the reclassification of "IBM TP" and "PDA" to "MC". Therefore, the shifted item "IBM TP" will change the support count of the generalized items "PC", and "MC", and also affect the support count of any itemsets containing "PC" or "MC".

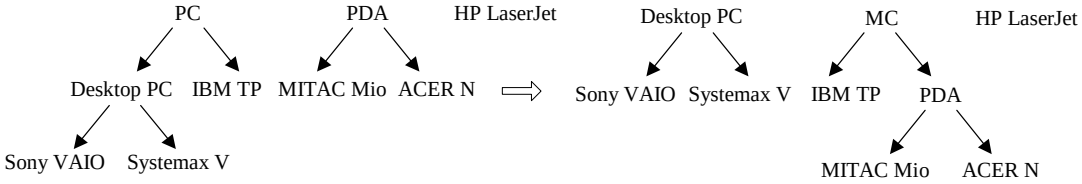


Figure 7. Example of a taxonomy evolution caused by item reclassification.

3. The proposed algorithms

Two algorithms for updating discovered frequent itemsets under incremental database update and taxonomy evolution, called IDTE and IDTE2, are proposed. We first introduce the basic paradigm, and then detail the process of each algorithm.

3.1. Algorithm IDTE

3.1.1 Basic paradigm

A straightforward way to find updated generalized frequent itemsets would be to run any of the algorithms for finding generalized frequent itemsets, such as Cumulate and Stratify [4], on $\overline{UE}$. This simple way, however, ignores some of the discovered frequent itemsets are not affected by incremental update and/or taxonomy evolution; that is, these itemsets survive in the taxonomy evolution and remain frequent in the updated database $\overline{UE}$. If we can identify the unaffected itemsets, then we can avoid unnecessary computations in counting their supports. In view of this, we decided to adopt the Apriori-based maintenance framework depicted in Figure 8.

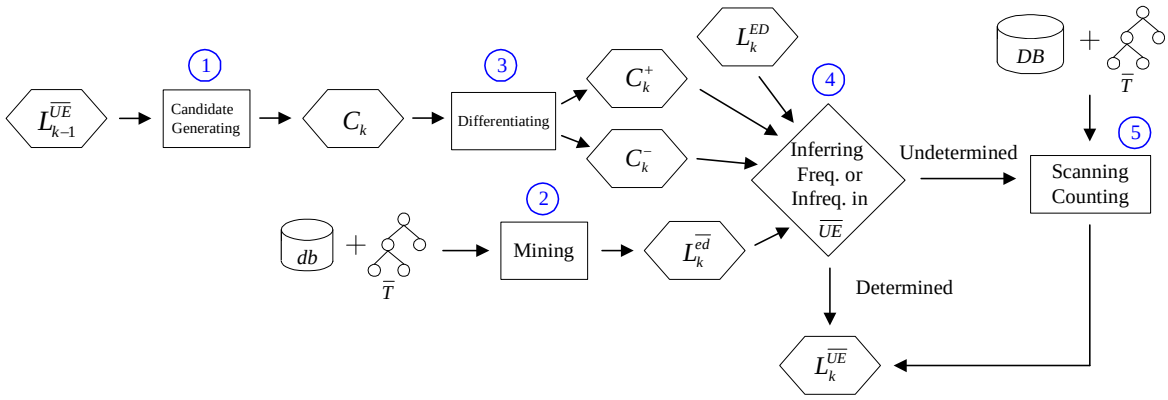


Figure 8. Proposed Apriori-based framework for updating the frequent k-itemsets.

Each pass of mining the frequent *k*-itemsets involves the following main steps:

1. Generate candidate *k*-itemsets C_k.
2. Scan the incremental database *db* with the new taxonomy $\overline{T}$ to find $L_k^{\overline{ed}}$.
3. Differentiate in C_k the affected itemsets (C_k^+) from the unaffected ones (C_k^-).
4. Incorporate L_k^{ED}, $L_k^{\overline{ed}}$, C_k^+, and C_k^- to determine whether a candidate itemset is frequent or not in the resulting database $\overline{UE}$.
5. Scan *DB* with $\overline{T}$, i.e., $\overline{ED}$, to count the supports of itemsets that are undetermined in Step 4.

In summary, our approach is to differentiate the unaffected itemsets by the taxonomy evolution from the affected ones, and then use them to reduce the work spent on support counting of itemsets. Below, we first elaborate on each of the kernel steps, i.e., Steps 3 and 4, and then present a description of algorithm IDTE and illustrate its execution with a simple example. The notation that will be used is summarized in Table 1.

Table 1. Summary of the notation used in IDTE.

Notation	Definition	Notation	Definition
DB	Original database	C_k	Candidate k-itemsets
db	Incremental database	C_k^+	Affected k-itemsets
UD	Updated database $UD = DB + db$	C_k^-	Unaffected k-itemsets
T	Old item taxonomy	$L_k^{\overline{ed}}$	Frequent k-itemsets in $\overline{ed}$
$\overline{T}$	New item taxonomy	$L_k^{\overline{UE}}$	Frequent k-itemsets in $\overline{UE}$
ED	Extension of DB with generalized items in T	I	Set of primitive items in T
ed	Extension of db with generalized items in T	$\overline{I}$	Set of primitive items in $\overline{T}$
UE	Updated extended database $UE = ED + ed$	J	Set of generalized items in T
$\overline{ED}$	Extension of DB with generalized items in $\overline{T}$	$\overline{J}$	Set of generalized items in $\overline{T}$
$\overline{ed}$	Extension of db with generalized items in $\overline{T}$	$count_{DB}(A)$	Support count of an itemset A
$\overline{UE}$	Updated extended database $\overline{UE} = \overline{ED} + \overline{ed}$	$sup_{DB}(A)$	Supports of an itemset A
L_k^{ED}	Frequent k-itemsets in ED		

3.1.2 Differentiation of affected and unaffected itemsets

In this subsection, we elaborate on the way for differentiating affected and unaffected itemsets. First we introduce the terms "affected item" and "unaffected item".

Definition 1. An item (primitive or generalized) is called an *affected item* if its support would be changed with respect to the taxonomy evolution; otherwise, it is called an *unaffected item*.

Consider an item $x \in T \cup \overline{T}$, and the three independent subsets $T - \overline{T}$, $\overline{T} - T$ and $T \cap \overline{T}$. There are three different cases in differentiating whether x is an affected item or not. For simplicity, we ignore the case that x is a renamed item, which can be simply regarded as an unaffected item.

1. $x \in T - \overline{T}$. In this case, x is an obsolete item. Then the support of x in the updated database should be counted as 0 no matter whether x is a primitive or a generalized item, and so x is an affected item.
2. $x \in \overline{T} - T$. In this case, x denotes a new item, whose support may change from zero to nonzero. Thus, x should be regarded as an affected item.
3. $x \in T \cap \overline{T}$. This case is more complex than the previous ones since the situations are different, depending on whether x is a primitive or a generalized item, as clarified in the following two lemmas.

Lemma 1. Consider a primitive item $x \in I \cup \overline{I}$. Then $count_{\overline{ED}}(x) = count_{ED}(x)$ if $x \in I \cap \overline{I}$.

Proof. This lemma follows from the fact that the original transactions in ED and $\overline{ED}$ that contains x are the same. ■

Lemma 2. Consider a generalized item $x \in J \cap \overline{J}$. If $pdes_T(x) = pdes_{\overline{T}}(x)$, then $count_{ED}(x) = count_{\overline{ED}}(x)$, where $pdes_T(x)$ and $pdes_{\overline{T}}(x)$ denote the sets of primitive descendant items of x in T and $\overline{T}$, respectively.

Proof. Consider a primitive descendant of *x*, say *y*. From the proof of Lemma 1, we know that the original transactions containing *y* in *ED* and $\overline{ED}$ remain unchanged. Furthermore, as a generalization of *y*, *x* must appear in the extended part of each transaction in which *y* appears. The lemma then follows if $pdes_T(x) = pdes_{\overline{T}}(x)$. ■

For example, in Figure 5, the generalized items "Desktop PC" and "PDA" are unaffected items since their primitive descendants do not change after the taxonomy evolution. Similarly in Figure 4(b), "PC" and "PDA" are unaffected by the insertion of the new generalization "Desktop PC".

In summary, Lemmas 1 and 2 state that an item is unaffected by a taxonomy evolution if it is a primitive item that survived after the taxonomy evolution, or if it is a generalized item whose primitive descendant set remains unchanged.

Definition 2. For a candidate itemset *A*, we say *A* is an *affected itemset* if it contains at least one affected item.

Lemma 3. Consider an itemset *A*. Then

1. *A* is an unaffected itemset with $count_{\overline{ED}}(A) = count_{ED}(A)$, if *A* contains unaffected items only, and
2. $count_{\overline{ED}}(A) = 0$ if *A* contains at least one item *x*, for $x \in T - \overline{T}$, or if *A* contains at lease one new primitive item, i.e., $\exists\, x \in A, x \in \overline{I} - I$.

Proof:

1. If *A* contains unaffected items only, then the transactions that contains *A* in *ED* and $\overline{ED}$ are the same. Therefore, $count_{\overline{ED}}(A) = count_{ED}(A)$.
2. This is straightforward because any itemset containing an obsolete item should be deleted, and any itemset containing a new primitive item could not appear in $\overline{ED}$. ■

3.1.3 Inference of frequent and infrequent itemsets

Now that we have clarified how to differentiate the unaffected and affected itemsets, we will further show how to utilize this information to determine in advance whether or not an itemset is frequent before scanning the extended database $\overline{ED}$, and show the corresponding actions in counting the supports of itemsets. Consider a candidate itemset *A* generated during the mining process. We observe that there are five different cases.

1. If *A* is an unaffected itemset and is frequent in *ED* and $\overline{ed}$, then it is also frequent in the updated extended database $\overline{UE}$.
2. If *A* is an unaffected itemset and is infrequent in *ED* and $\overline{ed}$, then it is also infrequent in $\overline{UE}$.
3. If *A* is an unaffected itemset and is frequent in *ED* but infrequent in $\overline{ed}$, then *A* is an undetermined itemset in $\overline{UE}$; but a simple calculation can determine whether or not *A* is frequent in $\overline{UE}$.
4. If *A* is an unaffected infrequent itemset in *ED* but frequent in $\overline{ed}$, then it is an undetermined itemset in $\overline{UE}$, i.e., it may be either frequent or infrequent.
5. If *A* is an affected itemset then it is an undetermined itemset in $\overline{UE}$, no matter whether it is frequent or infrequent in *ED* and is frequent or infrequent in $\overline{ed}$.

These five cases are summarized in Table 2. Note that only Cases 4 and 5 require an additional scan of $\overline{ED}$ to determine the support count of *A* in $\overline{UE}$. That is, we have utilized the information on unaffected itemsets and discovered frequent itemsets in order to avoid such a database scan. For Case 4, after scanning $\overline{ed}$ and comparing its support with *ms*, if *A* is frequent in $\overline{ed}$, *A* may become frequent in $\overline{UE}$. Then we need to scan $\overline{ED}$ to

determine the support count of *A*. For Case 5, since *A* is an affected itemset, its support could be changed in $\overline{ED}$. Therefore, a further scan of $\overline{ED}$ is also required to decide whether or not it is frequent.

Table 2. Five cases for inferring whether a candidate itemset is frequent or not.

$T \to \overline{T}$	L^{ED}	$\overline{ed}$	$\overline{UE}$	Action	Case
unaffected	$\in$	frequent	frequent	no	1
		infrequent	undetermined	compare $sup_{\overline{UE}}(A)$ with *ms*	3
	$\notin$	frequent	undetermined	scan $\overline{ED}$	4
		infrequent	infrequent	no	2
affected	$\in$, $\notin$	frequent, infrequent	undetermined	scan $\overline{ED}$	5

3.1.4 Algorithm description and example

Based on the aforementioned concepts, the IDTE algorithm is presented in Figure 9. First, generate the candidate *k*-itemsets C_k from the frequent (*k*-1)-itemsets $L_{k-1}^{\overline{UE}}$. Next, scan $\overline{ed}$ for C_k and generate $L_k^{\overline{ed}}$. Then load the original frequent *k*-itemsets L_k^{ED} and divide C_k according to Lemma 3 into three subsets: $C_>^-$, $C_<^-$ and C_k^+, which denote the set of unaffected *k*-itemsets in L_k^{ED}, the set of unaffected *k*-itemsets not in L_k^{ED}, and the set of affected *k*-itemsets, respectively. For each member of the set $C_>^-$, accumulate its support count in $\overline{ed}$ and $\overline{ED}$, compare the result with *ms* for Case 3, and if frequent, put it into $L_k^{\overline{UE}}$. For the set $C_<^-$, only those itemsets being frequent in $\overline{ed}$ need undergo an additional scan of $\overline{ED}$ (Case 4); and the infrequent ones are pruned immediately (Case 2). After getting the support count in $\overline{ED}$, accumulate the support count for each *k*-itemset in $\overline{ed}$ and $\overline{ED}$, compare the resulting support with *ms* and if frequent, put it into $L_k^{\overline{UE}}$. The procedure for dealing with the set C_k^+ is the same as that for $C_<^-$ in Case 4. The IDTE algorithm is shown in Figure 9.

Consider the example in Figures 2 and 3. For simplicity, let item "A" stand for "PC", "B" for "Desktop PC", "C" for "Sony VAIO", "D" for "IBM TP", "E" for "Systemax V", "F" for "PDA", "G" for "MITAC Mio", "H" for "ACER N", "I" for "HP LaserJet", "J" for "MC", and "K" for "Gateway GE". The resulting item taxonomy is shown in Figure 10. Let *ms* = 20%. The set of frequent itemsets L^{ED} is shown in Table 3. The overall process of running this example using IDTE is illustrated in Figure 11.

Correctness: The correctness of algorithm IDTE lies mainly in two aspects: the procedure for candidate generation and that for inferring which candidate itemsets are frequent. The correctness of the former has been justified in [2]. For the latter, we need to verify that all cases dictated in Table 2. We only show the first case; the others can be proved in a similar way. Consider an unaffected candidate itemset *A*. From Lemma 3, $count_{ED}(A) = count_{\overline{ED}}(A)$. Since *A* is frequent in *ED* and $\overline{ed}$, we have $count_{ED}(A) \geq |ED| \times ms$ and $count_{\overline{ed}}(A) \geq |\overline{ed}| \times ms$. Then $count_{\overline{UE}}(A) = count_{\overline{ED}}(A) + count_{\overline{ed}}(A) \geq (|\overline{ED}| + |\overline{ed}|) \times ms = |\overline{UE}| \times ms$. That is, $sup_{\overline{UE}}(A) \geq ms$ and so *A* is frequent in $\overline{UE}$.

Input: (1) *DB*: original database; (2) *db*: incremental database; (3) *ms*: minimum support setting; (4) *T*: old item taxonomy; (5) $\overline{T}$: new item taxonomy; (6) L^{ED} : set of original frequent itemsets.

Output: $L^{\overline{UE}}$: set of new frequent itemsets with respect to $\overline{T}$ and $\overline{ed}$.

Steps:

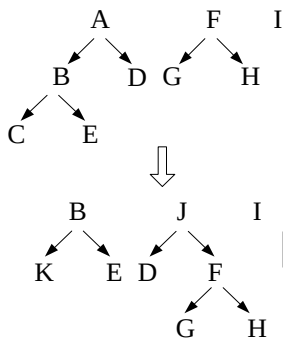
1. Identify affected items;
2. $k = 1$;
3. repeat
4. **if** $k = 1$ **then** generate C_1 from $\overline{T}$;
5. **else** C_k = apriori-gen($L_{k-1}^{\overline{UE}}$);
6. Delete any candidate in C_k that consists of an item and its ancestor;
7. Scan $\overline{ed}$ to count $count_{\overline{ed}}(A)$ for each itemset A in C_k;
8. $L_k^{\overline{ed}} = \{A \mid A \in C_k \text{ and } sup_{\overline{ed}}(A) \geq ms\}$;
9. Load original frequent k-itemsets L_k^{ED} ;
10. Divide C_k into three subsets: $C_>^-$, $C_<^-$ and C_k^+ ;
11. **for** each $A \in C_>^-$ **do** /* Cases 1 & 3 */
12. Assign $count_{\overline{ED}}(A) = count_{ED}(A)$;
13. Delete any candidate A from $C_<^-$ if $A \notin L_k^{\overline{ed}}$; /* Case 2 */
14. **for** each $A \in C_<^- \cup C_k^+$ **do** /* Cases 4 & 5 */
15. **if** A contains no new primitive item **then** /* Lemma 3 */
16. Count the occurrences of A over $\overline{ED}$;
17. Calculate $count_{\overline{UE}}(A) = count_{\overline{ED}}(A) + count_{\overline{ed}}(A)$ for each itemset A in C_k;
18. $L_k^{\overline{UE}} = \{A \mid A \in C_k \text{ and } sup_{\overline{UE}}(A) \geq ms\}$;
19. **until** $L_k^{\overline{UE}} = \varnothing$
20. $L^{\overline{UE}} = \cup_k L_k^{\overline{UE}}$;

Figure 9. Algorithm IDTE.

3.2. Algorithm IDTE2

We now propose another algorithm called IDTE2. The main difference between IDTE and IDTE2 is in the approach for inferring whether or not an affected candidate itemset is frequent in the updated whole database, i.e., corresponding to Case 5 of Table 2. We observe that it is not always necessary to scan the whole database $\overline{ED}$ to know the occurrences of an affected candidate itemset and to determine if it is frequent in $\overline{UE}$. Indeed, it suffices for the purpose if we know which part of the transactions is affected by the taxonomy evolution.

Definition 3. A transaction is called an *affected transaction* if it contains at least one of the affected primitive items with respect to a taxonomy evolution.



Original Extended Database (*ED*)

TID	Primitive Items	Generalized Items
1	E	A, B
2	D, I	A
3	C, E, G, H	A, B, F
4	C, D	A, B
5	D, I	A
6	E	A, B
7	I	
8	I	

Updated Extended Database ($\overline{ED}$)

TID	Primitive Items	Generalized Items
1	E	B
2	D, I	J
3	C, E, G, H	B, F, J
4	C, D	J
5	D, I	J
6	E	B
7	I	
8	I	

Original Extended Incremental Database (*ed*)

TID	Primitive Items	Generalized Items
9	K, I	
10	K, I	

Updated Extended Incremental Database ($\overline{ed}$)

TID	Primitive Items	Generalized Items
9	K, I	B
10	K, I	B

Figure 10. Example of taxonomy evolution and incremental database.

Table 3. Summary for frequent itemsets and counts generated from ED.

L_1^{ED}	Counts	L_2^{ED}	Counts	C_3& L_3^{ED}
A	6	AI	2	∅
B	4	DI	2	
C	2			
D	3			
E	3			
I	4			

For example, transactions 1 and 2 are affected transactions in Figure 12 because they contain affected items "G" and "E".

Let Δ and $\overline{\Delta}$ denote the part of affected transactions in *ED* and $\overline{ED}$, respectively. Note that $|\overline{ED}| = |ED|$, and $\overline{ED} = ED - \overline{\Delta} + \Delta$. Then for a candidate itemset *A*, we have $count_{\overline{ED}}(A) = count_{ED}(A) - count_{\Delta}(A) + count_{\overline{\Delta}}(A)$ and $count_{\overline{UE}}(A) = count_{\overline{ED}}(A) + count_{\overline{ed}}(A)$. Therefore, if a candidate itemset *A* is an affected itemset and is frequent in *ED*, we only have to scan Δ and $\overline{\Delta}$ to decide whether or not *A* is frequent in $\overline{UE}$.

As for the affected candidate itemsets that are not frequent in *ED*, if *A* is frequent in $\overline{ed}$, then we still have to scan $\overline{ED}$ to determine the occurrences of *A* in $\overline{ED}$; otherwise, let us consider the following lemma.

Lemma 4. If an affected itemset $A \notin L^{ED}$ and $count_{\overline{\Delta}}(A) \leq count_{\Delta}(A)$, then $A \notin L^{\overline{ED}}$.

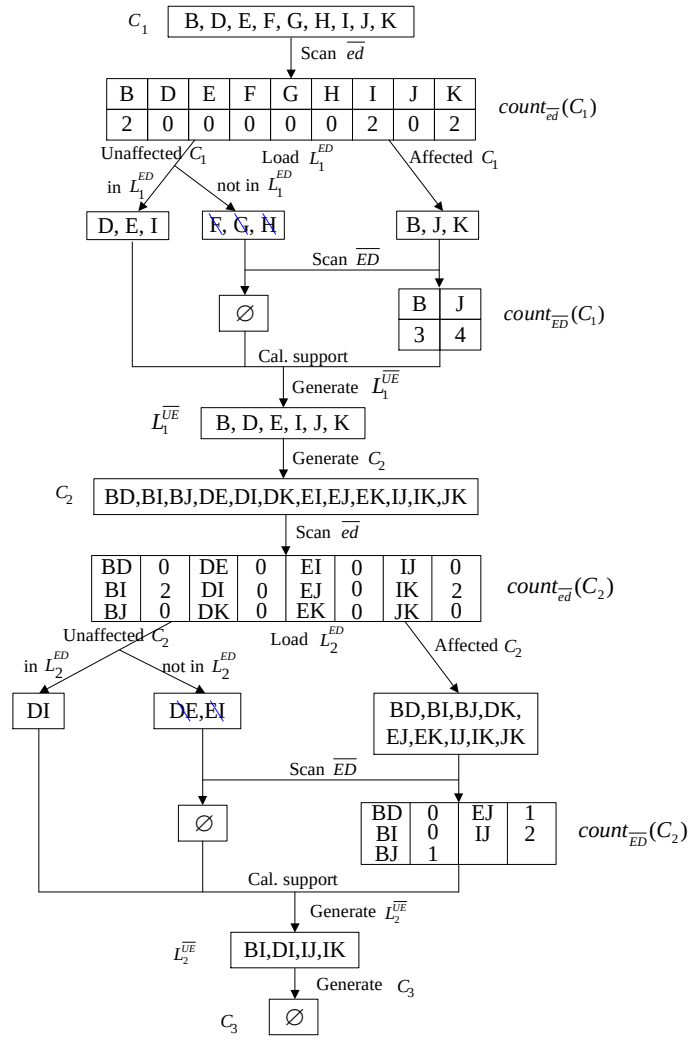


Figure 11. Illustration of algorithm IDTE.

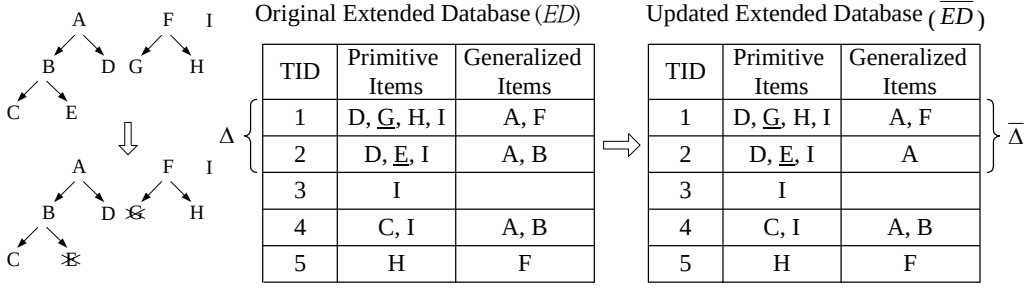
Proof. If $A \notin L^{ED}$, then $count_{ED}(A) < |ED| \times ms$. Note that $|ED| = |\overline{ED}|$ and $|\Delta| = |\overline{\Delta}|$. Hence $count_{\overline{ED}}(A) = count_{ED}(A) + (count_{\overline{\Delta}}(A) - count_{\Delta}(A)) < |ED| \times ms = |\overline{ED}| \times ms$. Thus, $A \notin L^{\overline{ED}}$. ■

In light of Lemma 4, we scan the affected transactions in Δ and $\overline{\Delta}$, respectively, to count the occurrences of A. If the support count of A in $\overline{\Delta}$ is no larger than that in Δ, i.e., $count_{\overline{\Delta}}(A) \le count_{\Delta}(A)$, then A must be infrequent in $\overline{ED}$ and so can be pruned immediately; otherwise we have to scan $\overline{ED} - \overline{\Delta}$ to decide whether A is frequent or not.

Example 5. Consider Figure 12. Assume $ms = 60\%$ (3 transactions). Then items "B" and "F" are not frequent in ED. That is, $count_{ED}(\mathrm{B})$ and $count_{ED}(\mathrm{F})$ are not available in L_1^{ED}. From Δ and $\overline{\Delta}$, we have $count_{\Delta}(\mathrm{B}) = count_{\Delta}(\mathrm{F}) = 1$, $count_{\overline{\Delta}}(\mathrm{B}) = 0$, $count_{\overline{\Delta}}(\mathrm{F}) = 1$, $count_{\overline{\Delta}}(\mathrm{B}) - count_{\Delta}(\mathrm{B}) = -1$, and $count_{\overline{\Delta}}(\mathrm{F}) - count_{\Delta}(\mathrm{F}) = 0$. Items "B" and "F" are not frequent in $\overline{ED}$ according to Lemma 4. Therefore, there is no need to

scan the transactions in $\overline{ED} - \overline{\Delta}$ to determine whether items "B" and "F" are frequent or not. This also reduces the number of candidates invoking the scanning of $\overline{ED} - \overline{\Delta}$.

Table 4 summarizes the previous discussions and refines the cases for inferring frequent/infrequent itemsets and shows the corresponding actions. A further improvement for Case 7 is possible, as described below.



TID	Primitive Items	Generalized Items
1	D, G, H, I	A, F
2	D, E, I	A, B
3	I	
4	C, I	A, B
5	H	F

TID	Primitive Items	Generalized Items
1	D, G, H, I	A, F
2	D, E, I	A
3	I	
4	C, I	A, B
5	H	F

Figure 12. Illustration of the concept of affected transactions.

Table 4. Seven cases arising from the updated extended incremental database and the taxonomy evolution.

$T \to \overline{T}$	L^{ED}	$\overline{ed}$	$\overline{UE}$	Action	Case
unaffected	$\in$	frequent	frequent	no	1
		infrequent	undetermined	compare $sup_{\overline{UE}}(A)$ with *ms*	3
	$\notin$	frequent	undetermined	scan $\overline{ED}$	4
		infrequent	infrequent	no	2
affected	$\in$	frequent, infrequent	undetermined	scan Δ & $\overline{\Delta}$, cal. $count_{ED}(A)$ $- count_{\Delta}(A) + count_{\overline{\Delta}}(A) + count_{\overline{ed}}(A)$	5
	$\notin$	frequent	undetermined	scan $\overline{ED}$	6
		infrequent	undetermined	scan Δ & $\overline{\Delta}$, if $count_{\overline{\Delta}}(A) > count_{\Delta}(A)$, then scan $\overline{ED} - \overline{\Delta}$	7

Lemma 5. If an itemset *A* contains at least one new generalized item, i.e., $\exists\ x \in A$, $x \in \overline{T} - T$, then $count_{\overline{ED} - \overline{\Delta}}(A) = 0$.

Proof. This is straightforward since a new generalized item only exists in $\overline{\Delta}$, and so it does not appear in $\overline{ED} - \overline{\Delta}$. ■

The IDTE2 algorithm is described in Figure 13.

For illustration, let us consider Figure 10 and let *ms* = 20% again. Transactions 1 to 6 in Figure 10 are affected transactions. The overall process of running this example using IDTE2 is illustrated in Figure 14.

Correctness: Note that IDTE2 differs from IDTE in the approach for inferring whether or not an affected candidate itemset is frequent afterwards. This corresponds to the Cases 5 and 7 in Table 4, whose correctness have been exposed in Lemmas 4 and 5.

Input: (1) *DB*: original database; (2) *db*: incremental database; (3) *ms*: minimum support setting; (4) *T*: old item taxonomy; (5) $\overline{T}$: new item taxonomy; (6) L^{ED} : set of original frequent itemsets.

Output: $L^{\overline{UE}}$: set of new frequent itemsets with respect to $\overline{T}$ and $\overline{ed}$.

Steps:

1. Identify affected items;
2. Identify affected transactions, Δ and $\overline{\Delta}$;
3. $k = 1$;
4. **repeat**
5. **if** $k = 1$ **then** generate C_1 from $\overline{T}$;
6. **else** C_k = apriori-gen($L_{k-1}^{\overline{UE}}$);
7. Delete any candidate in C_k that consists of an item and its ancestor;
8. Scan $\overline{ed}$ to count $count_{\overline{ed}}(A)$ for each itemset A in C_k;
9. $L_k^{\overline{ed}} = \{A \mid A \in C_k \text{ and } sup_{\overline{ed}}(A) \geq ms\}$;
10. Load original frequent k-itemsets L_k^{ED};
11. Divide C_k into four subsets: $C_>^-$, $C_<^-$, $C_>^+$ and $C_<^+$; /* $C_>^+$ and $C_<^+$ denote affected itemsets in L_k^{ED} and not in L_k^{ED}, respectively */
12. **for** each $A \in C_>^-$ or $A \in C_>^+$ **do** /* Cases 1, 3 & 5 */
13. Assign $count_{\overline{ED}}(A) = count_{ED}(A)$;
14. Delete any candidate A from $C_<^-$ if $A \notin L_k^{\overline{ed}}$; /* Case 2 */
15. Count the occurrences for each A in $C_>^+$ or $C_<^+$ over Δ; /* Cases 5 & 7 */
16. Count the occurrences for each A in $C_<^-$, $C_>^+$ or $C_<^+$ over $\overline{\Delta}$; /* Cases 4, 5, 6 & 7 */
17. **for** each $A \in C_>^+$ **do** /* Case 5 */
18. Calculate $count_{ED}(A) - count_{\Delta}(A) + count_{\overline{\Delta}}(A) + count_{\overline{ed}}(A)$;
19. Delete any candidate A from $C_<^+$ if $A \notin L_k^{\overline{ed}}$ and $count_{\overline{\Delta}}(A) \leq count_{\Delta}(A)$; /* Case 7 & Lemma 4 */
20. **for** each $A \in C_<^-$ **do** /* Case 4 */
21. **if** A contains no new primitive item **then** count the occurrences of A over $\overline{ED} - \overline{\Delta}$; /* Lemma 3 */
22. **for** each $A \in C_<^+$ **do** /* Case 6 & 7 */
23. **if** A contains no new item **then** count the occurrences of A over $\overline{ED} - \overline{\Delta}$; /* Lemmas 3 & 5 */
24. Calculate $count_{\overline{UE}}(A) = count_{\overline{ED}}(A) + count_{\overline{ed}}(A)$ for each $A \in C_k$ in C_k;
25. $L_k^{\overline{UE}} = \{A \mid A \in C_k \text{ and } sup_{\overline{UE}}(A) \geq ms\}$;
26. **until** $L_k^{\overline{UE}} = \varnothing$
27. $L^{\overline{UE}} = \bigcup_k L_k^{\overline{UE}}$;

Figure 13. Algorithm IDTE2.

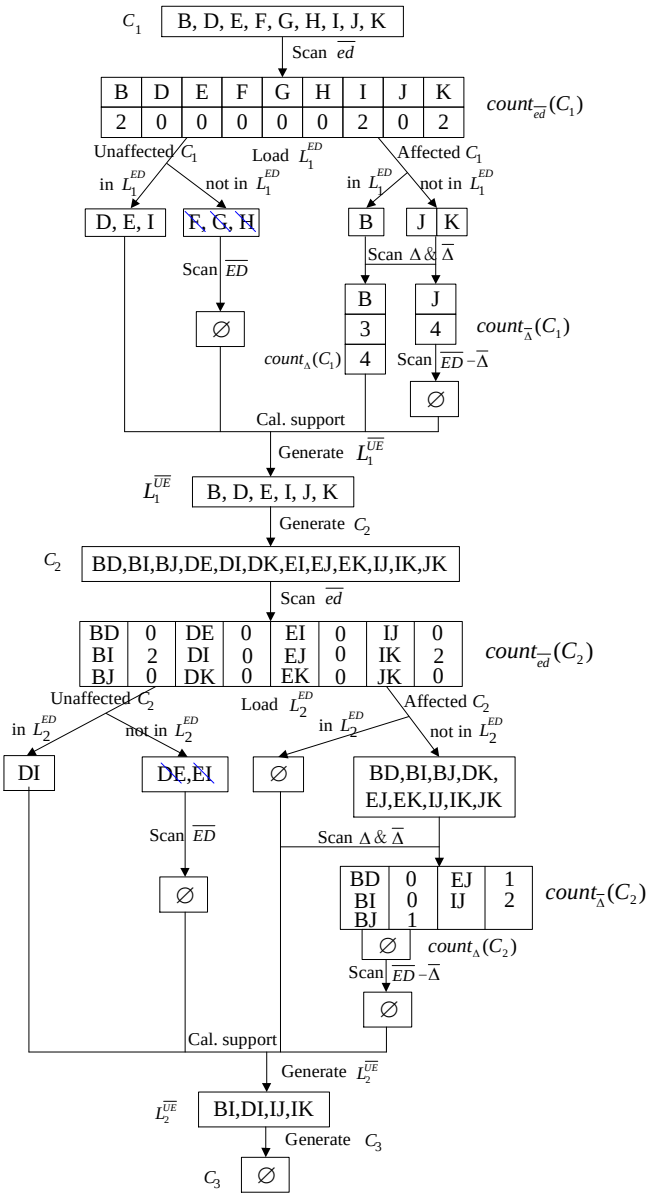


Figure 14. Illustration of algorithm IDTE2.

3.3. Situation when obsolete transactions occur

In the aforementioned mining process, the proposed algorithms, IDTE and IDTE2, are based on the assumption of constant number of transactions, i.e., $|ED| = |\overline{ED}|$. However, some transactions may become null when all the items contained are obsolete and discarded. Under this circumstance, the frequent itemsets in *ED* remain frequent in $\overline{ED}$, but the infrequent itemsets in *ED* may become frequent in $\overline{ED}$ because there are fewer transactions in $\overline{ED}$ than in *ED*, i.e., $|ED| > |\overline{ED}|$. This implies that the scanning of $\overline{ED}$ is inevitable. Fortunately, not all such infrequent itemsets in *ED* have to be counted in $\overline{ED}$. This can be clarified through the following lemmas.

Lemma 6. If an unaffected itemset $A \notin L^{ED}$ and $count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$, then $A \notin L^{\overline{UE}}$.

Proof. We can derive

$count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + |ED| \times ms < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + count_{ED}(A) < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + count_{\overline{ED}}(A) < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{UE}}(A) < |\overline{UE}| \times ms.$ ■

For example, let $|ED| = 100$, $|\overline{ED}| = 100$, $|\overline{ed}| = 15$, $count_{ED}(A) = 39$, $count_{\overline{ed}}(A) = 5$ and $ms = 40\%$. We have $sup_{\overline{UE}}(A) = count_{\overline{UE}}(A) / |\overline{UE}| = (39 + 5) / (100 + 15) = 38.3\% < ms = 40\%$, and so A is infrequent in $\overline{UE}$. Lemma 6 can verify this; $count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms \Rightarrow 5 < (15 - (100 - 100)) \times 40\% = 6$. On the contrary, if $|\overline{ED}| = 95$, we have $sup_{\overline{UE}}(A) = 44 / 110 = 40\% = ms$, and A is frequent in $\overline{UE}$.

Therefore, during the course of counting a candidate itemset that satisfies Case 2, either running by algorithm IDTE or IDTE2, we can utilize Lemma 6 to decide whether to scan $\overline{ED}$ or not. More precisely, we just need to modify Step 13 in algorithm IDTE and 14 in IDTE2, as shown below; the other part remains unchanged.

> Delete any candidate A from $C_<^-$ if $A \notin L_k^{\overline{ed}}$ and $count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$; /* Case 2 & Lemma 6 */

Lemma 7. If an affected itemset $A \notin L^{ED}$, $count_{\overline{\Delta}}(A) \le count_{\Delta}(A)$ and $count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$, then $A \notin L^{\overline{UE}}$.

Proof. Analogous to Lemma 6, it is easy to derive

$count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + |ED| \times ms < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + count_{ED}(A) < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + count_{ED}(A) + count_{\overline{\Delta}}(A) - count_{\Delta}(A) < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{ed}}(A) + count_{\overline{ED}}(A) < (|\overline{ed}| + |\overline{ED}|) \times ms$

$\Rightarrow count_{\overline{UE}}(A) < |\overline{UE}| \times ms.$ ■

Lemma 7 indeed gives the evidence for avoiding a further scanning of $\overline{ED} - \overline{\Delta}$ to deal with Case 7 in algorithm IDTE2. It suffices for this purpose by revising Step 19 of algorithm IDTE2 as:

> Delete any candidate A from $C_<^+$ if $A \notin L_k^{\overline{ed}}$, $count_{\overline{\Delta}}(A) \le count_{\Delta}(A)$ and $count_{\overline{ed}}(A) < (|\overline{ed}| - (|ED| - |\overline{ED}|)) \times ms$; /* Case 7 & Lemma 7 */

4. Performance evaluation

In order to examine the performance of IDTE and IDTE2, we conducted experiments to compare their performance with that of applying generalized association mining algorithms, including Cumulate and Stratify, to the whole updated database. The comparisons are evaluated according to certain aspects: minimum support, incremental transaction size, and evolution degree. Here, the evolution degree is measured by the fraction of items that are affected. In the implementation of each algorithm, we also adopted two different support counting strategies: one with the horizontal counting [1][2][8][9] and the other with the vertical intersection counting [10][11]. For the horizontal counting, the algorithms are denoted as Cumulate(H), Stratify(H), IDTE(H) and IDTE2(H) while for the vertical intersection counting, the algorithms are denoted as Cumulate(V), Stratify(V), IDTE(V) and IDTE2(V). All experiments were performed on an Intel Pentium-IV 2.80GHz with 2GB RAM, running on Windows 2000. Both synthetic data and real data are considered.

We choose a synthetic dataset (denoted as Synth) generated by the IBM data generator [2] and a real supermarket dataset (Foodmart for short) in the experiments. The data for Foodmart is drawn from Microsoft foodmart2000, which is a supermarket data warehouse provided in MS SQL2000. The corresponding item taxonomy consists of three levels, comprising 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). The parameter settings for both datasets are shown in Table 5. Note that in some figures, vertical scales are in logarithmic representation for better resolution.

Table 5. Default parameter settings for test datasets.

Parameter		Default value	
		Synth	Foodmart
$\|DB\|$	Number of original transactions	177,783	35,000
$\|db\|$	Number of incremental transactions	40,000	5,000
$\|t\|$	Average size of transactions	16	12
N	Number of items	231	1,717
R	Number of groups	30	47
L	Number of levels	3	3
F	Fanout	5	14

Minimum supports: We first compared the performance of these four algorithms with varying minimum supports at 40,000 incremental transactions for Synth and 5,000 incremental transactions for Foodmart with constant affected item percent 1.4% and 0.23%, respectively. The experimental results are shown in Figures 15 and 16 for Synth and Foodmart, respectively. Conforming to previous result in the literature, association mining methods implemented with vertical counting strategy are significant better than those with horizontal counting strategy [11]. It can be further observed that with the same counting strategy, for Synth, IDTE(V) performs 110% and 388% faster than Cumulate(V) and Stratify(V) at ms = 0.5%, respectively, and for Foodmart, IDTE(V) performs 23% faster than Cumulate(V) and Stratify(V) at ms = 0.05%. Among our algorithms, IDTE(V) and IDTE2(V) perform better than IDTE(H) and IDTE2(H).

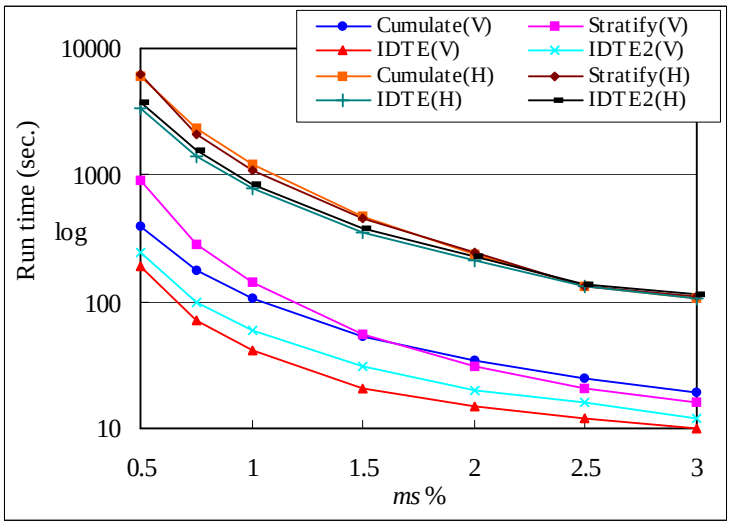


Figure 15. Performance comparison of IDTE, IDTE2, Cumulate, and Stratify for different ms over Synth.

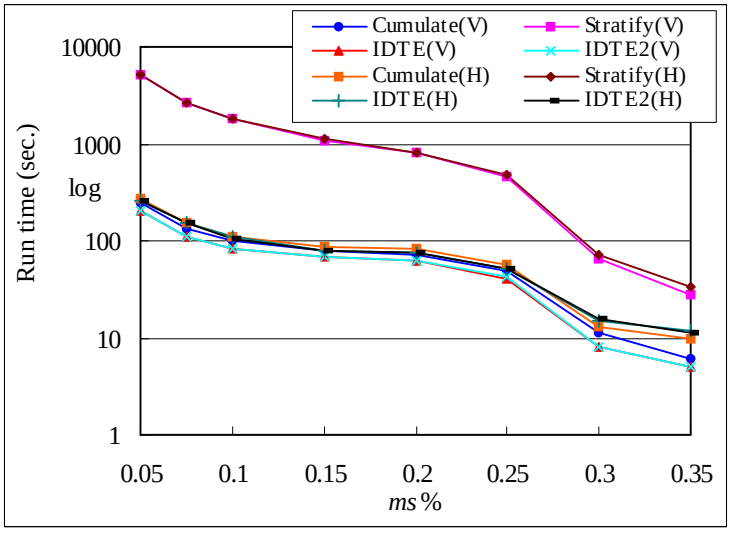


Figure 16. Performance comparison of IDTE, IDTE2, Cumulate, and Stratify for different ms over Foodmart.

Transaction sizes: We then compared the four algorithms under varying transaction sizes at *ms* = 1.0% with constant affected item percent 1.4% for Synth and at *ms* = 0.05% with affected item percent 0.23% for Foodmart; other parameters were set to default values. The results are depicted in Figures 17 and 18 for Synth and Foodmart, respectively. All algorithms exhibit linear scalability. Note that in Figure 18, for clearness of comparison we only show Cumulate and our algorithms because Stratify performed significantly poorly at low *ms* for Foodmart. It is noteworthy that the horizontal version of our algorithms performs worse than the vertical version of Cumulate and Stratify for Synth, and the gap becomes more significant as the data size increases. This is because the processing time per transaction in the horizontal version of our algorithms is greater than that in the vertical versions of Cumulate and Stratify.

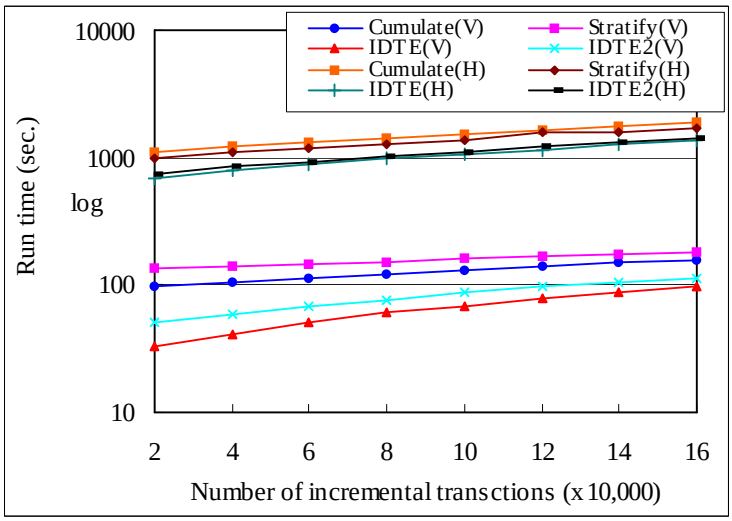


Figure 17. Performance comparison of IDTE, IDTE2, Cumulate, and Stratify for different transactions over Synth.

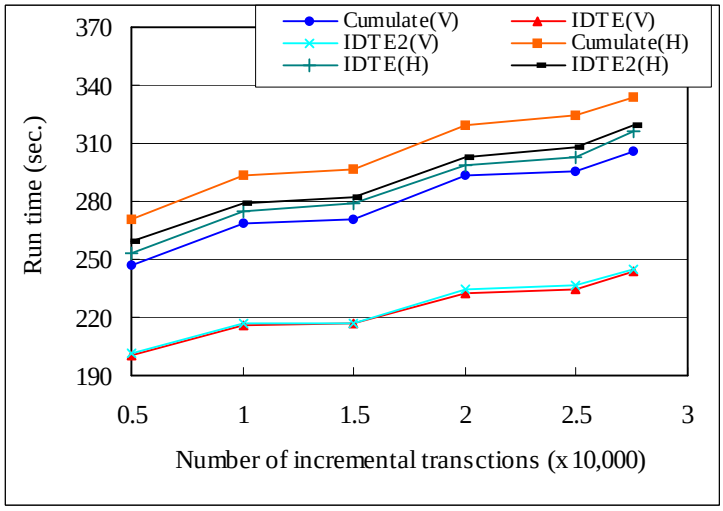


Figure 18. Performance comparison of IDTE, IDTE2, and Cumulate for different transactions over Foodmart.

Evolution degree: Finally, we compared the four algorithms under varying degrees of evolution with *ms* = 1.0% and 40,000 incremental transactions for Synth and with *ms* = 0.05% and 5,000 incremental transactions for Foodmart. The other parameters were set to default values. In this experiment the affected generalizations are rearranged randomly, and the results are depicted in Figures 19 and 20 for Synth and Foodmart, respectively. As the results show, our algorithms are greatly affected by the degree of evolution, whereas Cumulate and Stratify exhibit steady performance. In Figure 19, IDTE2(V) performs better than Cumulate(V) and Stratify(V) only under 3.5% of affected items while IDTE2(H) performs better than Cumulate(H) and Stratify(H) only under 4% of affected items. In Figure 20, IDTE2(H) performs better than Cumulate(H) only under 2.1% of affected items. We observe that IDTE2 takes longer process time than IDTE while increasing evolution degree since IDTE2 require processing both the original extended database *ED* and updated extended database $\overline{ED}$ simultaneously, and the advantage of Lemma 4 for IDTE2 disappears when the number of affected transactions increases.

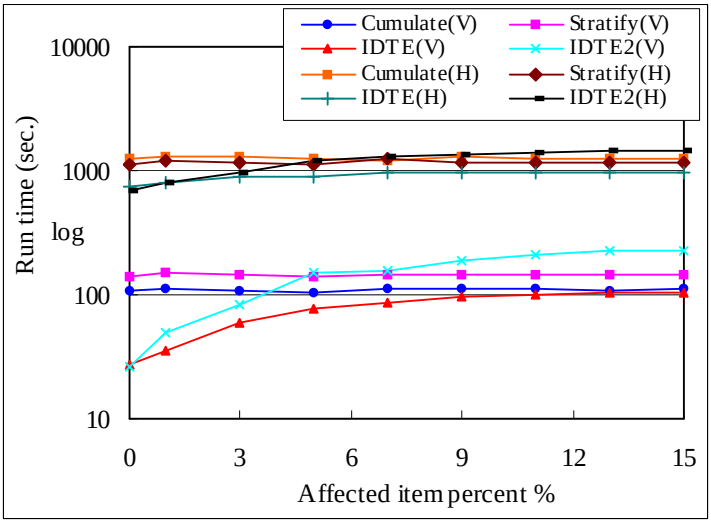


Figure 19. Performance comparison of IDTE, IDTE2, Cumulate, and Stratify for different degrees of evolution over Synth.

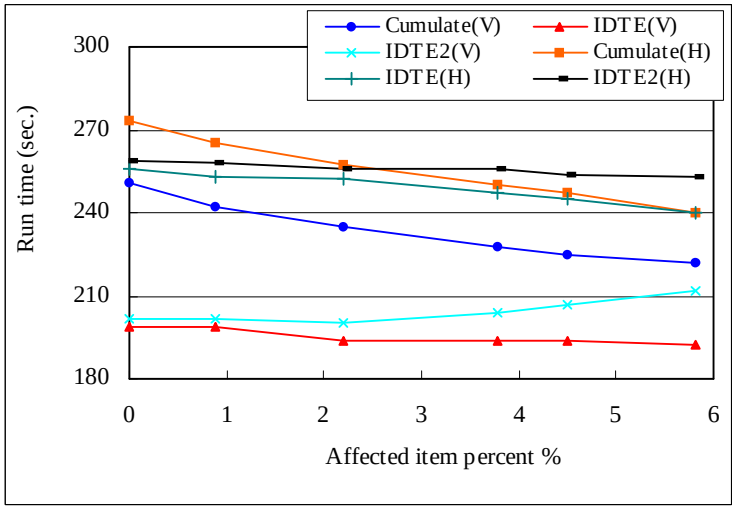


Figure 20. Performance comparison of IDTE, IDTE2 and Cumulate for different degrees of evolution over Foodmart.

In summary, we observe that IDTE(V) is superior to Cumulate(V) and Stratify(V), while IDTE(H) beats Cumulate(H) and Stratify(H) in all aspects of the evaluation. Moreover, all algorithms with vertical support counting strategy perform better than their counterparts with horizontal counting strategy.

5. Related work

5.1. Generalized association rules mining and maintenance

The problem of mining association rules in the presence of taxonomy information was initially addressed by Han et al. [3] and Srikant et al. [4], working independently. The problem was referred to as mining generalized association rules, aiming to find associations among items at any level of the taxonomy under the minimum support and minimum confidence constraints [4]. In Han et al., the problem mentioned was somewhat different from that considered in Srikant et al., because they generalized the uniform minimum support constraint to a form of assignment according to level, i.e., items at the same level received the same minimum support. The objective was to discover association level-by-level in a fixed hierarchy. That is, only associations among items on the same level were examined progressively from the top level to the bottom. Since then, several improvements or extensions have been proposed. Sriphaew and Theeramunkong [5] presented a method that exploits two types of constraints on generalized itemset relationships, called subset-superset and ancestor-descendant constraints, to speedup the mining process. In [6], Domingues and Rezende proposed an algorithm, called GART, which uses taxonomies, in the step of knowledge post-processing, to generalize and to prune uninteresting rules that may help the user to analyse the generated association rules.

The problem of updating association rules incrementally was first addressed by Cheung et al. [12]. They developed the essence of updating the discovered association rules when new transaction records are added to the database over time and proposed an algorithm called FUP (Fast UPdate). They further extended the model to incorporate the situations of deletion and modification [13]. Their approaches [12][13], however, did not consider the generalized items, and hence could not discover generalized association rules. Subsequently, a number of techniques have been proposed to improve the efficiency of incremental mining algorithms [14][15][16][17][18], although all of them were confined to mining associations among primitive items.

The maintenance issue for generalized association rules was also first studied by Cheung et al. [19], who proposed an extension of their FUP algorithm, called MLUp, to accomplish the task. In [20], Huang and Wu tackled the problem from a different point of view. Their approach used the original primitive frequent itemsets and association rules to directly generate new generalized association rules. Hong et al. [21] then considered the problem of updating generalized associations for record modification. They extended Han and Fu's approach [3] by introducing the concept of pre-large itemsets [14] to postpone the original database rescanning until a number of records have been modified. In [22], we have extended the problem of maintaining generalized associations incrementally to that incorporates non-uniform minimum support.

In summary, all previous work on mining generalized association rules required the taxonomy to be static, ignoring the fact that the taxonomy of items may change over time.

5.2. Mining association rules over data streams

Recently, due to many applications generating large volumes of data in a continuous and automated way, the problem of data stream mining has come to an emerging issue [23][24]. According to a recent survey conducted by R. Jin and G. Agrawal [25], the state-of-the-art association mining techniques over data streams can be categorized into four different models: landmark window [26][27][28][29], sliding window [30][31], damped window [32], and tilted-time window models [33]. Among them, the model of sliding window is the closet to the problem of incremental association rule mining. The fast changing and potentially infinite characteristics of stream data, however, makes the problem of mining stream data very different from that of mining traditional transaction data. The most important one is that "you can only look once", which implies that mining algorithms requiring more than one pass of dataset scan are not applicable. Besides, the memory limitation always is a main concern. All of the proposed stream mining algorithms thus have struggled to discover *approximate solutions* as accurate as possible within the constraint of only one pass of data scan and limited memory space. In this context, our method, though works for transactional data update and can discover accurate patterns, in its current form is not applicable for stream data. On the other hand, to the best of our knowledge, the taxonomy information and the

effect of taxonomy evolution, which is the main focus of our work, has not yet been addressed in any literature regarding mining association rules over data streams. The problem of mining generalized association rules over streaming data with or without taxonomy evolution thus remains an unexplored issue and deserves further investigation.

6. Conclusions

In this paper we have investigated the problem of updating generalized association rules when new transactions were inserted into the database and the taxonomy of items evolves over time. We also have presented two novel algorithms, IDTE and IDTE2, for updating generalized frequent itemsets. Empirical evaluation on synthetic data showed that the IDTE and IDTE2 algorithms are very efficient, significantly outperforming the contemporary generalized associations mining algorithms that are applied to the whole updated database.

In the future, we will extend the problem of updating generalized association rules to a more general model that adopts non-uniform minimum support to solve the problem that new introduced items usually have much lower supports. We will also apply the results to the problem of on-line discovery and maintenance of multi-dimensional association rules from a data warehouse data under evolution of taxonomy or attributes.

7. References

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

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

[3] J. Han and Y. Fu, Discovery of multiple-level association rules from large databases, *Proceedings of the 21st International Conference on Very Large Data Bases* (1995) 420-431.

[4] R. Srikant and R. Agrawal, Mining generalized association rules, *Proceedings of the 21st International Conference on Very Large Data Bases* (1995) 407-419.

[5] K. Sriphaew and T. Theeramunkong, Fast algorithms for mining generalized frequent patterns of generalized association rules, *IEICE Transaction on Information and Systems* 87(3) (2004) 761-770.

[6] M.A. Domingues and S.O. Rezende, Using taxonomies to facilitate the analysis of the association rules, *Proceedings of the Second International Workshop on Knowledge Discovery and Ontologies* (2005) 59-66.

[7] J. Han and Y. Fu, Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases, *Proceedings of AAAI'94 Workshop on Knowledge Discovery in Databases* (1994) 157-168.

[8] S. Brin, R. Motwani, J.D. Ullman and S. Tsur, Dynamic itemset counting and implication rules for market-basket data, *SIGMOD Record* (26) (1997) 255-264.

[9] J.S. Park, M.S. Chen and P.S. Yu, An effective hash-based algorithm for mining association rules, *Proceedings of ACM-SIGMOD International Conference on Management of Data* (1995) 175-186.

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

[11] M.J. Zaki, Scalable algorithms for association mining, *IEEE Transactions on Knowledge and Data Engineering* 12(2) (2000) 372-390.

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

[13] D.W. Cheung, S.D. Lee and B. Kao, A general incremental technique for maintaining discovered association rules, *Proceedings of the 5th International Conference on Database Systems for Advanced Applications* (1997) 185-194.

[14] T.P. Hong, C.Y. Wang and Y.H. Tao, A new incremental data mining algorithm using pre-large itemsets, *Intelligent Data Analysis* 5(2) (2001) 111-129.

[15] K.K. Ng and W. Lam, Updating of association rules dynamically, *Proceedings of 1999 International Symposium Database Applications in Non-Traditional Environments* (2000) 84-91.

[16] N.L. Sarda and N.V. Srinivas, An adaptive algorithm for incremental mining of association rules, *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications* (1998) 240-245.

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

[18] A. Veloso, W. Meira Jr., M. Carvalho, S. Parthasarathy and M.J. Zaki, Parallel, incremental and interactive mining for frequent itemsets in evolving databases, *Proceedings of the 6th International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining* (2003).

[19] D.W. Cheung, V.T. Ng and B.W. Tam, Maintenance of discovered knowledge: a case in multi-level association rules, *Proceedings of 1996 International Conference on Knowledge Discovery and Data Mining* (1996) 307-310.

[20] Y.F. Huang and C.M. Wu, Mining generalized association rules using pruning techniques, *Proceedings of 2002 IEEE International Conference on Data Mining* (2002) 227-234.

[21] T.P. Hong, T.J. Huang and C.S. Chang, Maintenance of multiple-level association rules for record modification, *Proceedings of 2004 IEEE International Conference on Systems, Man & Cybernetics* (2004) 3140-3145.

[22] M.C. Tseng and W.Y. Lin, Maintenance of generalized association rules with multiple minimum supports, *Intelligent Data Analysis* 8(4) (2004) 417-436

[23] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, Models and issues in data stream systems, *Proceedings of the 2002 ACM Symposium on Principles of Database Systems* (2002).

[24] L. Golab and M.T. Ozsu, Issues in data stream management, *SIGMOD Record* 32(2) (2003) 5-14.

[25] R. Jin and G. Agrawal, Frequent pattern mining in data streams. In: C.C. Aggarwal (ed.), *Data Streams: Models and Algorithms* (Springer, 2007).

[26] G.S. Manku and R. Motwani, Approximate frequency counts over data streams, *Proceedings of the 28th International Conference on Very Large Data Bases* (2002) 346-357.

[27] H.F. Li, S.Y. Lee, and M.K. Shan, An efficient algorithm for mining frequent itemsets over the entire history of data streams, *Proceedings of the 1st International Workshop on Knowledge Discovery in Data Streams* (2004).

[28] R. Jin and G. Agrawal, An algorithm for in-core frequent itemset mining on streaming data, *Proceedings of the IEEE International Conference on Data Mining* (2005) 210-217.

[29] J.X. Yu, Z. Chong, H. Lu, Z. Zhang, and A. Zhou, A false negative approach to mining frequent itemsets from high speed transactional data streams, *Information Sciences* 176(14) (2006) 1986-2015.

[30] M. Datar, A. Gionis, P. Indyk, and R. Motwani, Maintaining stream statistics over sliding windows, *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms* (2002) 635-644.

[31] Y. Chi, H. Wang, P.S. Yu, and R.R. Muntz, Moment: Maintaining closed frequent itemsets over a stream sliding window, *Proceedings of the IEEE International Conference on Data Mining* (2004) 59-66.

[32] J.H. Chang and W.S. Lee, Finding recent frequent itemsets adaptively over online data streams, *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003) 487-492.

[33] C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu, Mining frequent patterns in data streams at multiple time granularities. In: H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), *Next Generation Data Mining*, AAAI/MIT (2003).