An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases

Shiby Thomas **Sreenath Bodagala** **Khaled Alsabti*** **Sanjay Ranka**
Computer and Information Science and Engineering Department
University of Florida, Gainesville FL 32611
email: {sthomas, sre, kalsabti, ranka}@cise.ufl.edu

Abstract

Efficient discovery of association rules in large databases is a well studied problem and several approaches have been proposed. However, it is non trivial to maintain the association rules current when the database is updated since, such updates could invalidate existing rules or introduce new rules. In this paper, we propose an incremental updating technique based on *negative borders*, for the maintenance of association rules when new transaction data is added to or deleted from a transaction database. An important feature of our algorithm is that it requires a full scan(exactly one) of the whole database only if the database update causes the negative border of the set of large itemsets to expand.

Introduction

Database mining, or knowledge discovery in databases (KDD) has recently attracted tremendous amount of attention in the database research community because of its wide applicability in many areas, including decision support, market strategy and financial forecast. One of the important characteristics of the database mining problem is handling large volumes of data. Further, the rules discovered from a database only reflect the current state of the database. In order to make the rules discovered reliable and useful, large volumes of data need to be collected and analyzed over a period of time. This entails the development of techniques to handle large volumes of data, and to maintain rules over a significantly long period of time. Therefore, efficient algorithms to update, maintain and manage the discovered rules are central to the database mining technology.

The problem of mining association rules over basket data was introduced in (Agrawal, Imielinski, and Swami 1993) and several algorithms have been proposed (Agrawal and Srikant 1994; Savasere, Omiecinski, and Navathe 1995; Toivonen 1996). These algorithms provide efficient mechanisms for finding *large* itemsets (itemsets with the user specified minimum support) and the association rules are computed from the large itemsets. Updates to the transaction database could potentially invalidate existing rules or introduce new rules. The problem of updating the association rules can be reduced to finding the new set of large itemsets. After that, the new association rules can be computed using the new large itemsets. A simple solution to the update problem is to re-compute the large itemsets of the whole updated database. This is clearly inefficient because all the computations done initially for finding the old large itemsets are wasted. An algorithm, FUP (Fast Update), for updating the large itemsets has been developed for the addition of new transactions to the database (Cheung et al. 1996). It is based on the Apriori algorithm and needs $O(n)$ passes over the database where n is the size of the maximal large itemset.

In this paper, we present an algorithm to find the new large itemsets with minimal re-computation when new transactions are added to or deleted from the transaction database. The important characteristics of our algorithm are the following:

- Along with the large itemsets, we also maintain the *negative border* (Toivonen 1996)[1]. The algorithm uses negative borders to decide when to scan the whole database and it can be used in conjunction with any level-wise algorithm like Apriori or Partition.
- We first compute the large itemsets of the increment database. The algorithm requires a full scan of the whole database only if the negative border of the large itemsets expands, that is, if an itemset outside the negative border gets added to the large itemsets or its negative border. Even in such cases, it requires only one I/O pass over the whole data set.
- The algorithm can be easily parallelized with minimal communication and synchronization between the processing nodes. This is particularly important since large volumes of data need to be handled. However, we are not including the details of parallelization in this paper due to space limitations.

Feldman et al. concurrently developed a similar approach for discovering frequent sets in incremental databases (Feldman et al. 1997). The rest of the paper is organized as follows: We develop our incremental updation algorithm in the next section. The experimental results and a comparison of our incremental algorithm

---

* Visiting from the Department of Computer Science, Syracuse University

[1] Note that the negative border can be maintained while computing the large itemsets without any additional computation overhead.

with FUP is presented next. The last section concludes the paper and outlines possible extensions.

Incremental updation of association rules

In this section we develop an efficient method for updating the association rules when the database is updated. Since we deal mainly with *basket* data, database update effectively means addition of new transactions to the database or deletion of existing transactions. Assuming that the two thresholds, minimum support and confidence, do not change the update problem can be reduced to finding the new set of large itemsets. After that, the new association rules can be computed from the new large itemsets. In this paper we concentrate on updating the large itemsets. Before we explain the algorithm in detail, we explain some results which aid in presenting the algorithm.

Computing $\mathcal{NB}d(L)$ from L

In this subsection, we explain how to compute the *negative border* ($\mathcal{NB}d(L)$) of a set of large itemsets L. This can be accomplished by repeating the join and prune steps of the *apriori-gen* function in the apriori algorithm (Agrawal and Srikant 1994). This computation can be done using only the set of large itemsets L and the database need not be scanned.

Definition 1 *The* negative border $\mathcal{NB}d(L)$, *of a collection of itemsets L i s defined as follows: Given a collection $L \subseteq \mathcal{P}(R)$ of sets, closed with respect to the set inclusion relation, the negative border $\mathcal{NB}d(L)$ of L consists of the minimal itemsets $X \subseteq R$ not in L (Mannila and Toivonen 1996).*

The *apriori-gen* function takes as argument L_{k-1}, the set of all large $(k-1)$-itemsets. It returns a superset of the set of all large k-itemsets.

---

function apriori-gen(L_{k-1})
    for each p and $q \in L_{k-1}$ *do*
        if $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}$ and $p.item_{k-1} < q.item_{k-1}$ *then* insert $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$ into C_k
    for each $c \in C_k$
        delete c from C_k if some $(k-1)$-subset of c is not in L_{k-1}

Figure 1: A high-level description of the apriori-gen function

The negative border consists of all itemsets that were candidates of the level-wise method which did not have enough support. That is, $\mathcal{NB}d(L_k) = C_k - L_k$ where C_k is the set of candidate k-itemsets, L_k is the set of large k-itemsets and $\mathcal{NB}d(L_k)$ is the set of k-itemsets in $\mathcal{NB}d(L)$. Therefore, $L_k \cup \mathcal{NB}d(L_k) = C_k$. The *apriori-gen* function uses only L_{k-1} to compute C_k.

Lemma 1 *All* 1*-itemsets should be present in* $L \cup \mathcal{NB}d(L)$.

Addition of new transactions

When new transactions are added to the database, an old large itemset could potentially become small in the updated database. Similarly, an old small itemset could potentially become large in the new database.

---

function negativeborder-gen(L)
    Split L into $L_1, L_2, \ldots, L_n$ where n is the size of the largest itemset in L
    forall $k = 1, 2, \ldots, n$ *do*
        compute C_{k+1} using *apriori-gen*(L_k)
    $L \cup \mathcal{NB}d(L) = \bigcup_{i=2,\ldots,n+1} C_k \cup I_1$ where I_1 is the set of 1-itemsets.

Figure 2: A high-level description of the negativeborder-gen function

In order to solve the update problem efficiently, we maintain the large itemset and the negative border along with their support count in the database. That is, for every $s \in L \cup \mathcal{NB}d(L)$, we maintain *s.count*. In the rest of this section, DB denotes the original database, db denotes the transactions that are newly added and $DB+$ denotes the updated database. Also L^{DB}, L^{db} and L^{DB+} denotes the large itemset and $\mathcal{NB}d(L^{DB})$, $\mathcal{NB}d(L^{db})$ and $\mathcal{NB}d(L^{DB+})$ denotes the negative border of the original database, increment database and the updated database respectively.

Lemma 2 *Let s be any itemset such that $s \notin L^{DB}$. Then $s \in L^{DB+}$ only if $s \in L^{db}$.*

Proof: Assume that there exists an itemset s such that $s \in L^{DB+}$, $s \notin L^{DB}$ and $s \notin L^{db}$. Let $t_{DB}(s)$ and $t_{db}(s)$ be the number of transactions in DB and db respectively containing the itemset s. Also let t_{DB} and t_{db} be the total number of transactions in DB and db respectively. Since $s \notin L^{DB}$ and $s \notin L^{db}$,

$$\frac{t_{DB}(s)}{t_{DB}} < minSupport \text{ and } \frac{t_{db}(s)}{t_{db}} < minSupport.$$

From these two equations, it can be shown that

$$\frac{t_{DB}(s) + t_{db}(s)}{t_{DB} + t_{db}} < minSupport$$

Therefore, $s \notin L^{DB+}$ which is a contradiction. □

Lemma 3 *Let s be an itemset such that $s \in \mathcal{NB}d(L)$. Then all possible subsets of s must be present in L.*

Proof: For a contradiction, let t be an itemset such that $t \subset s$ and $t \notin L$. By the definition of negative border, $\mathcal{NB}d(L)$ consists of the *minimal* itemsets not in L. Since $t \notin L$, s is not a minimal itemset not in L. Therefore s cannot be in $\mathcal{NB}d(L)$, which is a contradiction. □

Theorem 1 *Let s be an itemset such that $s \notin L^{DB} \cup \mathcal{NB}d(L^{DB})$ and $s \in L^{DB+}$. Then there exists an itemset t such that $t \subset s$, $t \in \mathcal{NB}d(L^{DB})$ and $t \in L^{DB+}$. That is, some subset of s has moved from $\mathcal{NB}d(L^{DB})$ to L^{DB+}.*

Proof: Since $s \in L^{DB+}$, all possible subsets of s should be in L^{DB+}. But all the subsets of s cannot be in L^{DB} because if that was the case, then s should be present in at least $\mathcal{NB}d(L^{DB})$ if not in L^{DB} itself. By our assumption, $s \notin L^{DB} \cup \mathcal{NB}d(L^{DB})$. Therefore, there exists an itemset t such that $t \subset s$ and $t \notin L^{DB}$. Now we have two cases.

Case i **:** $t \in \mathcal{NB}d(L^{DB})$.

In this case, $t \in L^{DB+}$ since $s \in L^{DB+}$ and $t \subset s$. Therefore, we have found a subset of s which has moved from $\mathcal{NB}d(L^{DB})$ to L^{DB+}.

Case ii **:** $t \notin \mathcal{NB}d(L^{DB})$.

That is, $t \notin L^{DB} \cup \mathcal{NB}d(L^{DB})$. But, we know that $t \in L^{DB+}$ since $s \in L^{DB+}$ and $t \subset s$. Therefore, $t \notin L^{DB} \cup \mathcal{NB}d(L^{DB})$ and $t \in L^{DB+}$ and hence we can apply the theorem recursively on t. Note that the size of t is less than the size of s since $t \subset s$.

When this is applied recursively, there are two possibilities. First is, for some subset of t, case i holds true in which case, there is a subset of t which has moved from $\mathcal{NB}d(L^{DB})$ to L^{DB+}, and hence the theorem is proved. Otherwise, t will finally become a 1-itemset. By Lemma 1, we know that all 1-itemsets are present in $L^{DB} \cup \mathcal{NB}d(L^{DB})$. Since $t \notin L^{DB}$, $t \in \mathcal{NB}d(L^{DB})$ which contradicts the assumption for case ii. That is, case ii is not possible if t is a 1-itemset. □

By theorem 1, if none of the itemsets move from the negative border to the large itemset, we do not need to scan the whole database. Even in cases where some itemsets move from the negative border to the large itemset, a complete database scan is required only if the negative border expands because, for all the itemsets in the negative border, we can derive the updated support count easily.

We maintain the support count for all itemsets in the large itemset and the negative border. First, we compute the large itemset in db using a level-wise algorithm like Apriori or Partition. Simultaneously we count the support for all itemsets in $L^{DB} \cup \mathcal{NB}d(L^{DB})$ in db. If an itemset $t \in L^{DB}$ does not have minimum support in $DB \cup db$, then t is removed from L^{DB}. This can be easily checked since we know the support count for t in DB and db. The change in L^{DB} could potentially change $\mathcal{NB}d(L^{DB})$ also. Therefore, we have to recompute the negative border using the *negativeborder-gen* function explained in subsection .

On the other hand there could be some new itemsets which become large in the updated database. Let s be an itemset which gets added to the large itemset of the updated database. By Lemma 2, we know that s has to be in L^{db}. We also know by theorem 1 that some subset of s must move from $\mathcal{NB}d(L^{DB})$ to L^{DB+}. For each itemset $s \in L^{db}$, we check if s gets the minimum support to move from $\mathcal{NB}d(L^{DB})$ to L^{DB+}. If none of the itemsets in $\mathcal{NB}d(L^{DB})$ gets the minimum support, no new itemsets will be added to L^{DB+}. If some itemsets in $\mathcal{NB}d(L^{DB})$ gets the minimum support move them to L^{DB+} and recompute the negative border. If $L^{DB+} \cup \mathcal{NB}d(L^{DB+}) \neq L^{DB} \cup \mathcal{NB}d(L^{DB})$, we have to find the negative border closure of L^{DB+} and scan the entire database(DB) once to find the updated large itemset and negative border. The negative border closure of L is found by repeatedly finding $L = L \cup \mathcal{NB}d(L)$ until L does not grow.

During the database scan, all the itemsets which are in the negative border closure that were not originally in $L \cup \mathcal{NB}d(L)$ are used as the candidate itemsets and

function Update-Large-Itemset(L^{DB}, $\mathcal{NB}d(L^{DB})$, db)

```
//DB and db denote the number of transactions in
  the original database and the increment database
  respectively.
Compute L^db
for each itemset s ∈ L^DB ∪ NBd(L^DB) do
  t_db(s) = number of transactions in db containing s
L^DB+ = φ
for each itemset s ∈ L^DB do
  if (t_DB(s) + t_db(s)) ≥ minsup * (DB + db) then
    L^DB+ = L^DB+ ∪ s
for each itemset s ∈ L^db do
  if s ∉ L^DB and s ∈ NBd(L^DB) and (t_DB(s) +
  t_db(s)) ≥ minsup * (DB + db) then
    L^DB+ = L^DB+ ∪ s
if L^DB ≠ L^DB+ then
  NBd(L^DB+) = negativeborder-gen(L^DB+)
else NBd(L^DB+) = NBd(L^DB)
if L^DB ∪ NBd(L^DB) ≠ L^DB+ ∪ NBd(L^DB+) then
  S = L^DB+
  repeat
    compute S = S ∪ NBd(S)
  until S does not grow
  L^DB+ = {x ∈ S | support(x) ≥ minsup}
  //support(x) is the support count of x in DB ∪ db
  NBd(L^DB+) = negativeborder-gen(L^DB+)
```

Figure 3: A high-level description of the Update-Large-Itemset function

their support count is computed. The candidate set can further be pruned by applying an optimization while finding the negative border closure. It can be observed that an itemset which is not large in the increment database (db) cannot get added to the updated set of large itemsets. Therefore, such itemsets can be pruned at each step of the negative border closure computation to get the pruned negative border closure. However, the support count of these pruned itemsets should also be found since they may potentially be in the updated negative border.

Deletion of existing transactions

Similar to the case where new transactions are added to the database, the large itemset and its negative border could potentially change when some existing transactions are deleted from the database. As in the former case, we maintain the large itemset and the negative border along with their support count in the database. Let $DB-$ denote the updated database and L^{DB-} and $\mathcal{NB}d(L^{DB-})$ denote its large itemset and negative border respectively.

Lemma 4 *Let s be an itemset such that $s \in L^{DB}$. Then $s \notin L^{DB-}$ only if $s \in L^{db}$. That is a large itemset s will become small only if $s \in L^{db}$.*

This lemma can be proved in the same way as lemma 2.

The algorithm to compute the large itemset and the negative border of $DB-$ is similar to the one in the case where new transactions are added to the database.

Experimental Results

We conducted a set of experiments to compare the performance of our incremental algorithm. The experiments were performed on a Sun SPARCstation 4 running SunOS 5.5. In this section, we report on the results of some of those experiments.

The experiments were performed on synthetic data generated using the same technique as in (Agrawal and Srikant 1994). The dataset used for the baseline experiment was T10.I4.D100K (Mean size of a transaction = 10, Mean size of maximal potentially large itemsets = 4, Number of transactions = 100 thousand). The increment database is created as follows: We generate 100 thousand transactions, of which $(100-d)$ thousand is used for the initial computation and d thousand is used as the increment, where d is the fractional size (in percentage) of the increment.

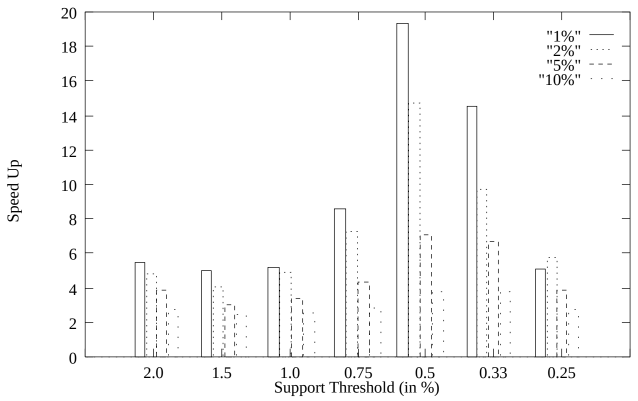

Figure 4: Performance Ratio

We compare the execution time of the incremental algorithm with respect to running Apriori on the whole data set. Figure 4 shows the speed up of the incremental algorithm over Apriori for different *minimum support* thresholds. We report the results for increment sizes of $1\%, 2\%, 5\%$ and 10%. From the graph, it can be seen that the incremental algorithm achieves speed up of about 3 to 20. The algorithm shows better speed up for medium support threshold than low and high support thresholds. At high support thresholds, the number of large itemsets is less and hence it is less costly to run Apriori on the whole database. At low support thresholds, the probability of the negative border expanding is higher and as a result the incremental algorithm may have to scan the whole database. Also, the speed up is higher for smaller increment sizes since the incremental algorithm needs to process less data.

Comparison with FUP

The framework of FUP (Cheung et al. 1996) is similar to that of Apriori and contains a number of iterations. Each iteration is associated with a complete scan of the whole database and in iteration k all the large k-itemsets are found. The candidate sets for iteration $k+1$ are generated based on the large itemsets found in iteration k. The speed up of FUP over Apriori can be mainly attributed to the reduction in the number of candidate itemsets. It uses the large itemset of the original database to filter and prune the candidate itemsets generated by Apriori. However, FUP may require $O(n)$ passes over the database where n is the size of the maximal large itemset.

The most important feature of our incremental updation algorithm is that the whole database is scanned only when required (and that too only once), thereby reducing the I/O requirements drastically. Computing the negative border closure may increase the size of the candidate set. However, a majority of those itemsets would have been present in the original negative border or large itemset. Only those itemsets which were not covered by the negative border need to be checked against the whole database. As a result, the size of the candidate set in the final scan could potentially be much smaller as compared to FUP.

Conclusions

We have presented an efficient, incremental updation algorithm for the maintenance of the association rules discovered by database mining. Our algorithm strives to reduce the I/O requirements for updating the set of large itemsets. This is achieved by maintaining the large itemsets and the negative border along with their support counts. The whole database is scanned only if required and that too just once. This incremental updation technique can be used in conjunction with any of the level-wise algorithms like Apriori and Partition. Further, our algorithm is applicable to addition as well as deletion of transactions.

References

R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM SIGMOD International Conference*, May 1993.

R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th VLDB, Santiago, Chile*, September 1994.

D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In *Proceedings of the 12th ICDE, New Orleans, Louisiana*, February 1996.

R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient Algorithms for Discovering Frequent Sets in Incremental Databases. In *Proceedings of the 1997 SIGMOD Workshop on DMKD, Tucson, Arizona*, May 1997.

H. Mannila and H. Toivonen. On an Algorithm for Finding all Interesting Sentences. In *Cybernetics and Systems, Volume II, The 13th European Meeting on Cybernetics and Systems Research, Vienna, Austria*, April 1996.

A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of the 21st VLDB, Zurich, Switzerland*, September 1995.

Hannu Toivonen. Sampling Large Databases for Association Rules. In *Proceedings of the 22nd VLDB, Mumbai(Bombay), India*, September 1996.