Maintenance of Discovered Knowledge : A Case in Multi-level Association Rules

David W. Cheung[†] Vincent T. Ng[‡] Benjamin W. Tam [†]

[†] Department of Computer Science, The University of Hong Kong, Hong Kong. Email: {dcheung|wktam}@cs.hku.hk.

[‡] Department of Computing, Hong Kong Polytechnic University, Hong Kong. Email: cstyng@comp.polyu.edu.hk.

Abstract

An incremental technique and a fast algorithm FUP have been proposed previously for the update of discovered single-level association rules(SLAR). In this study, a more efficient algorithm FUP*, which generates a smaller number of candidate sets when comparing with FUP, has been proposed. In addition, we have demonstrated that the incremental technique in FUP and FUP* can be generalized to some other kdd systems. An efficient algorithm MLUp has been proposed for this purpose for the updating of discovered multi-level association rules(MLAR). Our performance study shows that MLUp has a superior performance over ML-T2 in updating discovered MLAR.

1 Introduction

An association rule(AR) is a strong rule which implies certain association relationships among a set of objects in a database. Since finding interesting AR in databases may disclose some useful patterns for decision support, marketing analysis, financial forecast, system fault prediction, and many other applications, it has attracted a lot of attention in recent data mining research (6). Efficient mining of AR in transaction and/or relational databases has been studied substantially (1; 2; 8; 10; 12; 7; 11; 13).

In our previous study, we have investigated the maintenance problem in SLAR discovery (4). An efficient algorithm FUP (*Fast Update*) has been proposed, which can incrementally update the AR discovered, if updates to a database is restricted to insertions of new transactions. In this paper, we will report two progresses in our study of the maintenance problem in the mining of association rules. (1) A faster version FUP* of FUP has been proposed. The improvement of FUP* over FUP is in the candidate set generation procedure. (2) An algorithm MLUp (stands for *Multi-Level association rules Update*) has been proposed for the update of the discovered MLAR in relation databases (7).

The success of the incremental updating technique used in SLAR and MLAR suggests that, potentially, the technique could be generalized to solve the update problem in some other kdd systems. The remaining of the paper is organized as follows. In Section 2, the faster version FUP* is proposed. In Section 3, the problem of updating MLAR is discussed and the algorithm MLUp for the update of discovered MLAR is discussed. In Section 4, an in-depth performance study of MLUp is presented. Section 5 is the discussion and conclusions.

2 Update of Discovered SLAR

In the following discussion, we use the same notation as used in (4). We summarize the finding of (4) in Lemma 1. For a complete description of FUP, please see (4).

Lemma 1 (4) *A k-itemset X not in the original large k-itemsets L_k can become a* winner, *(i.e., become large) in the updated database $DB \cup db$ only if $X.support_d \geq s \times d$.* □

A faster update algorithm FUP*

The improvement of FUP* over FUP (4) is in the candidate set generation mechanism. FUP uses the Apriori-gen function defined in (2) to establish a set of candidate sets (4). In fact, it looks for itemsets in Apriori-gen(L'_{k-1}) which does not belong to L_k but appear in some transaction(s) in db, whose support count in db is larger than or equal to $s \times d$, where the set L'_{k-1} is the set of size-(k-1) large itemset in the update database found in the (k-1)-th iteration of FUP. We find out that the domain in this searching which is the set Apriori-gen(L'_{k-1}) can be further reduced to a smaller set. This finding is supported by the following result. (A result similar to Lemma 2 for partitioned databases has been reported in (5)).

Lemma 2 *A k-itemset X not in the original large k-itemsets L_k can become a* winner *(i.e., become large) in the updated database $DB \cup db$ only if $Y.support_d \geq s \times d$ for all the subsets $Y \subseteq X$.*

Proof. It follows from Lemma 1 that $X.support_d \geq s \times d$. If $Y \subseteq X$, then $Y.support_d \geq X.support_d$. Hence, the condition holds for all the subsets Y of X. □

From Lemma 2, the candidate sets can be restricted to the sets in Apriori-gen(L^*_{k-1}), where L^*_{k-1} are the itemsets in L'_{k-1}, whose support counts in *db* are larger than or equal to $s \times d$. In general, L^*_{k-1} is smaller than L'_{k-1}, and hence the number of candidate sets in Apriori-gen(L^*_{k-1}) is smaller than that in Apriori-gen(L'_{k-1}). In the following, we will use Example 1 to illustrate the execution of FUP*. In particular, the example will show that FUP* can reduce significantly the number of candidate sets.

Example 1 A database DB is updated with an increment *db* such that $D = 1000$, $d = 100$ and $s = 3\%$. X, Y, Z, and W are four items and the size-1 and size-2 large itemsets in DB are $L_1 = \{X, Y, Z\}$ and $L_2 = \{XY, YZ\}$, respectively. Also $XY.support_D = 32$ and $YZ.support_D = 31$. Suppose FUP* has completed the first iteration and found the "new" size-1 itemsets $L'_1 = \{X, Y, W\}$. Moreover, assuming that the support counts of X, Y, and W found in *db* are 2, 4 and 5, respectively. This example illustrates how FUP* will find out L'_2 in the second iteration, and also its effectiveness in reducing the number of candidate sets.

FUP* first filters out losers from L_2. Note that $Z \in L_1 - L'_1$, i.e., Z has become a loser; therefore, the set $YZ \in L_2$ must also be a loser and is filtered out. For the remaining set $XY \in L_2$, FUP* scans *db* to update its support count. Assume that $XY.support_{db} = 2$. Since $XY.support_{UB} = (2+32) > 3\% \times 1100$, therefore, XY is large in $DB \cup db$ and is stored in L'_2.

Secondly, FUP* needs to find out the "new" large itemsets from *db*. For this purpose, FUP* has to find out the set L^*_1 from L'_1, which contains the itemsets in L'_1 that have enough support counts in *db*. Since $X.support_d = 2 < 3\% \times 100$, $X \notin L^*_1$, i.e., even though X is a winner in the 1st iteration, it will not be used to generate the size-2 candidate sets. On the other hand, both the support counts of Y and W in *db* are larger than the threshold $3\% \times 100$; therefore they will be used to generate the size-2 candidate sets, i.e., $L^*_1 = \{Y, W\}$. Following that, FUP* applies Apriori-gen on L^*_1 and generates the candidate set $C_2 = \{YW\}$. Note that in FUP, Apriori-gen is applied on $L'_1 = \{X, Y, W\}$ instead of L^*_1, and the set of candidate sets generated will have three itemsets which is three times larger than what is generated in FUP* in this example. This illustrates that FUP* can significantly and effectively reduce the number of candidate sets when comparing with FUP.

Suppose $YW.support_d = 4 > 3\% \times 100$. It follows from Lemma 1 that YW will not be pruned and remain in C_2. Following the pruning of the candidate sets in C_2, FUP* has to update the remaining candidate sets in C_2 against the original database DB. Suppose $YW.support_D = 29$. Since $YW.support_{UD} = 29 + 4 > 3\% \times 1100$, it is a large itemset in the updated database. Therefore YW is added into L'_2. At the end of the second iteration, $L'_2 = \{XY, YW\}$ is returned. □

3 Update of Discovered MLAR

The method used in FUP* (and FUP) could be applied to many other kdd systems to update the knowledge discovered. In particular, it can be used in the systems that are designed to discover various types of associations between generalized items and events. This includes the discovery of MLAR, generalized AR, sequential patterns, episodes, and quantitative AR (7; 12; 3; 9; 13). In the following, we will show that FUP* can be generalized to solve the update problem for MLAR. For this purpose, an algorithm MLUp, which is an adaptation of FUP*, will be proposed.

Mining of MLAR

In the study of mining MLAR, a series of algorithms have been proposed to facilitate a top-down, progressive deepening method based on the algorithms for mining SLAR. The method first finds large data items at the top-most level and then progressively deepens the mining process into their large descendants at lower concept levels. For details on the mining of MLAR, please refer to (7).

Update of discovered MLAR

The problem of updating the discovered MLAR is the same as that in the single-level environment. The only difference is that the rules in all the levels have to be updated instead of updating the rules in only one level. Also, the minimum support thresholds at different levels may not be equal. We use s_m to denote the minimum support threshold at level m for $m \geq 1$.

Since there are several variations of the algorithm in mining MLAR, the update algorithm should be designed according to the strategy used in the initial mining process. The algorithm MLUp we are proposing is associated with the representative mining algorithm ML-T2. The following two results are the bases of MLUp.

Lemma 3 *In a multi-level environment, a level-m 1-itemset X not in the original large 1-itemsets $L[m, 1]$, $(m \geq 1)$, can become a* winner *(i.e., become large) in the updated database $DB \cup db$ only if all ancestors of X are winners.*

Proof. This follows from the definition of large itemsets in the multi-level environment. □

Following Lemma 3, when MLUp scans the increment *db* to look for new size-1 winners, it not only has to ensure a candidate itemset has the required support count, but must also check that all its ancestors are large in the updated database. (Because of transitivity, MLUp only needs to check a candidate's immediate ancestor).

Lemma 4 *In a multi-level environment, a level-m k-itemset X not in the original large k-itemsets $L[m, k]$, $(m \geq 1)$, can become a* winner *(i.e., become large) in the updated database $DB \cup db$ only if $X.support_d \geq s_m \times d$ and $Y.support_d \geq s_m \times d$, for all subset $Y \subseteq X$.*

Proof. This follows directly from Lemmas 1 and 2. □

The implication of the result in Lemma 4 is that the candidate set generation mechanism in FUP* can be applied directly in MLUp for finding new winners in different levels. In the following, we describe the main procedure of the update algorithm MLUp. The input to the algorithm includes the original encoded transaction database $T[1]$, the increment database db, and the old large itemsets $L[m,k]$, $(m \geq 1,\ k \geq 1)$, and their support counts. Following the conventions in FUP*, the sizes of $T[1]$ and db are denoted by D and d respectively. Moreover, the minimum support threshold for different level is denoted by s_m, $(m \geq 1)$.

MLUp (main steps) :

1. Translate the increment transaction database db into an encoded transaction table $db[1]$ according to the given taxonomy information.

2. At level 1, scan $db[1]$ to update the support counts of the 1-itemsets in $L[1,1]$ to filter out the winners into $L'[1,1]$. In the same scan, find all the 1-itemsets in $db[1]$ which do not belong to $L[1,1]$, whose support count in $db[1]$ is larger than or equal to $s_1 \times d$, and store these 1-itemsets in the candidate set C_1. Subsequently, scan $T[1]$ to find out the new winners in C_1 and store them into $L'[1,1]$. Following that, $T[1]$ is filtered by $L'[1,1]$ to generate the encoded transaction table $T[2]$. Similarly, $db[1]$ is filtered to $db[2]$.

At level m, $(m > 1)$, scan $db[2]$ to update the support counts of the 1-itemsets in $L[m,1]$. An 1-itemset in $L[m,1]$ is a winner only if its immediate ancestor is large in the updated database and its support counts in the updated database is larger than or equal to $s_m \times (D+d)$.

In the same scan, find all level-m 1-itemsets in $db[2]$ which do not belong to $L[m,1]$, whose immediate ancestor belongs to $L'[m-1,1]$ and whose support count in $db[2]$ is larger than or equal to $s_m \times d$. Then store these 1-itemsets in the candidate set C_1. Subsequently, scan $T[2]$ to find out the new winners in C_1 and store them in $L'[m,1]$.

3. The large k-itemsets, $(k > 1)$, for the updated database at level m is derived in three steps:

(1) Remove all the k-itemsets in $L[m,k]$ for which one of its ancestors is not large in the updated database. Then scan $db[2]$ to update the support counts of the remaining itemsets in $L[m,k]$ to find out the winners. (2) Let $L^*[m,k-1]$ be the subsets of itemsets in $L'[m,k-1]$ whose support count in db is larger then or equal to $s_m \times d$. In the same scan on $db[2]$ performed in (1), find all level-m k-itemsets in Apriori-gen($L^*[m,k-1]$) which do not belong to $L[m,k]$, whose support count in $db[2]$ is larger than or equal to $s_m \times d$, and store them in the candidate set C_k. (3) Scan DB to update the support counts of the candidate sets in C_k and find all the level-m size-k winners in C_k, and store them in $L'[m,k]$.

4. At level m, return the union of $L'[m,k]$ for all the k's.

4 Performance Study of MLUp

Extensive experiments have been conducted to assess the performance of MLUp. It was compared with the algorithm ML-T2. The experiments were performed on an AIX system on an RS/6000 workstation with model 410. The result shows that MLUp is much faster than re-running ML-T2 to update the discovered AR. This improvement is not surprising given that FUP also has similar performance in updating SLAR. The databases used in our experiments are synthetic data generated using a technique similar to that in (2).

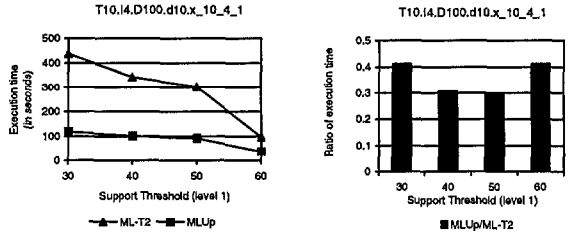

Figure 1: Performance Comparison (level 1)

Our test environments are denoted by T10.I4.D100.d10.s_1_s_2_s_3_s_4, which represents an updated database in which the original database DB has 100 thousands of transactions (D100), the increment db has 10 thousands of transactions (d10). The transactions on average has 10 items (T10), and the average size of the large itemsets is 4 (I4). Moreover, there are four levels in the taxonomy and the minimum supports are denoted by s_i, $(1 \leq i \leq 4)$. The performance comparison between MLUp and ML-T2 in the update of the level-1 AR is plotted in Figure 1 against different minimum support thresholds. Their performance ratios are also presented as bar charts in the same figure. It can be seen that MLUp is 2-3 times faster than ML-T2. MLUp also has similar speed-up over ML-T2 in the updates in the other levels.

As explained before, MLUp reduces substantially the number of candidate sets generated when comparing with ML-T2. In Figure 2, the number of candidate sets generated in MLUp in the same experiment is compared with that in ML-T2. The ratios in the comparison are presented as bar charts in the same figure. The chart shows that the number of candidate sets generated by MLUp is only about 2-3% of that in ML-T2.

A series of updates from 10K to 350K were generated on the databases T10.I4.D100, and the execution times for MLUp and ML-T2 to do the updates on these increments were compared. A gradually level off of the speed-up of MLUp over ML-T2 only appears when the increment size is about 3.5 times the size of the original database. The fact that MLUp still exhibits performance gain when the increment is much larger than the original database shows that it is very efficient.

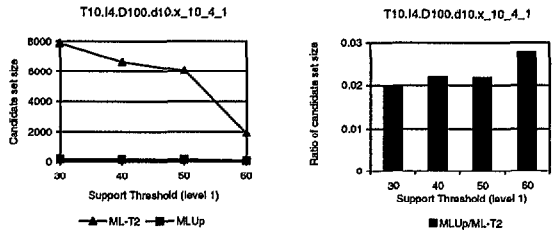

Figure 2: Reduction of Candidate Sets (level 1)

5 Discussion and Conclusions

We have shown that FUP* is an efficient algorithm for updating discovered SLAR. It improves the performance of FUP by significantly reducing its candidate sets.

We have also proposed an efficient algorithm MLUp for updating discovered MLAR. It is an adaptation of the FUP* algorithm in the multi-level environment. The algorithm MLUp is implemented and its performance is studied and compared with the ML-T2 algorithm . The study shows that MLUp has superior performance in the multi-level environment. The success of the incremental updating technique in both the SLAR and MLAR suggests that the technique could be generalized to solve the update problems in some other knowledge discovery systems.

Currently, both FUP* and MLUp are applicable only to a database which allow frequent or occasional updates restricted to insertions of new transactions. We have also investigated the cases of updates including deletions and/or modifications to a transaction database. In FUP* and MLUp, the incremental updating technique has made use of the fact that new winners generated in the updating process must appear and have enough support counts in the increment. However, this does not hold in general in the cases of deletion and modification. For example, in the case of deletion, because the size of the updated database has decreased, some itemsets which are "small" in the orginial database DB, could become large in the updated database, even though it is not contained in any transaction deleted. Consequently, the set of candidate sets cannot be limited to those appear in the increment, and potentially, all itemsets in the updated database have to be considered as candidates. Therefore, the current incremental technique cannot be applied directly to the cases of deletion and modification. However, it is possible to solve the deletion and modification cases if the initial mining process is enhanced to retain more informations to support the update.

The extension of our incremental update technique for the maintenance of other type of knowledge such as generalized AR, episodes, sequential patterns, and quantitative AR is an interesting topic for future research. However, as discussed above, a bigger challenge is to extend this technique to cover the cases of deletion and modification.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. VLDB*, pp. 487–499, Santiago, Chile, Sept. 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pp. 3–14, Taipei, Taiwan, March 1995.

[4] D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 Int'l Conf. on Data Engineering*, New Orleans, Louisiana, Feb. 1996.

[5] D.W. Cheung, J. Han, V. Ng, A. Fu and Y. Fu. A Fast Distributed Algorithm for Mining Association Rules. Technical Report, Dept. of Computer Science, The University of Hong Kong, 1996.

[6] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[7] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. VLDB*, pp. 420–431, Zurich, Switzerland, Sept. 1995.

[8] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int'l Conf. on Information and Knowledge Management*, pp. 401–408, Gaithersburg, Maryland, Nov. 1994.

[9] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering Frequent Episodes in Sequences. In *Proc. 1st Int'l Conf. on KDD*, pp. 210–215, Montreal, Quebec, Canada, Aug. 1995.

[10] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pp. 175–186, San Jose, CA, May 1995.

[11] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. VLDB*, pp. 432–443, Zurich, Switzerland, Sept. 1995.

[12] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. VLDB*, pp. 407–419, Zurich, Switzerland, Sept. 1995.

[13] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, Montreal, Canada, June 1996.