An Efficient Approach for Incremental Association Rule Mining

Pauray S.M. Tsai [1], Chih-Chong Lee [2], and Arbee L.P. Chen [2]

[1] Department of Information Management, Ming Hsin Institute of Technology, Hsin-Feng, Hsinchu 304, Taiwan, R.O.C.
pauray@mis.mhit.edu.tw

[2] Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C.
alpchen@cs.nthu.edu.tw

Abstract. In this paper, we study the issue of maintaining association rules in a large database of sales transactions. The maintenance of association rules can be mapped into the problem of maintaining large itemsets in the database. Because the mining of association rules is time-consuming, we need an efficient approach to maintain the large itemsets when the database is updated. In this paper, we present efficient approaches to solve the problem. Our approaches store the itemsets that are not large at present but may become large itemsets after updating the database, so that the cost of processing the updated database can be reduced. Moreover, we discuss the cases where the large itemsets can be obtained without scanning the original database. Experimental results show that our algorithms outperform other algorithms, especially when the original database need not be scanned in our algorithms.

1 Introduction

Data mining has attracted much attention in database communities because of its wide applicability. One major application area of data mining is to discover potentially useful information from transaction databases. The problem of mining *association rules* from transactional data was introduced in [1]. A transaction in the database consists of a set of items (itemset). An example of such an association rule might be " 80% of customers who buy itemset X also buy itemset Y". The *support count* of an itemset is the number of transactions containing the itemset, and the *support* of the itemset is the fraction of those transactions. The itemset ($X \cup Y$ in the example) involved in an association rule must be contained in a predetermined number of transactions. The predetermined number of transactions is called the *minimum support count*, and the fraction of transactions is called the *minimum support threshold*. An itemset is called a *large itemset* if its support is no less than the minimum support threshold. Since the generation of association rules is straightforward after the large itemsets are discovered, finding large itemsets becomes the main work of mining association rules.

A database is “dynamic” in the sense that users may periodically or occasionally insert data to or remove data from the database. The update to the database may cause not only the generation of new rules but also the invalidation of some existing rules. Thus, the maintenance of the association rules is a significant problem. One possible approach to the problem is to rerun the mining algorithm on the updated database. This approach is simple but inefficient since it does not use the existing information such as the support counts of the current large itemsets. In [2], FUP algorithm is proposed for the maintenance of discovered association rules in large databases. If there are frequent itemsets in the increment database, which are not large itemsets of the original database, then the algorithm scans the original database to check whether they are large itemsets or not. FUP, FUP* [3] and FUP2 [4] all belong to the kind of generate-and-test algorithms, and are referred to as k-pass algorithms because they have to scan the database k times. Two algorithms DIUP and DDUP [5], extended from DLG [7], are proposed to handle the incremental update problem. An association graph and an information table are constructed to record whether a previously generated large itemset remains large in the updated database. With this information, these two algorithms can finish the update work in one database scan.

In this paper, we present efficient approaches to maintain large itemsets. Our approaches store the itemsets that are not large in the original database but may become large itemsets after the database is updated, so that the cost of processing the updated database can be reduced. Moreover, we discuss the cases where the large itemsets can be obtained without scanning the original database. Experimental results show that our algorithms outperform FUP and FUP2 algorithms, especially when the original database need not be scanned in our algorithms.

The remaining of the paper is organized as follows. The problem description is given in Section 2. In Section 3, our approaches are proposed for the maintenance of large itemsets. Performance results are presented in Section 4. Section 5 contains the conclusion.

2 Problem Description

Let L be the set of all the large itemsets in the original database DB, s be the minimum support threshold, and d the number of transactions in DB. Assume that the support count of each large itemset in the original database has been recorded. After some updates, the increment database DB^+ is added into the original database DB and the decrement database DB^- is removed from the original database, resulting in the updated database DB'. The numbers of transactions in DB^+, DB^-, and DB' are denoted d^+, d^-, and d', respectively, and the support counts of itemset X in DB, DB^+, DB^-, and DB' are X_{count}, X^+_{count}, X^-_{count}, and $X^{'}_{count}$, respectively. With the same minimum support threshold s, a large itemset X of DB remains a large itemset of DB' if and only if its support in DB' is no less than s, i.e. $X^{'}_{count} \geq (d + d^+ - d^-) * s = d' * s$.

All the itemsets in DB can be divided into the following four groups:

Group 1: the itemsets which are large in both DB and DB'.

Group 2: the itemsets which are large in DB but not large in DB'.

Group 3: the itemsets which are not large in DB but large in DB'.
Group 4: the itemsets which are neither large in DB nor DB'.

The itemsets of group 4 can not be used to generate any rule because their supports are less than the minimum support threshold. Thus, these itemsets are indifferent. The itemsets in group 1 and group 2 can be obtained easily. For example, X is a large itemset of DB. The support count of X in DB' is $X'_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$, where X_{count} is known and X^{+}_{count} and X^{-}_{count} are available after scanning DB^{+} and DB^{-}, respectively. However, the itemsets of group 3 need further consideration. The support counts of these itemsets in DB' can not be determined after scanning DB^{+} and DB^{-} since their support counts in DB are unknown in advance. Thus, how to efficiently generate the itemsets of group 3 is an important problem for the maintenance of association rules. In the following section, we propose efficient approaches to solve the problem.

3 Our Approaches

Basically, the framework of our algorithm is similar to that of the FUP algorithm [3]. We store the large itemsets of DB and their support counts in the set L. Let t be the *tolerance degree*, $0 < t < s$. The degree is used to control the number of itemsets whose supports are less than the minimum support threshold s but no less than (s-t). We call these itemsets the *potential* large itemsets. The potential large itemsets and their support counts are stored in the set PL. It is easy to see that the more the number of the potential large itemsets is, the less the cost of processing the original database will be. Besides, we record the itemsets of the increment database, that are not in L and PL but may become large itemsets, in the set M.

3.1 Insertion

Let DB and DB^{+} be the original database and the increment database, respectively. Assume that the size of DB and DB^{+} are d and d^{+}, respectively, and sets L and PL are available before we update DB. When the increment database DB^{+} is scanned, the support counts of the itemsets in $L \cup PL$ are accumulated. Then L and PL are updated according to the support count thresholds $s*(d+d^{+})$ and $(s-t)*(d+d^{+})$, respectively. Let X be an itemset of DB^{+} and $X \notin L \cup PL$. By Theorem 1, if the support count of X in DB^{+} is greater than $d^{+}*(s-t)$, then X may be a large or potential large itemset of the updated database DB'. In the case, the itemset X is stored in set M. Let *MS* be the maximal value of the support counts of itemsets in M. By Theorem 2, if $MS < d^{+}*s + d*t$, we need not scan DB. Namely, in this case all itemsets in M can not be large itemsets of the updated database. The Insert algorithm is shown in [6].

Theorem 1: Assume that the support counts of itemset X in the original database DB, the increment database DB^+, and the updated database DB' are X_{count}, X^+_{count}, and X'_{count}, respectively. If $X \notin L \cup PL$ and $X^+_{count} \le d^+ * (s - t)$, then X is not a large or potential large itemset of DB'.

[PROOF]

$$\begin{aligned} X'_{count} &= X_{count} + X^+_{count} \\ &< d * (s - t) + X^+_{count} \\ &\le d * (s - t) + d^+ * (s - t) \\ &= (d + d^+) * (s - t) \end{aligned}$$

Therefore, X is not a large or potential large itemset of DB'.

Theorem 2: If $MS < d^+ * s + d * t$, then we need not scan the database DB.

[PROOF]

Assume that X is the itemset in M, which has the maximal support count.

Since $X \notin L \cup PL$, $X_{count} < d * (s - t)$.

Therefore, if $MS + d * (s - t) < (d + d^+) * s$

(that is, $MS < (d^+ * s + d * t)$), then X is not a large itemset in the updated database DB'. In other words, all the itemsets in M can not be large itemsets of DB'.

3.2 Deletion

The user may remove out-of-date data from the database. In the following, we discuss the maintenance of association rules when deletions occur. Let DB and DB^- be the original database and the decrement database, respectively. Assume that the sizes of DB and DB^- are d and d^-, respectively, and sets L and PL are available before we update database DB. When the decrement database DB^- is scanned, the support count of the itemset in $L \cup PL$ lessens if this itemset appears in DB^-. Then L and PL are updated according to the support count thresholds $s * (d - d^-)$ and $(s-t) * (d - d^-)$, respectively. Let X be an itemset of DB and $X \notin L \cup PL$. By Theorem 3, if $d^- \le d * t/s$, then we need not process the updated database $(DB - DB^-)$. Otherwise, we use DLG algorithm [7] to generate the large itemsets in the updated database. The Delete algorithm is shown in [6].

Theorem 3: If $d^- \le d * \frac{t}{s}$, i.e., $t \ge s * \frac{d^-}{d}$, then we need not scan the updated database DB'.

[PROOF]

Assume that X is an itemset of DB.

$< 1 > X \in L \cup PL$

Since $X'_{count} = X_{count} - X^-_{count}$

and X_{count} and X^-_{count} are available,

we need not scan DB'.

$< 2 > X \notin L \cup PL$

$$\begin{aligned} \text{Since } X'_{count} &= X_{count} - X^-_{count} \\ &< d * (s - t) - X^-_{count} \\ &\le d * s - d * t \\ &\le d * s - d * s * \frac{d^-}{d} \\ &= d * s - d^- * s = d' * s \end{aligned}$$

$\Rightarrow$ X is not a large itemset of DB'.

$\Rightarrow$ We need not scan DB'.

By $< 1 >< 2 >$, the theorem is proved.

3.3 Update with Insertion and Deletion

In the following, we consider insertion and deletion at the same time. Let DB, DB^+, and DB^- be the original database, the increment database, and the decrement database, respectively. Assume that the sizes of DB, DB^+, and DB^- are d, d^+, and d^-, respectively. When DB^+ and DB^- are scanned, the support count of the itemset in $L \cup PL$ is updated. Consider the following two cases.

Case 1: $d^+ \ge d^-$

(1) Let X be an itemset of DB^+ and $X \notin L \cup PL$. If the support count of X in DB^+ is no less than $(d^+ - d^-) * (s-t)$, it may be a large or potential large itemset of the updated database. In this case, itemset X is stored in set M. It is proved as follows:

[PROOF]

Assume $X^{+}_{count} < (d^{+} - d^{-})*(s-t)$.

$$X'_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$$
$$< d*(s-t) + (d^{+} - d^{-})*(s-t)$$
$$< (d + d^{+} - d^{-})*(s-t)$$

Thus X is not a large or potential large itemset of the updated database.

(2) Let X be an itemset of DB but not an itemset of DB^{+}. Assume $X \notin L \cup PL$. Then X can not be a large itemset of the updated database. It is proved as follows:

[PROOF]

$$X'_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$$
$$< d*(s-t)$$
$$\leq (d + d^{+} - d^{-})*(s-t)$$

Thus X is not a large or potential large itemset in the updated database.

Case 2: $d^{+} < d^{-}$

Let X be an itemset of DB but not an itemset of DB^{+} or DB^{-}. Assume $X \notin L \cup PL$. Then X may be a large itemset in the updated database. It is proved as follows:

[PROOF].

$$X'_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$$
$$< d*(s-t)$$

Since $d*(s-t) > (d + d^{+} - d^{-})*(s-t)$, X may be a large itemset of the updated database. In the case, we use DLG algorithm [7] to process the updated database and generate the sets L and PL.

Let *MS* be the maximal value of the support counts of itemsets in M. By Theorem 4, if $MS \leq d*t + d^{+}*s - d^{-}*s$, we need not process the updated database. The Update algorithm is shown in [6].

Theorem 4: If $MS \leq d*t + d^{+}*s - d^{-}*s$, we need not scan the updated database DB'.

[PROOF]

Assume that X is an itemset of $DB \cup DB^{+}$.

$< 1 >$ $X \in L \cup PL$

Since $X^{'}_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$

and $X_{count}, X^{+}_{count}, X^{-}_{count}$ are available, we need not scan $DB^{'}$.

$< 2 >$ $X \notin L \cup PL$

Since $X^{'}_{count} = X_{count} + X^{+}_{count} - X^{-}_{count}$

$< d*(s-t) + X^{+}_{count} - X^{-}_{count}$

$\leq d*s - d*t + MS - X^{-}_{.count}$

$\leq d*s - d*t + MS$

$\leq d*s - d*t + (d*t + d^{+}*s - d^{-}*s)$

$= d*s + d^{+}*s - d^{-}*s = d^{'}*s$

$\Rightarrow$ X is not a large itemset of $DB^{'}$.

$\Rightarrow$ We need not scan $DB^{'}$.

By $< 1 >< 2 >$ the theorem is proved.

4 Experimental Results

To assess the performance of our algorithms for the maintenance of large itemsets, we perform several experiments on Sun SPARC/10 workstation.

4.1 Generation of Synthetic Data

In the experiments, we used synthetic databases to evaluate the performance of the algorithms. The data is generated using the same approach introduced in [2]. The parameters used to generate the database are described in Table 1.

Table1. The parameters

d	Number of transactions in the original database DB
d^{+}, d^{-}	Number of transactions in the increment/decrement

	databases (DB^+ / DB^-)
$\|I\|$	Average size of the potentially large itemsets
$\|MI\|$	Maximum size of potentially large itemsets
$\|L\|$	Number of the potentially large itemsets
$\|T\|$	Average size of the transactions
$\|MT\|$	Maximum size of the transactions

We generate the dataset by setting N=1000, **d**=100,000, $|L|$=2000, $|I|$=3, $|MI|$=5, $|T|$=5, and $|MT|$=10. The way we insert the increment database DB^+ is to put the d^+ transactions in the rear of the original database. And the way we delete the decrement database DB^- is to remove the first d^- transactions from the original database.

4.2 Comparison of Insert Algorithm with FUP

We perform an experiment on the dataset with minimum support 1%. The experiment run eight insertions of increment databases and the relative execution time of each insertion is recorded for Insert algorithm and FUP algorithm. A new database is generated for each insertion. Fig. 1 shows the relative execution times in the cases of t=0.1% and t=0.5%. The size of each increment database is 1% of the size of the original database. In this experiment, Insert algorithm takes much less time than FUP algorithm since Insert algorithm does not scan the original database.

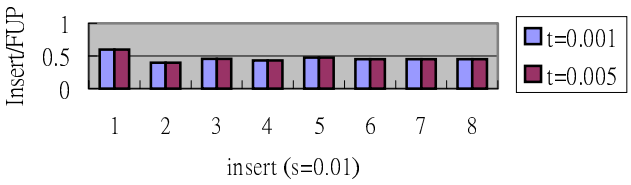

Fig. 1. Relative execution time for Insert and FUP (d^+ :d = 1:100)

4.3 Comparison of Delete Algorithm with FUP2

In the following, we compare Delete algorithm with FUP2 algorithm. The minimum support considered is also 1%. In Fig. 2, the size of each decrement database is 1% of the size of the original database and t=0.1% and t=0.5% are considered respectively. It can be seen that Delete algorithm take much less time than FUP2.

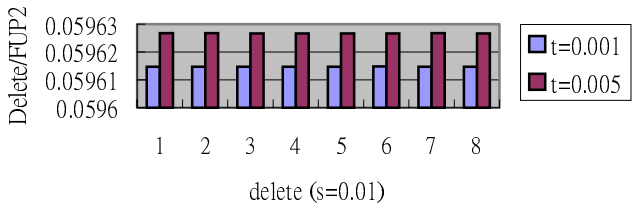

Fig. 2. Relative execution time for Delete and FUP2 (d^-:d = 1:100)

4.4 Comparison of Update Algorithm with FUP2

We assess the performance of Update algorithm considering insertion and deletion at the same time. In Fig. 3, the sizes of the increment database and the decrement database are 0.1% of the size of the original database. It can be seen that Update algorithm takes much less time than FUP2 algorithm since the size of the original database is much larger than that of the increment database and the decrement database.

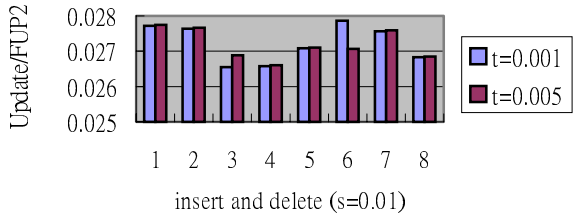

Fig. 3. Relative execution time for Update and FUP2 (d^+:d^-:d = 1:1:1000)

5 Conclusion

The database users may frequently or occasionally update the database. This behavior would change the characteristic of the database. To efficiently discover and maintain large itemsets in a large database, we present three algorithms to solve the problem:

1. Insert algorithm for the insertion of the increment database to the original database.

2. Delete algorithm for the deletion of the decrement database from the original database.
3. Update algorithm considering both the increment database and the decrement database at the same time.

In our algorithms, we store more information than other algorithms so that the cost of processing the original database can be reduced. Moreover, we discuss the cases where the large itemsets can be obtained without scanning the original database. Experimental results show that our algorithms outperform other algorithms, especially when the size of the original database is much larger than that of the increment database and the decrement database.

Acknowledgments

This work was partially supported by the Republic of China National Science Council under Contract No. NSC 88-2213-E-007-052 and NSC 88-2213-E-159-003.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules between Sets of Items in Large Databases, Proc. ACM SIGMOD, (1993) 207-216
2. Cheung, D.W., Han, J., Ng, V.T., Wong, C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique, Proc. the International Conference on Data Engineering, (1996) 106-114
3. Cheung, D.W., Ng, V.T., Tam, B.W.: Maintenance of Discovered Knowledge: A case in Multi-level Association Rules, Proc. 2nd International Conference on Knowledge Discovery and Data Mining, (1996) 307-310
4. Cheung, D.W., Lee, S.D., Kao, B.: A general Incremental Technique for Mining Discovered Association Rules, Proc. International Conference on Database System for Advanced Applications, (1997) 185-194
5. Lee, K.L., Efficient Graph-Based Algorithms for Discovering and Maintaining Knowledge in Large Databases, NTHU Master Thesis, (1997)
6. Tsai, P.S.M., Lee, C.C., Chen, A.L.P.: An Efficient Approach for Incremental Association Rule Mining, Technical Report, (1998)
7. Yen, S.J., Chen, A.L.P.: An Efficient Approach to Discovering Knowledge from Large Databases, Proc. the IEEE/ACM International Conference on Parallel and Distributed Information System, (1996) 8-18