

# A Decremental Algorithm for Maintaining Frequent Itemsets in Dynamic Databases\*

Shichao Zhang<sup>1</sup>, Xindong Wu<sup>2</sup>, Jilian Zhang<sup>3</sup>, and Chengqi Zhang<sup>1</sup>

<sup>1</sup> Faculty of Information Technology, University of Technology Sydney, Australia

<sup>2</sup> Department of Computer Science, University of Vermont, USA

<sup>3</sup> Department of Computer Science, Guangxi Normal University, China  
{zhangsc, chengqi}@it.uts.edu.au; xwu@cs.uvm.edu; zhangjilian@yeah.net

**Abstract.** Data mining and machine learning must confront the problem of pattern maintenance because data updating is a fundamental operation in data management. Most existing data-mining algorithms assume that the database is static, and a database update requires rediscovering all the patterns by scanning the entire old and new data. While there are many efficient mining techniques for data additions to databases, in this paper, we propose a decremental algorithm for pattern discovery when data is being deleted from databases. We conduct extensive experiments for evaluating this approach, and illustrate that the proposed algorithm can well model and capture useful interactions within data when the data is decreasing.

## 1 Introduction

The dynamics of databases can be represented in two aspects: (1) content updates over time and (2) incremental size changes. When some transactions of a database are deleted or modified, the content of the database has been updated. This database is referred to as an *updated database* and mining with the updated database is referred to as *decremental mining*. When some new transactions are inserted or appended into a database, the size of the database has been changed. This database is referred to as an *incremental database* and mining with the incremental database is referred to as *incremental mining*. This paper investigates the issue of mining updated databases<sup>1</sup>.

When a database is updated on a regular basis, running a data-mining program all over again each time when there is an update might produce significant computation and I/O loads. Hence, there is a need for data-mining algorithms to perform frequent pattern maintenance on incrementally updated databases without having to run the entire mining algorithm again [PZOD99]. This leads to many efficient min-

---

\* This work is partially supported by large grants from the Australian Research Council (DP0449535 and DP0559536), a China NSFC major research program (60496321), and a China NSFC grant (60463003).

<sup>1</sup> For simplicity, this paper deals with deletions only, because a modification can be implemented as a deletion followed by an insertion.

ing techniques for data additions to databases, such as the FUP algorithm [CHNW96], ISM [PZOD99], negative-border based incremental updating [TBAR97], incremental induction [U94] and the weighting model [ZZY03].

Unfortunately, no research efforts have been reported on pattern discovery when data is deleted from databases. Indeed, *delete* is one of the most frequently used operations in many DBMS systems, such as IBM DB2, MS SQL SERVER, and ORACLE. Usually, these DBMS systems use log files to record the committed changes in order to maintain the database consistency. Therefore we can easily obtain the data that is deleted from the original database by using the *delete* operation.

To solve the decrement problem, one can simply re-run an association rule mining algorithm on the remaining database. However, this approach is very inefficient, without making use of the previous computation that has been performed. In this paper, we propose an algorithm **DUA** (*Decrement Updating Algorithm*) for pattern discovery in dynamic databases when data is deleted from a given database. Experiments show that **DUA** is 2 to 13 times faster than re-running the Apriori algorithm on the remaining database. The accuracy of **DUA**, namely, the ratio of the frequent itemsets found by **DUA** in the remaining database over the itemsets found by re-running Apriori, is more than 99%.

The rest of the paper is organized as follows. We provide the problem description in Section 2. In Section 3, the DUA algorithm is described in detail. Experimental results are presented in Section 4. Finally, conclusions of our study are given in Section 5.

## 2 Problem Statement

### 2.1 Association rule mining

Let  $I = \{i_1, i_2, \dots, i_N\}$  be a set of  $N$  distinct literals called *items*, and  $DB$  be a set of variable length transactions over  $I$ , where transaction  $T$  is a set of items such that  $T \subseteq I$ . A transaction has an associated unique identifier called *TID*. An *itemset*  $X$  is a set of items, i.e., a subset of  $I$ . The number of items in an itemset  $X$  is the length of the itemset.  $X$  is called a *k-itemset* if the length of  $X$  is  $k$ . A transaction  $T$  contains  $X$  if and only if  $X \subseteq T$ . An association rule is an implication of the form  $X \rightarrow Y$ , where  $X, Y \subseteq I, X \cap Y = \emptyset$ .  $X$  is the *antecedent* of the rule, and  $Y$  is the *consequent* of the rule. The support of a rule  $X \rightarrow Y$ , denoted as  $supp(X \cup Y)$ , is  $s$  if  $s\%$  transactions in  $DB$  contain  $X$ . and the confidence of rule  $X \rightarrow Y$ , denoted as  $conf(X \rightarrow Y)$ , is  $c$  if  $c\%$  transactions in  $DB$  that contain  $X$  also contain  $Y$ . That is,  $conf(X \rightarrow Y) = supp(X \cup Y) / supp(X)$ .

The problem of mining association rules from database  $DB$  is to find out all the association rules whose support and confidence are greater than or equal to the minimum support (*minsupp*) and the minimum confidence (*minconf*) specified by the user respectively. Usually, an itemset  $X$  is called frequent, if  $supp(X) \geq minsupp$ , and  $X$  is called infrequent if  $supp(X) < minsupp$ . The first step of association rule mining is to generate *frequent* itemsets using an algorithm like Apriori [AR94] or the FP-

Tree [HPY00]. The second step then generates association rules based on the discovered *frequent* itemsets. This second step is straightforward, so the main problem of mining association rules is to find out all frequent itemsets in *DB*.

## 2.2 Updated Databases

The update operations include deletions and modifications on databases. Consider a transaction database  $TD = \{\{A, B\}; \{A, C\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$  where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The update operation on TD can basically be

**Case-1.** Deleting transactions from the database TD. For example, after deleting transaction  $\{A, C\}$  from TD, the updated database is TD1 as

$$TD1 = \{\{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

**Case-2.** Deleting specified attributes from all transactions in the database TD. For example, after deleting B in TD, the updated database is TD2 as

$$TD2 = \{\{A\}; \{A, C\}; \{A, C\}; \{C\}; \{A, D\}\}$$

**Case-3.** Updating a specified attribute with another attribute in all transactions. For example, after updating the attribute C to E in TD, the updated database is TD3 as

$$TD3 = \{\{A, B\}; \{A, E\}; \{A, B, E\}; \{B, E\}; \{A, B, D\}\}$$

**Case-4.** Modifying a transaction in the database TD. For example, after modifying the transaction  $\{A, C\}$  to  $\{A, C, D\}$  for TD, the updated database is TD4 as

$$TD4 = \{\{A, B\}; \{A, C, D\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

Mining updated databases generates a significant challenge: the maintenance of their patterns. To capture the changes of data, for each time of updating a database, we can possibly re-mine the updated database, but this would be a time-consuming procedure. In particular, when a database to be mined is very large and the changed content of each updating transaction is relatively small, re-mining the database is not an intelligent strategy. Our strategy for mining in updated databases is incremental discovery. To simplify the description, this paper focuses on the above Case-1 and Case-2.

## 2.3 Maintenance of association rules

Let *DB* be the original transaction database, *db* be a dataset randomly deleted from *DB*, and *DB-db* be the remaining database.  $|DB|$ ,  $|db|$ , and  $|DB-db|$  denote the size of *DB*, *db*, and *DB-db*, respectively. Let  $S_0$  be the minimum support specified by the user,  $L$ ,  $L'$ ,  $L''$  be the set of frequent itemsets in *DB*, *db*, and *DB-db* respectively. Assume that the frequent itemsets *L* and the support of each itemset in *L* are available in advance. After a period of time, some useless data is deleted from *DB*, forming the deleted database *db*. Suppose there is an itemset *X* in *DB*, and we know that

$X$  could be frequent, infrequent or absent in  $db$ . Also suppose the support of  $X$  in  $DB-db$  is  $X_{supp}$ , with respect to the same minimum support  $S_0$ , and  $X$  is expected to be frequent only if  $X_{supp} \geq S_0$ . So, a frequent itemset  $X$  in  $DB$  may no longer be frequent again in  $DB-db$  after some data is deleted from  $DB$ . Similarly, an infrequent itemset  $X$  can become frequent in  $DB-db$ .

There are two maintenance tasks for association rules: i) evaluating the approximate upper and lower support bounds between those itemsets in  $DB$  that are most likely to change their frequentness in the remaining database  $DB-db$ , and ii) finding out all the frequent itemsets in  $DB-db$ .

### 3 A Decremental Algorithm

This section presents our DUA algorithm. In the DUA algorithm, we take into account (1) the changes of frequent and infrequent itemsets in a database when some data is subtracted from the original database; and (2) the approximation range of the itemset support in  $DB$  between those itemsets that have a chance to change their frequentness in  $DB-db$ .

#### Algorithm DUA

**Input:**  $DB$ : the original database with size  $|DB|$ ;  $db$ : the deleted dataset from  $DB$  with size  $|db|$ ;  $L$ : the set of frequent itemsets in  $DB$ ;  $S_0$ : the minimum support specified by the user;

**Output:**  $L''$ : the set of frequent itemsets in  $DB-db$ ;

1. **compute**  $S'$ ; /\*  $S'$  is explained later in this section and also in Section 4.1. \*/
2. **mine**  $db$  with minimum support  $S'$  and **put** the frequent itemsets into  $L'$ ;
3. **for** each itemset in  $L$  **do**
4.     **for** each itemset in  $L'$  **do**
5.         **identify** the frequent itemsets in  $DB-db$ , **eliminate** them from  $L'$  and **store** to  $L''$ ;
6. randomly **sample** a set of transactions from  $DB$ , denoted as  $db'$ ;
7. **mine**  $db'$  with the threshold  $S_0$ , and obtain frequent itemsets  $L'''$ ;
8. **eliminate** the itemsets from  $L'''$  that occur in  $L$  and  $L'$ ;
9. **for** each remaining itemset in  $L'$  and  $L'''$
10.     **scan**  $DB$  to **obtain** their support in  $DB$ ;
11.     **identify** the frequent itemsets in  $DB-db$  and **append** them to  $L''$ ;
12. **end for**;
13. **return**  $L''$ ;

#### 14. end procedure;

From the above description, after eliminating the itemsets from  $L'$  and  $L'''$  that are infrequent or frequent in  $DB-db$ , it is obvious that the remaining itemsets in  $L'$  are all infrequent in  $DB$ , while the remaining itemsets in  $L'''$  are infrequent in  $DB$  and they do not occur in  $db$ .

Based on the previous work on the theoretical analysis for determining a lower threshold [Toivonen96], in DUA, we set  $S'$  as  $0.62 \times S_0$ . In our experiments in Section 4, we demonstrate that  $S' = 0.62 \times S_0$  is an appropriate support threshold that satisfies the requirements for both efficiency and accuracy when mining  $db$ .

## 4 Experiments

The above sections have discussed the problem of pattern maintenance when deleting data from a large database, and have proposed an algorithm named **DUA** to deal with this problem. This section presents experiments we have conducted on a DELL Workstation PWS650 with 2G main memory and 2.6G CPU. The operating system is WINDOWS 2000.

### 4.1 Experiments for determining $S'$ when mining $db$

We employ a traditional algorithm *Apriori* [AR94] with a lower minimum support threshold  $S'$  to get some infrequent itemsets in  $db$ . In order to choose an appropriate threshold experimentally, we take into account the theoretical analysis for obtaining a lower threshold [Toivonen96] and have conducted some experiments on synthetic databases.

We first generated a database T10.I4.D100K as  $DB$  and also a  $db$  of 10% transactions randomly deleted from  $DB$ . We set the minimum support  $S_0$  to 1.5% and 1% respectively, and use a threshold of  $S'$  from  $0.3 \times S_0$  to  $0.9 \times S_0$  to mine  $db$ . Those itemsets in  $DB$ , whose supports are greater than the lower bound  $S_{lower}$  (the lower support constraint) and less than  $S_0$ , are most likely to become frequent in  $DB-db$ . We denote these infrequent itemsets in  $db$  as *IIS*. The itemsets whose supports are greater than  $S_0$  and less than  $S_{upper}$  (the upper support constraint) are called *unstable frequent itemsets*, denoted as *UFIS*. Similarly, those itemsets whose supports are less than  $S_0$  and greater than  $S_{lower}$  are called *unstable infrequent itemsets (UIIS)*. Table 1 shows the execution time, total itemsets in  $db$ , *IIS* and *UIIS* when using a different minimum support  $S'$  to mine  $db$ .

It is possible that most of the transactions in  $db$  contain some identical items. For example, the user deleted all the transactions that contain either item "5" or item "8" from the database. Thus, the transactions in  $db$  either contain item "5" or item "8". In Table 2, we use another database T10.I4.D100K as  $DB$ , and construct  $db$  by deleting all transactions that contain item "120" from  $DB$ . The minimum support  $S_0$  is 1.5% and 1% respectively.

**Table 1** Using different  $S'$  for mining  $db$  (with random deletions)

| $S'$             | $S_0$ | Execute ime(s) | Total itemsets | IIS  | UIIS |
|------------------|-------|----------------|----------------|------|------|
| $0.9 \times S_0$ | 1.5%  | 195.406        | 266            | 26   | 7    |
|                  | 1%    | 434.265        | 387            | 36   | 13   |
| $0.8 \times S_0$ | 1.5%  | 283.891        | 294            | 54   | 13   |
|                  | 1%    | 525.282        | 442            | 91   | 24   |
| $0.7 \times S_0$ | 1.5%  | 341.906        | 339            | 99   | 13   |
|                  | 1%    | 617.453        | 482            | 131  | 28   |
| $0.6 \times S_0$ | 1.5%  | 429.875        | 387            | 147  | 13   |
|                  | 1%    | 745.625        | 538            | 187  | 28   |
| $0.5 \times S_0$ | 1.5%  | 578.937        | 465            | 225  | 13   |
|                  | 1%    | 887.015        | 628            | 277  | 28   |
| $0.4 \times S_0$ | 1.5%  | 746.719        | 538            | 298  | 13   |
|                  | 1%    | 1090.937       | 1065           | 714  | 28   |
| $0.3 \times S_0$ | 1.5%  | 980.485        | 727            | 487  | 13   |
|                  | 1%    | 1369.718       | 2658           | 2307 | 28   |

As explained before, all the *UIIS* in *DB* must be examined in order to find out whether they will become frequent in *DB-db*. So a lower threshold  $S'$  is used to find out as many *UIIS* as possible in *db*. From these tables, we can see that the computational costs are less with fewer *UIIS* found when  $S'$  is set to  $0.9 \times S_0$ ,  $0.8 \times S_0$  and  $0.7 \times S_0$  respectively. More *UIIS* can be found when  $S'$  is below  $0.6 \times S_0$ , but more computational costs are also required. The appropriate tradeoff point for  $S'$  is in the interval  $[0.6 \times S_0, 0.7 \times S_0]$ . We use  $0.62 \times S_0$  as the minimum support threshold  $S'$  for mining *db*.

## 4.2 Experiments on algorithm DUA

We have conducted two sets of experiments to study the performance of DUA. The first set of experiments is done when *db* is generated by randomly deleting transactions from *DB*. The second set of experiments is done when *db* is generated by deleting transactions that contain specified items from *DB*.

### 4.2.1 Experiments with *db* formed by random deletions

We construct *db* by randomly deleting some transactions from *DB* (T10.I4.D100K), and then use **DUA** on *db* to study its performance against the Apriori algorithm. We define the accuracy of **DUA** as the ratio of the number of frequent itemsets found by **DUA** against the number of frequent itemsets found by Apriori in *DB-db*.

**Table 2** Using different  $S'$  for mining  $db$  (with specified deletions)

| $S'$             | $S_0$ | Execute time(s) | Total itemsets | IIS   | UIIS |
|------------------|-------|-----------------|----------------|-------|------|
| $0.9 \times S_0$ | 1.5%  | 24.985          | 1631           | 52    | 0    |
|                  | 1%    | 53.047          | 1955           | 106   | 0    |
| $0.8 \times S_0$ | 1.5%  | 32.469          | 1733           | 154   | 0    |
|                  | 1%    | 77.453          | 3025           | 1176  | 0    |
| $0.7 \times S_0$ | 1.5%  | 43.937          | 1849           | 270   | 0    |
|                  | 1%    | 91.156          | 1651           | 1802  | 1    |
| $0.6 \times S_0$ | 1.5%  | 53.047          | 1955           | 376   | 1    |
|                  | 1%    | 115.25          | 5397           | 3818  | 2    |
| $0.5 \times S_0$ | 1.5%  | 75.188          | 3025           | 1446  | 2    |
|                  | 1%    | 166.546         | 8867           | 7018  | 2    |
| $0.4 \times S_0$ | 1.5%  | 115.25          | 5397           | 3818  | 2    |
|                  | 1%    | 379.578         | 18103          | 16254 | 2    |
| $0.3 \times S_0$ | 1.5%  | 380.594         | 18103          | 16254 | 2    |
|                  | 1%    | 872.64          | 37301          | 35452 | 2    |

We set  $|db|=1\%|DB|$ ,  $5\%|DB|$ ,  $10\%|DB|$ ,  $20\%|DB|$  and  $30\%|DB|$  respectively. The following figures present the performance ratio against Apriori and the accuracy of **DUA**.

Figures 1.1 and 1.2 reveal that **DUA** performs 6 to 8 times faster than Apriori for several databases of 100K transactions, and the accuracy of **DUA** is 99.2% to 100%. A trend also can be seen that with the decrease of the minimum support from 1.5% to 0.5%, the accuracy is dropping. The reason is that there are many itemsets with lower supports in a database in general. When the minimum support threshold  $S_0$  is set very low, many itemsets whose supports are slightly below  $S_0$ , namely *UIIS*, may become frequent in  $DB-db$ . While only a fraction of the *UIIS* can be found in  $db$  and the sampling database  $db'$  when using **DUA**. This means that there are some frequent itemsets in  $DB-db$  that cannot be found using **DUA**. So the accuracy of **DUA** drops when a lower minimum support threshold is given.

With the increase of the size of  $db$ , the average time ratio against Apriori slows down. In Figures 2.1 and 2.2, when  $|db|=30\%|DB|$ , **DUA** is only 1.2 times faster than Apriori. It is clear that Apriori should be applied to mining  $DB-db$  instead of using **DUA**, when the amount of the data deleted from  $DB$  is greater than 30% of the total data in  $DB$ , i.e.,  $|db|>30\%|DB|$ .

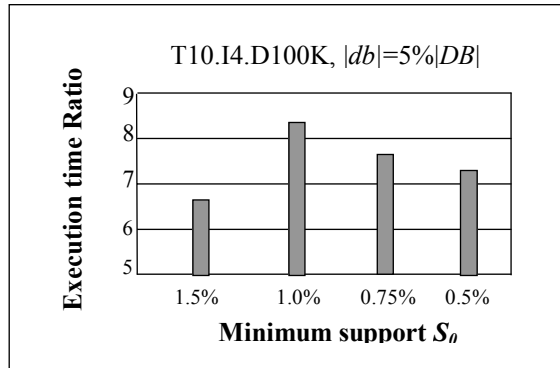


Figure 1.1: Performance ratio between DUA and Apriori

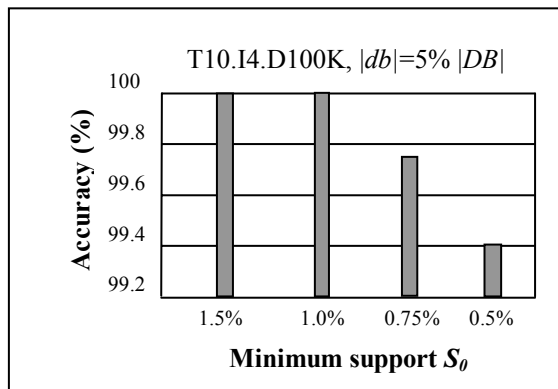


Figure 1.2: Accuracy of DUA

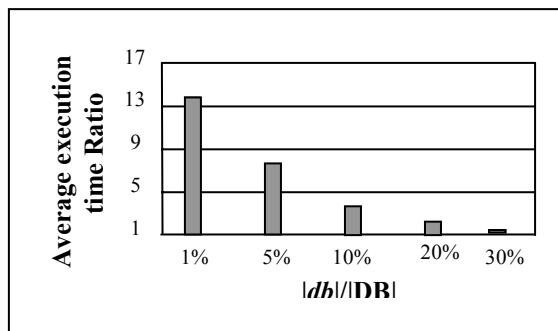


Figure 2.1: Execution time against decrement size

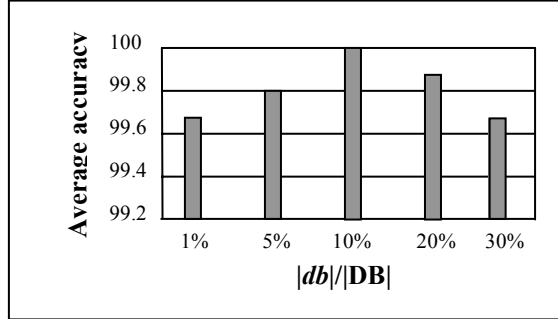


Figure 2.2: Average accuracy with different decrement size

#### 4.2.2 Experiments with $db$ formed by specified deletions

In the following experiments, we construct  $db$  by deleting transactions that contain a specified item from  $DB$ . Figures 3.1 and 3.2 present the performance and accuracy of DUA on  $db$  that is formed by deleting transactions containing the specified item "100", which has a moderate support in  $DB$ . In this case, DUA is still 6 to 10 times faster than re-running Apriori on  $DB-db$ .

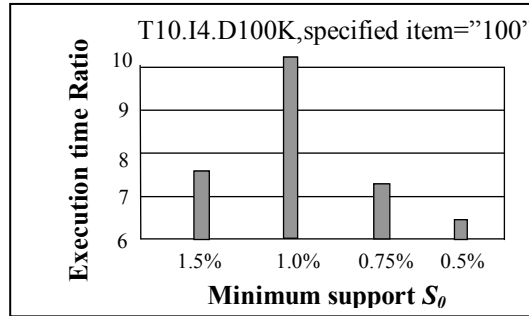


Figure 3.1: Performance ratio between DUA and Apriori

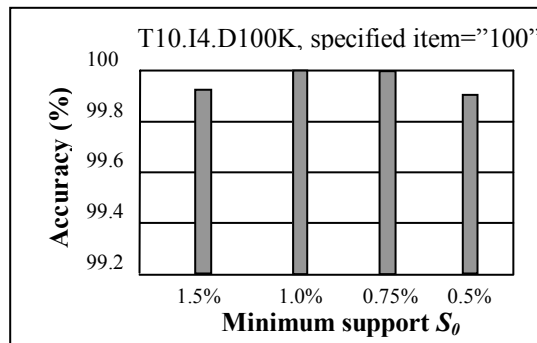


Figure 3.2: Accuracy of DUA

## 5 Conclusion and future work

Pattern maintenance is a challenging task in data mining and machine learning. In this paper, we have studied the problems of pattern maintenance when data is subtracted from a large database, and have proposed an efficient decremental algorithm to deal with the problem. Our method checks the itemsets found in the subtracted dataset and the itemsets found in the original database in order to determine which itemsets are frequent in the remaining database and which are not frequent any more. Experiments have shown that our **DUA** algorithm is faster than using Apriori on the remaining database, with a high accuracy.

In practice, the operations of data insertions and deletions are interleaving, which make the problem of pattern maintenance in large databases more complicated. Also, a theoretical analysis of the error bounds for **DUA** is very important. These are items for our future work.

## References

- [AR94] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules. *VLDB94*, 1994: 487–499.
- [CHNW96] D. Cheung, J. Han, V. Ng, and C. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *ICDE'96*, 1996: 106–114.
- [HPY00] J. Han, J. Pei, and Y. Yin, Mining Frequent Patterns without Candidate Generation. *SIGMOD00*, 2000: 1–12.
- [PZOD99] S. Parthasarathy, M. Zaki, M. Ogihara, and S. Dwarkadas, Incremental and Interactive Sequence Mining. *CIKM'99*, 1999: 251–258.
- [TBAR97] S. Thomas, S. Bodagala, K. Alsabti and S. Ranka, An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. *KDD'97*, 1997: 263–266.
- [Toivonen96] H. Toivonen, Sampling Large Databases for Association Rules. *VLDB96*, 1996: 134–145.
- [U94] P. Utgoff, An Improved Algorithm for Incremental Induction of Decision Trees. *ICML'94*, 1994: 318–325.
- [ZZY03] S. Zhang, C. Zhang, and X. Yan, Post-mining: Maintenance of Association Rules by Weighting. *Information Systems*, Vol. 28, 7(2003): 691–707.