Updating Generalized Association Rules with Evolving Taxonomies

Wen-Yang Lin
Dept. of Comp. Sci. & Info. Eng.
National University of Kaohsiung
Kaohsiung 811, Taiwan
wylin@nuk.edu.tw

Ming-Cheng Tseng
Institute of Information Engineering
I-Shou University
Kaohsiung 840, Taiwan
clark.tseng@msa.hinet.net

Abstract*-Mining generalized association rules between items in the presence of taxonomy has been recognized as an important model in data mining. Earlier work on mining generalized association rules confined the taxonomy to be static. However, the taxonomy of items cannot be kept unchanged all the time. Some items will be sifted from one hierarchy tree to another for more suitable classification or be abandoned from the taxonomy if they will not be produced any more; new born items will also be added into the taxonomy. Under these circumstances, how to update the discovered generalized association rules effectively is a crucial task. In this paper, we examine this problem and propose a novel algorithm, called Taxo_UP, to update the discovered frequent itemsets. Empirical evaluation shows that the proposed algorithm is very effective and has good linear scale-up characteristic.*

Keywords: Data mining, generalized association rules, frequent itemsets, evolving 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]. 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 contain items in Y, and the probability, measured as the fraction of transactions containing X also containing 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 *minsup*, and minimum confidence, called *minconf*, respectively. The problem of mining association rules is to discover all association rules that satisfy *minsup* and *minconf*.

In many applications, there are taxonomies (hierarchies), explicitly or implicitly, over the items. It may be more useful to find associations at different levels of the taxonomy than only at the primitive concept level [6][10]. For example, consider Figure 1, the taxonomies of items from which the previous association rule derived.

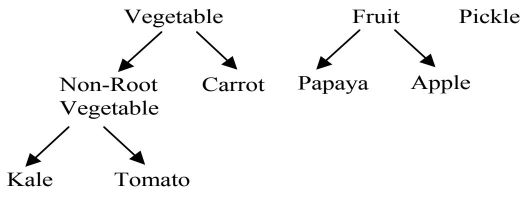


Figure 1. An example of taxonomies *T*.

It is likely to happen that the association rule

Carrot ⇒ Apple (*Sup.* = 30%, *Conf.* = 60%)

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

Vegetable ⇒ Fruit.

Earlier work on mining generalized association rules confined the taxonomies to be static. However, the taxonomy of items may change as time passes [5]. Some items will be sifted from one classification tree to another. For example, tomato would be reclassified from a vegetable to a fruit. Some trees of the taxonomies will be merged together or be split into smaller trees if the items on the trees cannot meet the demands in a new classification. Items will be abandoned if those items do not be produced any more, and newborn items are added. Under these circumstances, how to update the discovered generalized association rules effectively becomes a critical task.

In this paper, we introduce the problem of updating the discovered generalized association rules under evolving taxonomy. We will give a formal problem description and clarify the situations of taxonomy updates in Section 2.

A simple way for dealing with this problem is to adopt the mining approach to re-scan the whole database from scratch to reflect the most recent associations. This approach, however, has the following disadvantages:

(1) It is not cost-effective because the discovered frequent itemsets are not reused.
(2) It is not acceptable in general since the process of generating frequent itemsets is very time-consuming.

To be more realistic and cost-effective, it is better to perform the association mining algorithms to generate the initial association rules, and when update to the taxonomy occurs, apply an updating method to re-build the discovered rules. The challenge thus falls into developing an efficient updating algorithm to facilitate the whole mining process. This problem is nontrivial because updates to the taxonomy not only can reshape the concept hierarchy and the form of generalized items, but also may invalidate some of the discovered association rules, turn previous weak rules into strong ones, and generate import new, undiscovered rules.

We propose an algorithm called Taxo_UP (Taxonomy Update) for mining the generalized frequent itemsets, which is capable of effectively reducing the number of candidate sets and database re-scanning, and so can update the generalized association rules efficiently. Detail description of the Taxo_UP algorithm will be given in Section 3.

In Section 4, we evaluate the performance of the proposed Taxo_UP with two leading generalized associations mining algorithms, Cumulate and Stratify from [10], on synthetic data. A brief description of previous related work is presented in Section 5. Finally, we summarize our work and future investigation in Section 6.

2. Update of Generalized Association Rules

2.1. Problem description

Let $I = \{i_1, i_2, \ldots, i_p\}$ be a set of items and $DB = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where each transaction $t_i = \langle tid, A\rangle$ has a unique identifier *tid* and a set of items A ($A \subseteq I$). Assume that a set of taxonomies of items, T, is available and is denoted as a set of hierarchies (trees) on $I \cup J$, where $J = \{j_1, j_2, \ldots, j_p\}$ represents the set of generalized items derived from I. 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.

Given a user specified *minsup* and *minconf*, the problem of mining generalized association rules is to discover all generalized association rules whose supports and confidence are larger than the specified thresholds. This problem is reduced to the problem of finding all frequent itemsets for a given minimum support [6][10].

Let L be, after an initial discovery of all the generalized association rules in *DB*, the set of all frequent itemsets with respect to *minsup*. As time passes, some updated activities may occur to the taxonomies due to some reasons [5]. We denote the updated taxonomies as T'. The problem of updating discovered generalized association rules in *DB* is to find the set of frequent itemsets L' with respect to the refined taxonomies T'.

2.2. Types of taxonomy updates

In this subsection, we will describe different situations for taxonomy evolution, and clarify the essence of frequent itemsets update for each type of taxonomy evolutions.

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 pruned from the taxonomy.
(3) Item rename: Items are renamed for some reasons, such as error correction, product promotion, etc.
(4) Item reclassification: Items are classified into different categories.

Note that hereafter the term "item" refers to a *primitive* or a *generalized* item. Each type of evolutions is further explained in the following.

Type 1: Item insertion. The strategies to handle this type of update operation are different in whether the inserted item is primitive or generalized.

When the new inserted item is primitive, we cannot process this item until there is an incremental database update, because the new item does not appear in the original set of frequent itemsets.

On the other hand, if the new item represents a generalization, then the insertion itself also has no effect on the discovered associations until the new generalization incurs some item reclassification.

Figure 2 shows an example of this type of taxonomy evolution, where a new item "J" is inserted as (a) a primitive item or (b) a generalized item. Note that in Figure 2b item "E" is reclassified to the generalization represented by "J".

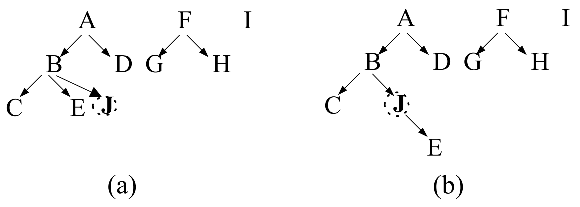


Figure 2. An example of taxonomy evolution caused by item insertion. The inserted item "J" is: (a) primitive; (b) generalized.

Type 2: Item deletion. This case is similar to the insertion case. There is nothing to do with the deletion of primitive items if no transaction update to the original database, and the removal of a generalization may also lead to items reclassification.

Figure 3 shows an example of this type of taxonomy evolution.

Type 3: Item rename. When items are renamed, we do not have to process the database. Instead, we just replace the frequent itemsets with new names and so the association rules.

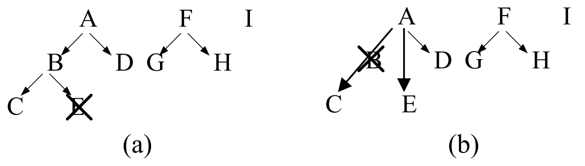


Figure 3. An example of taxonomy evolution caused by item deletion: (a) The primitive item "E" is deleted; (b) The generalized item "B" is deleted, and items "C" and "E" are reclassified to "A".

Type 4: Item reclassification. Among the four types of taxonomy updates this is the most profound operation. Once an item, primitive or generalized, is reclassified into another category, all of its ancestor (generalized items) in the old and the new taxonomies are affected. In other words, the supports of these affected generalized items have to be updated and so do the frequent itemsets containing any one of the affected generalized items. For example, in Figure 4, the two shifted items E and G will change the support counts of generalized items A, B, and F, and also affect the support counts of itemsets containing A, B, or F.

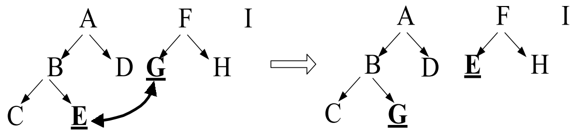


Figure 4. An example of taxonomy evolution caused by item reclassification.

In this paper, we assume that there is no transaction update to the original database, and so we only have to consider, according to the above discussions, the taxonomy evolution caused by insertion or deletion of generalized items, and reclassification of primitive or generalized items.

3. The Proposed Taxo_UP Algorithm

3.1. Algorithm description

Let *ED* denote the extended version of *DB* by adding, in taxonomies *T*, the ancestors of each primitive item to each transaction, while *UE* be another extension of *DB* by adding generalized items in the updated taxonomies *T'*. A straight-forward way to find updated generalized frequent itemsets would be to run any of the algorithms, such as Cumulate and Stratify [10], for finding generalized frequent itemsets on the updated extended transactions *UE*. This simple way, however, ignores the fact that many discovered frequent itemsets would not be affected by the taxonomy evolution.

Lemma 1. The supports of all primitive items do not change with respect to a taxonomy evolution.

Thus, the proposed Taxo_UP algorithm follows a simple guideline: Identify the affected and unaffected generalizations first, and so the affected transactions. Then utilize them to lessen the work of support counting of itemsets as could as possible.

An *affecting item* is a primitive item whose ancestor set changes with respect to a taxonomy evolution. An item is an *affected item* if its descendent primitive item set changes with respect to a taxonomy evolution. A transaction is called an *affected transaction* if it contains at least one of affecting items. For a given itemset *A*, we say *A* is an *affected itemset* if it contains at least one of affected items. There are four different cases in dealing with the support counting of *A*.

(1) If *A* is an unaffected itemset and is frequent in *ED*, then it is also frequent in *UE*.
(2) If *A* is an unaffected itemset and is infrequent in *ED*, then it is also infrequent in *UE*.
(3) If *A* is an affected itemset and is frequent in *ED*, then it may be frequent or infrequent in *UE*.
(4) If *A* is an affected itemset and is infrequent in *ED*, then it may be frequent or infrequent in *UE*.

Note that only cases 3 and 4 need further database scan to determine the support count of *A*. Indeed, for case 3, we only have to scan the affected transactions in *ED* and *UE*. Then calculate its support to decide whether it is frequent or not.

As for case 4, the following lemma provides an effective pruning strategy to reduce the number of candidate itemsets and so can avoid unnecessary database scan.

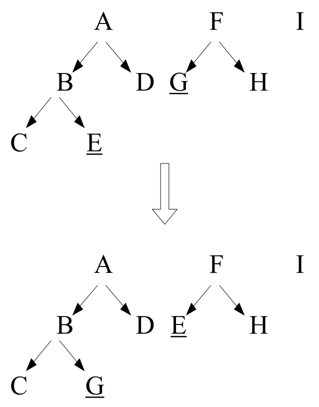
Lemma 2. If an affected itemset $A \notin L$ (frequent itemsets in *ED*) and $\delta'_A - \delta_A \le 0$, then $A \notin L'$ (frequent itemsets in *UE*), where Δ is the set of transactions in *ED* affected by taxonomy update, Δ' the set of affected transactions in *UE*, and δ_A and δ'_A are support counts of *A* in transactions Δ and Δ', respectively.

Proof. If $A \notin L$, then $\sigma_A < |ED| \times minsup$, where σ_A denotes the support count of *A* in *ED*. Note that $|UE| = |ED| = |ED| - |\Delta| + |\Delta'|$ due to $|\Delta| = |\Delta'|$. Hence, $\sigma'_A = \sigma_A + (\delta'_A - \delta_A) < |ED| \times minsup = |UE| \times minsup$, where σ'_A is the support count of *A* in *UE*. Thus, $A \notin L'$. ■

Thus in case 4, we scan the affected transactions in *ED* and *UE* to count the appearances of *A*. If the support count of *A* in *UE*'s affected transactions is greater than that in *ED*, then we have to scan the rest of *UE* to decide whether *A* is frequent or not.

An overview of the Taxo_UP algorithm is presented below.

Algorithm: Taxo_UP
Inputs: (1) *DB*: the database; (2) *ms*: the minimum support setting; (3) *T*: the old item taxonomy; (4) T': the new item taxonomy; (5) $L = \bigcup_k L_k$: the set of old



Original Extended Database (*ED*)

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

Updated Extended Database (*UE*)

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

Figure 5. An example of mining generalized association rules caused by item reclassified.

frequent itemsets

Output: $L' = \bigcup_k L'_k$: the set of new frequent itemsets.

Method:

1. Load L_1; let C_1 be the set of items in T'.
2. Divide the set of candidate 1-itemsets C_1 into two parts: one X consists of unaffected items in L_1, and the other Y contains affected items.
3. Add generalized items in T and T' into the original database *DB* to form *ED* and *UE* respectively.
4. Compute the count of each 1-itemset A in Y over the affected transactions of *ED* (Δ) and *UE* (Δ'); let the values be δ_A and δ'_A, respectively.
5. For each 1-itemset A that is in L_1 and the affected set Y, i.e., $A \in Y \cap L_1$, calculate $\sigma'_A = \sigma_A - \delta_A + \delta'_A$.
6. For any candidate $A \notin L_1$ and $(\delta'_A - \delta_A) > 0$, count A over the transactions in *UE* - Δ' and add the counts to δ'_A.
7. Create L'_1 by combining X and those itemsets which are frequent in Y.
8. Generate candidates C_2 from L'_1.
9. Repeat Steps 1-8 for new candidates C_k until no frequent k-itemsets L'_k created.

3.2. An example

Consider Figure 5. Let *minsup* = 25% (3 transactions). The set of frequent itemsets L includes: A(7), D(6), F(5), G(3), I(8), AF(4), AG(3), AI(6), DF(3), DG(3), DI(6), FI(3), GI(3), AFI(3), AGI(3), DFI(3), and DGI(3), where the support counts of itemsets are shown within parentheses.

The Taxo_UP algorithm first divides all items in C_1 into two sets: one consists of unaffected items D, G, and I in L_1, and the other contains affected generalized items A, B and F, where A and F are frequent in L_1, while B is not. Since items D, G, and I are primitive frequent items and do not change their supports in *ED* and *UE*, we do not need to process them; we only have to process generalized items A, B and F. Next, transactions 1, 2 and 3 in *ED* and *UE* are scanned since these transactions are affected by exchanging items G and E. We then subtract the support counts of items A and F in *ED*'s affected transactions from and add their counts in *UE*'s affected transactions to their original support counts. For example, we have $\sigma'_{\{F\}} = \sigma_{\{F\}} - \delta_{\{F\}} + \delta'_{\{F\}} = 5 - 3 + 1 = 3$. Item B is not frequent in *ED* and may become frequent because $\delta'_{\{B\}} - \delta_{\{B\}} = 3 - 1 = 2 > 0$ according to Lemma 2. Therefore, we scan transaction 4 to counting item B and add 1 to the count from *UE*. That is, $\sigma'_{\{B\}} = \delta'_{\{B\}} + 1 = 3 + 1 = 4$. After comparing the supports of A, B, and F to *minsup*, the new frequent 1-itemsets L'_1 are A, B, D, F, G, and I.

Next, we use L'_1 to generate candidate 2-itemsets C_2, obtaining AF, AI, BD, BF, BI, DF, DG, DI, FG, FI, and GI. However, only AF, AI, BD, BF, BI, DF, FG, and FI undergo support counting, because the others are composed of primitive items. Note that the original frequent 2-itemset AG is deleted in *UE* due to the existence of item-ancestor relationship. After that, the procedure of generating frequent 2-itemsets is the same as that for generating L'_1. The new frequent 2-itemsets L'_2 are AI, BD, BI, DG, DI, and GI. Finally, we use the same approach to generat L'_3, obtaining BDI and DGI. Note that transaction 2 has the same generalized items after the taxonomy evolution; however, we still require processing this transaction in Step 4 of the proposed algorithm. If we do not process this transaction, 2-itemset FG will not become frequent when *minsup* = 10%.

The whole process of running this example using Taxo_UP is illustrated in Figure 6.

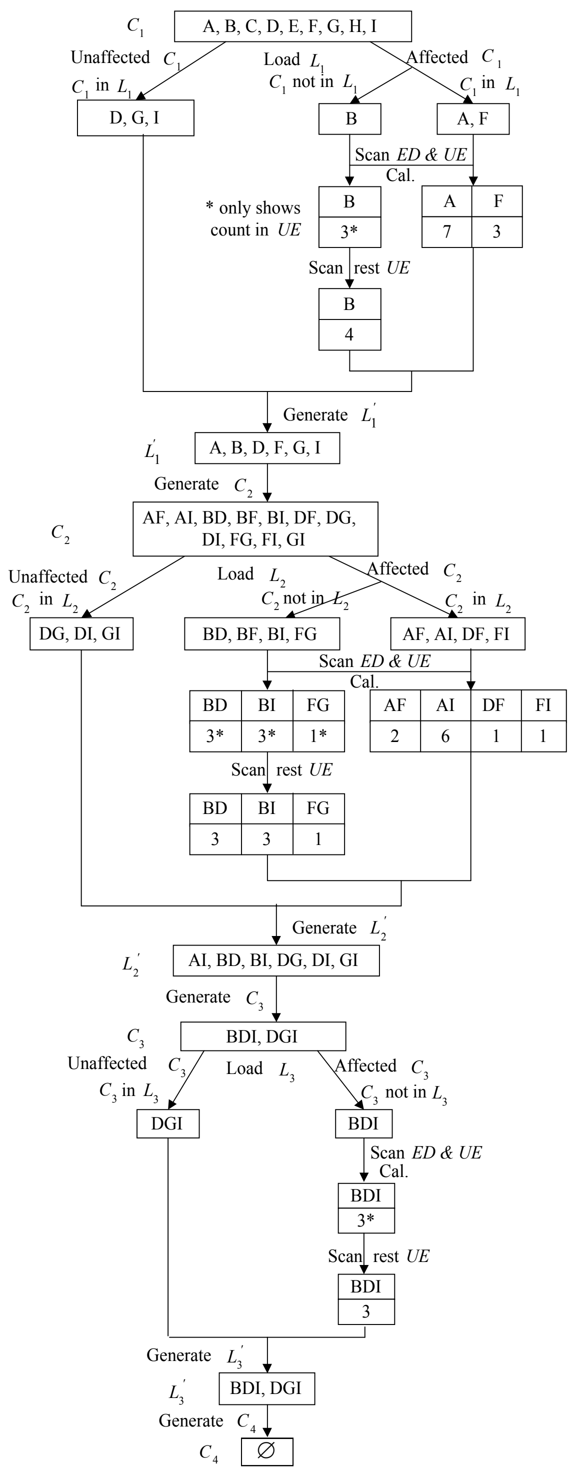


Figure 6. Illustration of algorithm Taxo_UP.

4. Performance Evaluation

In order to examine the performance of Taxo_UP, we conducted experiments to compare its performance with that of Cumulate and Stratify, using the synthetic dataset generated by the IBM data generator [1]. The parameter settings are shown in Table 1. Two items in the taxonomies were randomly chosen to exchange their positions. All experiments were performed on an Intel AMD-800 with 512MB RAM, running on Windows 2000.

Table 1. Default parameter settings for synthetic data generation.

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

We first compared the performance of these algorithms with varying minimum supports. As shown in Figure 7, Taxo_UP performs significantly better than Stratify and Cumulate; the improvement ranges from 3 to 6 times and increases as *minsup* decreases.

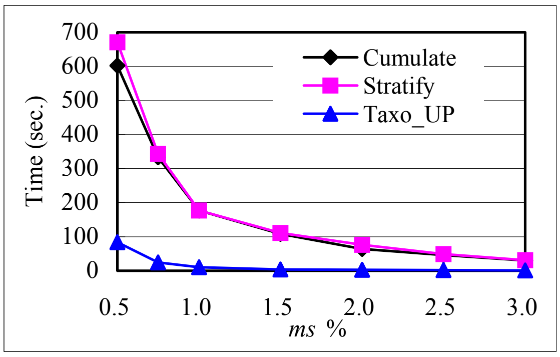


Figure 7. Performance comparison of Taxo_UP, Cumulate, and Stratify for different *minsups*.

We then compared the three algorithms under varying transaction sizes. The result is depicted in Figure 8. As shown in the figure, Taxo_UP performs significantly better than Cumulate and Stratify. Algorithm Cumulate performs a little better than Stratify. The results also demonstrate that Taxo_UP has better scalability than Cumulate and Stratify.

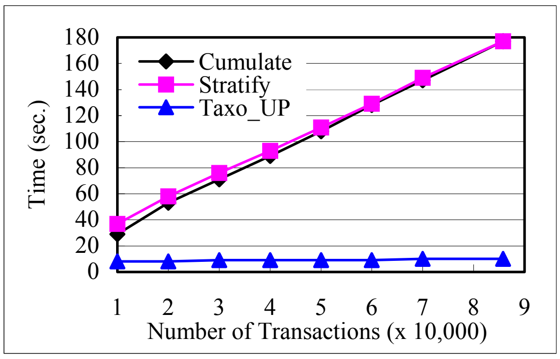


Figure 8. Performance comparison of Taxo_UP, Cumulate, and Stratify for different transactions at *minsup* = 1.0%.

5. Related Work

The problem of mining association rules in the presence of taxonomy information was addressed first in [6] and [10], independently. In [10], the problem is named as mining generalized association rules, which aims to find associations among items at any level of the taxonomy under the minimum support and minimum confidence constraints. In [6], the problem mentioned is somewhat different from that considered in . They generalized the uniform minimum support constraint to a form of level-wise assignment, i.e., items at the same level receive the same minimum support. The objective was mining associations level-by-level in a fixed hierarchy. That is, only associations among items at the same level are examined progressively from the top level to the bottom.

The problem of updating association rules incrementally was first addressed by Cheung et al. [2]. They coined the essence of updating the discovered association rules when new transaction records are added into the database over time and proposed an algorithm called FUP (Fast UPdate). They further examined the maintenance of multi-level association rules [3], and extended the model to incorporate the situations of deletion and modification [4]. Their approaches [3][4], however, did not consider the generalized items, and hence could not discover generalized association rules.

Since then, a number of techniques have been proposed to improve the efficiency of incremental mining algorithm [7][8][9][11]. But all of them were confined to mining associations between primitive items. The problem of maintaining generalized associations incrementally has been recently investigated in [12], wherein the model of generalized associations was extended to that with non-uniform minimum support.

To sum up, all related previous works addressed in part the aspects discussed in this paper; no work, to our knowledge, has considered the issues of evolving taxonomies.

6. Conclusions

We have investigated in this paper the problem of updating generalized association rules under evolving taxonomy. We presented an algorithm, Taxo_UP, for updating generalized frequent itemsets. Empirical evaluation showed that the Taxo_UP algorithm is very effective and has good linear scale-up characteristic.

In the future, we will extend the problem of updating generalized association rule to a more general model that deals with an incremental database update and fuzzy taxonomic structure. We will also investigate the problem of on-line discovery and maintenance of multi-dimensional association rules from data warehouse data under taxonomy or attribute evolution.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 487-499.

[2] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. "Maintenance of discovered association rules in large databases: An incremental update technique," *Proc. 1996 Int. Conf. Data Engineering*, 1996, pp.106-114.

[3] D.W. Cheung, V.T. Ng, B.W. Tam, "Maintenance of discovered knowledge: a case in multi-level association rules," *Proc. 1996 Int. Conf. Knowledge Discovery and Data Mining*, 1996, pp. 307-310.

[4] D.W. Cheung, S.D. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," *Proc. DASFAA'97*, 1997, pp. 185-194.

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

[6] J. Han and Y. Fu, "Discovery of multiple-level association rules from large databases," *Proc. 21st Int. Conf. Very Large Data Bases*, 1995, pp. 420-431.

[7] T.P. Hong, C.Y. Wang, Y.H. Tao, "Incremental data mining based on two support thresholds," *Proc. 4 Int. Conf. Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, 2000, pp.436-439.

[8] K.K. Ng and W. Lam, "Updating of association rules dynamically," *Proc. 1999 Int. Symp. Database Applications in Non-Traditional Environments,* 2000, pp. 84-91.

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

[10] R. Srikant and R. Agrawal, "Mining generalized association rules," *Proc. 21st Int. Conf. Very Large Data Bases*, 1995, pp. 407-419.

[11] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, "An efficient algorithm for the incremental updation of association rules in large databases," *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining,* 1997.

[12] M.C. Tseng and W.Y. Lin, "Maintenance of generalized association rules with multiple minimum supports," *Intelligent Data Analysis*, in print.