Ameliorated Algorithm to Maintain Discovered Frequent Itemsets

Du, Xiaoping
Department of Intelligent Systems, Faculty of Information Science and Electrical Engineering, Kyushu University

Makinouchi, Akifumi
Department of Intelligent Systems, Faculty of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.15017/1515711

出版情報：九州大学大学院システム情報科学紀要. 6 (1), pp.19-24, 2001-03-26. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第6巻 第1号 平成13年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.6, No.1, March 2001

Ameliorated Algorithm to Maintain Discovered Frequent Itemsets

Xiaoping DU* and Akifumi MAKINOUCHI*

(Received December 15, 2000)

Abstract: It is an important task in data mining to maintain discovered frequent itemsets for association rule mining. Because most time-consuming operation for mining association rules is to find the frequent itemsets from the transaction database. And the database is always updated. However, the algorithms proposed so far for the maintenance of discovered frequent itemsets can only perform with a minimum support threshold which is the same as that of previous mining. If the new result derived with such minimum support is unsatisfactory to a user, the maintaining process may fail. In this paper we propose a new algorithm to maintain discovered frequent itemsets. Our algorithm allows users to adjust the minimum support of maintaining process. And it can be performed repeatedly with a different minimum support until the satisfying results are obtained. We prove the efficiency of our algorithm by experiments with several synthetic transaction databases.

Keywords: Frequent itemset, Association rule, Data mining, Database, Algorithm

1. Introduction

Mining of association rules from transaction databases was well known as an important problem in data mining. It consists of two subproblems, including mining association rules from a transaction database and maintaining the discovered association rules when the transaction database was updated.

Mining association rules consists of two steps[1]. The first is to find all *frequent itemsets*, which frequently occur in a significant number of transactions with respect to a threshold, called *minimum support*. The second is, with respect to another threshold, called *minimum confidence*, to generate all the association rules using the obtained frequent itemsets. The most time-consuming operation of the process is to find the frequent itemsets. Therefore, the mining association rules and the maintenance of discovered association rules focus on finding frequent itemsets. In order to find out the frequent itemsets from a database, many interesting and efficient algorithms were proposed[1),2),10),7),6)]. For the maintenance of discovered frequent itemsets, there have been several algorithms[3),4),5),8),9)].

FUP (Fast Update)[4] and FUP_2[5] are two efficient algorithms for maintaining the discovered frequent itemsets. FUP is used in the case that some new transaction data is added to a database. And FUP_2 is a generalization of the FUP algorithm, including the cases of insertion into, deletion from and modification of the original database. Only in the insertion case, FUP_2 is equivalent to FUP. The algorithm DELI[8),9)] (Difference Estimation for Large Itemsets)†1 was proposed to decide how often or when the maintenance algorithm should be applied. And the algorithm MLUp (Multi-Level association rules Update)[3] is used to solve the update problem for multi-level frequent itemsets.

All of the maintenance algorithms mentioned above are only performed with a minimum support which is the same as that in previous mining process †2. However, if the new results are not satisfactory with that minimum support, users have to rerun the mining algorithms from scratch (is not the maintenance algorithms) with a different minimum support repeatedly until the satisfying results are obtained. And the mining process has to be performed from scratch in each time. In that way, the maintenance cost may be massive.

To the best of our knowledge, very little work has been done on the maintenance of discovered frequent itemsets that allow a user, according to the already existed results, to adjust the minimum support. In this paper, we propose an algorithm AFUP (Ameliorated FUP) to maintain the discovered frequent itemsets when some new transaction data is added to a database. AFUP not only can be performed with the same minimum support but also

* Department of Intelligent System

†1 In some paper, the frequent itemset is also called the large itemset.

†2 In MLUp algorithm, though the minimum supports at different level may not be equal, they must be the same as that of in the mining process at the same level.

with a different (bigger or smaller) minimum support efficiently. AFUP cuts down the cost of the maintaining process by using two kinds of useful information. One is the already existed frequent itemsets and another is meta-results during the maintaining process.

2. Problem Description

2.1 Mining association rules

Let $\mathcal{I}=\{i_1, i_2, ...i_n\}$ be a set of *items*. Let D be a transaction database, a set of transactions. Each transaction T is a subset of the *itemset* $\mathcal{I}$. A transaction T *contains* an itemset X if and only if $X \subseteq T$. The *support-count* of X in D, X_s^D is the number of transactions that contain X in D, and the *support* of X in D is the percentage of transactions in D which contains X.

An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subseteq \mathcal{I}$, $Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has support s in D if $s\%$ of transactions in D contain both X and Y. The rule $X \Rightarrow Y$ holds in D with confidence c if $c\%$ of transactions in D that contain X also contain Y.

For a pair of a minimum support (*minsup*) and a minimum confidence (*minconf*) given by a user, the task of mining association rules is to find all the association rules whose support is no less than the minsup and whose confidence is also no less than the minconf. An itemset is *frequent* in D if its support is no less than the minsup.

Here, we assume that the satisfactory frequent itemsets in D are obtained with s_{min}. And we called the mining process in D with s_{min} a *final mining process.*

2.2 Maintaining process

After some new transaction data Δ (called an increment database) is added into the original database D, all support-counts of itemsets may change. We use UD to denote the updated database $D \cup \Delta$. The maintenance of discovered frequent itemsets is to find out the satisfactory frequent itemsets to a user in UD. There are, potentially, three selections for a new minimum support s_1 in the *first maintaining process*, based on the user's estimation for the increment database. Those selections are $s_1 = s_{min}$, $s_1 > s_{min}$ and $s_1 < s_{min}$.

If the obtained frequent itemsets in the first maintaining process are unsatisfactory, the minimum support must be changed again and the maintaining process is repeated (called *repeated maintaining process*) until the obtained frequent itemsets are satisfactory to the user. We use s_n to represent the minimum support which is used in the n-th maintaining process. For $s_n (n > 1)$ there are only two potential selections relative to s_{n-1}, $s_n > s_{n-1}$ and $s_n < s_{n-1}$. The selection of $s_n = s_{n-1}$ is insignificant because such maintaining process is the equivalent of the last maintaining process with s_{n-1}.

We call an itemset containing k items a *k-itemset.* Let F_k denote the set of all the frequent k-itemsets in D with s_{min}, and F_k^n denote the set of frequent k-itemsets in UD with s_n.

3. Related Works

In this section, in order to help motivate our approach formulations, we briefly introduce the algorithm FUP[4], which is related to our work.

FUP is a fast algorithm for maintenance of discovered frequent itemsets when some transaction data Δ is added to the original database D. FUP finds new frequent itemsets in UD with s_{min}. It includes several passes over the databases, and it scans Δ and D individually in each pass. FUP reuses the already existed frequent itemsets to reduce the number of candidate itemsets when D is scanned.

In pass k, FUP creates the set of candidate k-itemsets C_k using apriori-gen(F'_{k-1})[2], where F'_{k-1} is a set of frequent $(k-1)$-itemsets in UD obtained in previous pass. Then it scans Δ to compute the support-counts of candidates in C_k. After Δ is scanned, the support-counts of all itemsets belonging to $C_k \cap F_k$ in UD are obtained. Here F_k is the set of already existed frequent k-itemsets in D. For the rest of candidates which belong to $C_k - F_k$, FUP omits useless itemsets from them according to following strategy.

A k-itemset X belonging to $C_k - F_k$ can become a frequent itemset in UD only if $X_s^\Delta \geq |\Delta| \times s_{min}\%$, here X_s^Δ is the support-count of X in Δ, and $|\Delta|$ is the number of transactions in Δ.

After the candidate itemsets in C_k are reduced further, FUP scans D to update the support-counts for all the remaining candidates in $C_k - F_k$. Finally, a set F'_k, which contains all the frequent k-itemsets in UD, is obtained from F_k and C_k. The maintaining process can't terminates until F'_k becomes empty.

4. AFUP Algorithm

FUP algorithm can only be performed with s_{min} no matter whether the obtained result is satisfying or not. We propose a new algorithm AFUP given in **Fig.** 1 to overcome this shortcoming. AFUP

includes two parts. The part 1 is used in the first maintaining process, and the part 2 is used in the repeated maintaining processes.

1) n: n-th maintaining process
2) s_n: the *minsup* used in the n-th maintaining process
3) D: the original transaction database
4) Δ: the increment transaction database
5) if(n=1)then do begin/*Part 1, refer to Section 4.1*/
6) F_1^1={all frequent 1-itemsets in $D \cup \Delta$ with s_1};
7) for($k = 2; F_{k-1}^1 \neq \emptyset; k++$) do begin
8) C_k =apriori-gen(F_{k-1}^1);
9) decomposes C_k to P_k, R_k and Q_k;
10) prunes useless itemsets in P_k, R_k and Q_k;
11) scans Δ to compute X_s^Δ in P_k, R_k and Q_k;
12) prunes useless itemsets in R_k and Q_k;
13) scans D to compute X_s^D in R_k and Q_k;
14) F_k^1={$X | X \in P_k \cup R_k \cup Q_k$, $X_s \geq |UD| \times s_1\%$};
/* $X_s = X_s^D + X_s^\Delta$ and $|UD| = |D| + |\Delta|$ */
15) end
16) else /*Part 2: $n > 1$, refer to Section 4.2*/
17) if($s_n < s_{n-1}$) then do begin
18) F_1^n={all frequent 1-itemsets in $D \cup \Delta$ with s_n};
19) for($k = 2; F_{k-1}^n \neq \emptyset; k++$) do begin
20) C_k =apriori-gen(F_{k-1}^n);
21) decomposes C_k to P_k, R_k and Q_k;
22) prunes useless itemsets in R_k and Q_k;
23) scans Δ to compute X_s^Δ in R_k and Q_k;
24) prunes useless itemsets in R_k and Q_k;
25) scans D to compute X_s^D in R_k and Q_k;
26) F_k^n={$X | X \in P_k \cup R_k \cup Q_k$, $X_s \geq |UD| \times s_n\%$};
27) end
28) end
29) end

Fig.1 AFUP Algorithm.

The part 1 of AFUP algorithm is performed only once, but the part 2 can be performed repeatedly until the satisfying frequent itemsets are obtained. In each part, the framework of AFUP is similar to that of FUP algorithm. AFUP includes several passes over Δ and D. In pass k, it finds out the frequent k-itemsets in UD. It stop when no frequent k-itemsets are found. AFUP algorithm is differ from FUP in that the different minimum support can be handled. Our goal is to reduce the number of itemsets which the support-counts must be computed to improve the performance for finding frequent itemsets in UD. We explain the first and the repeated maintaining process individually in follows.

4.1 The first maintaining process

In the first maintaining process, there are three alternatives to select the minimum support s_1, i.e., $s_1 = s_{min}$, $s_1 > s_{min}$ and $s_1 < s_{min}$, We call them *the same, the enlargement* and *the lessening threshold cases*, respectively. We explain the lessening threshold case in following, and then represent the other two cases.

The lessening threshold case In the pass 1, we scans Δ and D to determine all the new frequent 1-itemsets F_1^1 in UD according to s_1. A subsequent pass, we say pass k, consists of seven steps (lines 8-14 in **Fig.** 1).

In step 1, we generate C_k by apriro-gen(F_{k-1}^1)[2] which the same as FUP. In step 2, we decompose C_k into three disjoint subsets P_k, R_k and Q_k according to the three cases that the itemsets in C_k **must be**, **possibly are**, or **could never be** the frequent itemsets relative to s_1 in D (not in UD), respectively.

Due to $s_1 < s_{min}$, the itemsets in F_k, which derived in the final mining process with s_{min}, are also frequent itemsets in D relative to s_1. Thus, we assign the itemsets in $C_k \cap F_k$ to P_k.

We adopt a upper bound of the support-count (upper bound in brief) for a itemset in $C_k - P_k$ to decide whether or not the itemset is the frequent itemsets in D relative to s_1. Let b_X^D and b_X^Δ denote the upper bound of the itemset X in D and Δ, respectively. For any itemset X in $C_k - P_k$, we have obtained the support-counts of its $(k-1)$-subsets in Δ and D, Y_s^Δ and Y_s^D in the preovious pass. Note that, the fact that an itemset appears in a transaction will necessarily leads all of its subsets to also appear in this transaction. So, $b_X^D \leq \min\{Y_s^D | Y \subset X\}$ and $b_X^\Delta \leq = \min\{Y_s^\Delta | Y \subset X\}$. According to the definition of the frequent itemset, if $b_X^D < |D| \times s_1\%$ then X in $C_k - P_k$ could never be frequent in D relative to s_1. We assign such itemsets to Q_k. Finally, we obtain $R_k = C_k - P_k - Q_k$.

Example 1 We suppose $|D|$=1000, $|\Delta|$=200, s_{min}=10% and s_1=9%, F_2^1={bc : 109 : 1 : 110, be : 103 : 6 : 109, bf : 99 : 11 : 110, ce : 102 : 10 : 112, cf : 101 : 15 : 116, ef : 89 : 20 : 109} and F_3={bce : 101 : × : ×}. For convenience, we use bc:109:1:110 as shorthand for the itemset $\{b, c\}$ and its support-count in D, Δ and UD, respectively. Because F_3 is the set of frequent 3-itemsets in D, so the support-counts in Δ and UD don't exist and we note them as ×.

According to the assumption above, in the step 1 we get C_3={bce, bcf, bef} and P_k={bce} since $bce \in C_3 \cap F_3$. The upper bound of $\{b, e, f\}$ in D is min{103,99,89}=89<1000× 9%=90, therefore Q_k={bef}. Final, R_k={bcf}.

After C_k was decomposed into P_k, R_k and Q_k, we

reduce the useless itemsets from P_k, R_k and Q_k according to the following methods, which are based on the definition of frequent itemset and their upper bounds (line 10).

If $X \in P_k$ and $X_s^D + b_X^\Delta < |UD| \times s_1\%$, X will be deleted from P_k, which X_s^D is derived from F_k. If $X \in R_k$ and $b_X^D + b_X^\Delta < |UD| \times s_1\%$, X will be deleted from R_k. If $X \in Q_k$ and $b_X^\Delta < |\Delta| \times s_1\%$, we can delete it from Q_k since $b_X^D < |D| \times s_1\%$ is ture for all $X \in Q_k$.

Example 2 Counting the example above, b, c, e is deleted from P_k since $\{b,c,e\}_s^D + b_{\{b,c,e\}}^\Delta = 101 + \min\{1,6,10\} = 102 < 1200 \times 9\% = 108$. Since the upper bounds of $\{b,c,f\}$ in R_k in D and Δ are $\min\{109,99,101\} + \min\{1,11,15\} = 100 < 108$, $\{b,c,f\}$ is deleted from R_k. Similarly, $\{b,e,f\}$ is also pruned from Q_k because the upper bound of $\{b,e,f\}$ in Q_k in Δ is $\min\{6,11,20\} = 6 < 20 \times 9\% = 18$.

Thus, before the database is scanned in pass k, we have deleted some useless candidate itemsets. Then, we scan Δ to compute the support-counts of the itemsets in P_k, R_k and Q_k in Δ (line 11). In the end of scan, the itemsets in P_k have obtained the support-counts in UD. We, then, use the support-counts of all the itemsets X of R_k and Q_k in Δ, X_s^Δ instead of b_X^Δ in the methods of useless itemsets above, to reduce the useless itemsets in R_k and Q_k again. Since $X_s^\Delta \leq b_X^\Delta$, many itemsets in R_k and Q_k are pruned in this step (line 12).

For the remaining itemsets in R_k and Q_k, we compute their support-counts in D by D scan (line 13). Finally, in step 7 (line 14), the set of frequent k-itemsets in UD with s_1, F_k^1 is obtained from P_k, R_k and Q_k according to $(X_s^D + X_s^\Delta) \geq |UD| \times s_1\%$.

The process doesn't stop until the obtained F_k^1 becomes an empty set. During the AFUP performing, the candidate itemsets after the pass 2 are stored and searched in a hash-tree which the same as in FUP. But in pass 2, AFUP adopts an array-data-structure method proposed in [6] to decrease the computation cost of pass 2. In addition, methods of reducing the database size used in FUP algorithm are also used in AFUP. For details, please refer to [4].

The same and the enlargement threshold cases Based on the definition of frequent itemset, the support of the itemsets in F_k must be great than or equal to s_{min}. In cases of $s_1 \geq s_{min}$, all frequent itemsets in D relative to s_1 must be in F_k. That is to say, the set R_k will be empty in these two cases. Therefore, the part 1 of AFUP algorithm showed in **Fig.** 1 is also applicable to these two cases directly.

4.2 The repeated maintaining process

When the user is not satisfied with the frequent itemsets derived from the first maintaining process, the maintaining process must be repeated continuously until the satisfying frequent itemsets are obtained. We assume it is the n-th ($n > 1$) maintaining process at present with the minimum support s_n. There are two alternatives to select s_n, i.e., $s_n > s_{n-1}$ and $s_n < s_{n-1}$. See in Section 2.2 for detailed.

During the n-th maintaining process, the itemsets to be reused are derived in the $(n-1)$-th maintaining process. Due to all itemsets in F_k^{n-1} include the support-counts in UD. So, in case of $s_n > s_{n-1}$, we just obtain the frequent k-itemsets in UD from F_k^{n-1} directly but do not need to scan the databases any more. It is very fast apparently. Therefore, we don't include this case in **Fig.** 1.

In case of $s_n < s_{n-1}$, similar to part 1, there are also seven steps (lines 20-26) in pass k. The dissimilarity between part 1 with part 2 is in step 3 (lines 10 and 22) and step 4 (lines 11 and 23). In step 3, we only prune the useless itemsets in R_k and Q_k but don't handle P_k any longer. And in step 4, we only computes the support-counts of itemsets in R_k and Q_k by Δ scan. Because the support-counts of itemsets in P_k in UD have known.

5. Empirical Results

In order to assess the performance of FMP and AFUP algorithms, extensive experiments have been conducted. And these experiments were compared with Apriori and FUP. The experiments were performed on a SUN workstation model 170E with a UltraSPARC CPU, 128 MB of main memory and, its OS is Solaris5.7. The databases used in our experiments are synthetic data generated by the IBM test data generator†3. We used the following parameters to generate the databases. Number of items in the databases is 1000 and the average length of transactions is 15. The number of maximal potentially frequent itemsets is set as 1000 and the average length of such frequent itemsets is set as 4.

The databases used in our environments are denoted by T15I4D100Δx, which represents an updated database UD in which the original database D has 100 thousands of transactions (D100) and the increment Δ has x thousands of transactions (Δx). For the increment Δ, we generated a series of databases $\Delta 1$, $\Delta 10$, $\Delta 25$, $\Delta 50$, $\Delta 75$ and $\Delta 100$ to

†3 http://www.almaden.ibm.com/as/quest/syndata.html.

conduct our experiments. To compare with the performance of Apriori, we run Apriori on the updated database UD, rather that on D and Δ separately.

The minimum supports used in our experiments are 1.75%, 1.5%, 1.25%, 1.0% 0.75% and 0.5%. We explain the experiments for the first maintaining process in Section 5.1. The repeated maintaining process experiments are discussed in Section 5.2.

5.1 First maintaining process experiments

We store in advance separately the frequent itemsets in the original database D relative to each minimum support above s_{min}. Then we add the six increment databases mentioned above to D.

Experiments with the same thresholds We performed the same threshold experiments with 1.5%, 1.25%, 1.0% and 0.75%. When we perform AFUP and FUP algorithms with each minimum support, the already stored frequent itemsets relative to the same minimum support are reused.

The experimental results show that AFUP is faster than Apriori by factors ranging from 3 to more than an order of magnitude. To compare with FUP algorithm, AFUP is still faster than FUP algorithm 2 to 3 times as well. We have given an experimental result in **Fig.** 2, in which the increment database (Δ100) has the same size of the original database (T15I4D100).

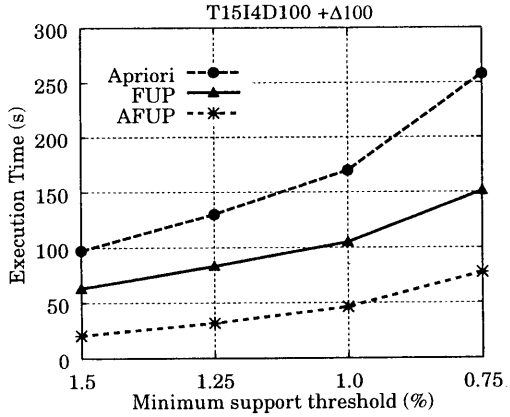

Fig.2 Experimental results with the same thresholds.

There are two reasons for AFUP being faster than Apriori algorithm. One is that the number of the candidate itemsets in AFUP is a lot less than that in Apriori algorithm when the original database is scanned. For example, the number of candidate itemsets in AFUP for T15I4D100Δ50 is only about 1% of in Apriori.

The second reason is that FRM adopts an array-data-structure[6] instead of the hash-tree used in Apriori algorithm in the pass 2 to improve the performance of this pass. This is also the main reason that AFUP is faster than FUP algorithm. According to our experimental results, as compared with FUP, the execution time of AFUP in the second pass speeds up from five to eight times by using the array-data-structure.

Experiments with the enlargement thresholds We perform our enlargement threshold experiments with a minimum support s_1 relative to s_{min} according to following table. For example, when we perform with s_1=0.75%, only the already stored frequent itemsets relative to s_{min}=1.0% are reused.

s_{min}	0.5%	0.75%	1.0%	1.25%
s_1	0.75%	1.0%	1.25%	1.5%

According to the characteristic of FUP algorithm, FUP should directly apply to the maintaining process in the case of enlarging the minimum support without any modification.

The experimental results demonstrate that AFUP is faster than Apriori by more than an order of magnitude for small increments database and 4 to 5 times for large increments database. As compared with FUP algorithm, AFUP is also faster by 2~5 times. The reasons are similar to that in the same threshold case. Since the minimum support is enlarged, the frequent itemsets in the updated database are almost contained in the already existed frequent itemsets. So, after the pass 2, AFUP almost obtains the new frequent itemsets as long as AFUP scans the increment database but doesn't scan the original database.

Experiments with the lessening thresholds In the experiments with lessening thresholds, the pair of s_{min} and s_1 used in our experiments are shown in following table.

s_{min}	1.75%	1.5%	1.25%	1.0%
s_1	1.5%	1.25%	1.0%	0.75%

Since FUP algorithm cannot run in this case directly, in order to compare AFUP with FUP, we perform FUP with an empty original database and a particular increment database, which is the updated database.

The experimental results show that AFUP is 2~3 times faster than Apriori algorithm and 2~4 times faster than FUP algorithm because the performance in pass 2 was improved by the array-data-structure and the candidate itemsets in each pass except in pass 1 were much decreased. For the FUP algorithm, it cannot use any already existed frequent

itemsets in this case. Moreover, it also spends extra overhead in the process of candidate itemset reduction. So it is also slower than Apriori.

5.2 Repeated maintaining process experiments

In the case of $s_n > s_{n-1}$, we obtain the new frequent itemsets without database scan as discussed in Section 4.2. It is very fast apparently. Therefore, we don't discuss this case here.

In the case of $s_n < s_{n-1}$, the minimum supports s_n relative to s_{n-1} in our experiments are given in following table. The experiments for FUP algorithm are also performed with an empty original database and a particular increment database, which is the updated database since the FUP algorithm isn't also applied to this case directly. The experimental results show that AFUP is faster than Apriori about 2~4 times and faster than FUP about 3-5 times..

s_n	1.5%	1.25%	1.0%	0.75%
s_{n-1}	1.75%	1.5%	1.2%	1.0%

6. Conclusions and Future Work

In this paper, we discussed the problem of maintenance of discovered frequent itemsets for mining association rules when some new data is added to a original transaction database. In general, in order to obtain satisfactory results to a user, the maintenance algorithm needs to be performed several times with different minimum support thresholds. We proposed algorithm AFUP for the maintenance of discovered frequent itemsets efficiently, which not only reuse the already existed frequent itemsets but also can be performed repeatedly with different minimum support thresholds until the satisfying results are obtained. Our algorithms is based on reusing efficiently the information of the already existed frequent itemsets and the information obtained in the previous pass to reduce the cost of finding new frequent itemsets.

We compared our algorithms to the existing well-known algorithms, Apriori and or FUP algorithms. We reported the results of our experiments with synthetic data, including one original database and six increment databases. The results showed that our algorithms always outperform Apriori and FUP algorithms no matter whether the size of the increment database is smaller than or equal to that of the original database.

An important perspective of future work is to extend this work to more general maintenance activities, including the cases of data addition, deletion and modification. Moreover, since sizes of databases become larger and larger, parallel maintaining process of discovered rules is also an important research task.

Acknowledgments

A part of this work is supported by the Grant-in-Aid for Scientific Research (10308012) from the Ministry of Education, Science, Sports and Culture of Japan.

References

1) R. Agrawal, T. Imielinski, A. Swami: Mining Associations between Sets of Items in Massive Databases. *Proc. of the 1993 ACM SIGMOD*, Washington, U.S.A., May, 1993.

2) R. Agrawal and R. Srikant: Fast Algorithms for Mining Association Rules. *Proc. of 20th Int. Conf. on VLDB*, Santiago, Chile, September, 1994.

3) D.W. Cheung, V. Ng, and B.W. Tam: Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules. *Proc. of 2nd Int. Conf. on Knowledge Discovery and Data Mining*, Portland, Oregon, August, 1996.

4) D.W. Cheung, J. Han, V. Ng, and C.Y. Wong: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Techniques. *Proc. of 12th IEEE Int. Conf. on Data Engineering*, New Orleans, Louisiana, U.S.A., March, 1996.

5) D.W. Cheung, S.D. Lee, B. Kao: A General Incremental Technique for Updating Discovered Association Rules. *Proc. Int. Conf. on Database Systems for Advanced Applications*, Melbourne, Australia, April, 1997.

6) X. P. Du, K. Kaneko, A. Makinouchi: Fast Algorithm to Find Frequent Itemsets for Mining of Association Rules. *Proc. of Int. Conf. on Information Society in the 21th Century: Emerging Technologies and new Challenges(IS2000)*. Fukushima, Japan, November, 2000.

7) Eui-Hong Han, George Karypis and Vipin Kumar: Scalable Parallel Data Mining foı Association Rules. IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, May/June, 2000.

8) S.D. Lee and D.W. Cheung: Maintenance of Discovered Association Rules: When to update? *Proc. of the 1997 ACM-SIGMOD Workshop on Data Mining and Knowledge Discovery*, Arizona, USA, May, 1997.

9) S.D. Lee D.W. Cheung, and B. Kao: Is Sampling Useful in Data Mining? A Case in the Maintenance of Discovered Association Rules. Data Mining and Knowledge Discovery, Kluwer Academic Publishers, V2, I3, September, 1998.

10) A. Savasere, E. Omiecinski and S. Navathe: An Efficient Algorithm for Mining Association Rules in Large Databases. *Proc. of 21th Int. Conf. on VLDB*, Zurich, Switzerland, September, 1995.