A General Incremental Technique for Maintaining Discovered Association Rules

David W. Cheung *S.D. Lee* *Benjamin Kao*

Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong

{*dcheung,sdlee,kao*}*@cs.hku.hk*

Abstract

A more general incremental updating technique is developed for maintaining the association rules discovered in a database in the cases including insertion, deletion, and modification of transactions in the database. A previously proposed algorithm FUP can only handle the maintenance problem in the case of insertion. The proposed algorithm FUP2 makes use of the previous mining result to cut down the cost of finding the new rules in an updated database. In the insertion only case, FUP2 is equivalent to FUP. In the deletion only case, FUP2 is a complementary algorithm of FUP which is very efficient when the deleted transactions is a small part of the database, which is the most applicable case. In the general case, FUP2 can efficiently update the discovered rules when new transactions are added to a transaction database, and obsolete transactions are removed from it. The proposed algorithm has been implemented and its performance is studied and compared with the best algorithms for mining association rules studied so far. The study shows that the new incremental algorithm is significantly faster than the traditional approach of mining the whole updated database.

Keywords: Association Rules, Data Mining, Knowledge Discovery, Large Databases, Maintenance.

1 Introduction

In recent years, *data mining* has attracted much attention in database research. This is due to its wide applicability in many areas, including the retail industry and the finance sector [6]. The availability of automated tools has enabled the collection of large amount of data. These large databases contain information that is potentially useful for making market strategies and financial forecasts. Data mining is the task to find out such useful information from large databases. The information includes association rules, characteristic rules, classification rules, generalized relations, discriminant rules, etc. [9]

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1–4, 1997.

Of the various data mining problems, mining of association rules is an important one [3]. A classical example is about the retail industry. Typically, a record in the sales data describes all the items that are bought in a single transaction, together with other information such as the transaction time, customer-id, etc. Mining association rules from such a database is to find out, from the huge amount of past transactions, all the rules like "A customer who buys item X and item Y is also *likely* to buy item Z in the same transaction", where X, Y and Z are initially unknown. Such rules are very useful for marketers to develop and implement customized marketing programs and strategies.

Recently, many interesting works have been published in association rules mining including mining of quantitative association rules and multi-level association rules, and parallel and distributed mining of association rules [2, 3, 4, 5, 8, 10, 12, 13, 14].

A feature of data mining problems is that in order to have stable and reliable results, a giant amount (often of the order of gigabytes) of data has to be collected and analyzed. The large amount of input data and mining results poses a maintenance problem. While new transactions are being appended to a database and obsolete ones are being removed, rules or patterns already discovered also have to be updated. In this paper we examine the problem of maintaining discovered association rules. We propose a new incremental algorithm FUP2 in this paper which can efficiently handle all the update cases including insertion, deletion and modification of transactions.

Previous works

The problem of mining association rules was first introduced in [2]. In that paper it was shown that the problem can be decomposed into two subproblems:

1. Find out all *large itemsets*, which are the sets of items that are contained in a sufficiently large number of transactions, with respect to a threshold *minimum support*.
2. From the set of large itemsets found, find out all the association rules that have a confidence value exceeding a threshold *minimum confidence*.

Since the solution to the second subproblem is straightforward [3], major research efforts have been spent on the first subproblem. Among the algorithms proposed to solve the first subproblem efficiently, the Apriori [3] and DHP [12] algorithms are the most successful. The Apriori algorithm finds out the large itemsets iteratively. In each iteration, it generates a number of candidate large itemsets and then verify them by scanning the database. The key success is the use of the `apriori_gen` function to generate a small number of candidate itemsets. DHP improves over Apriori by further reducing the number of candidate itemsets using a hashing technique.

While both Apriori and DHP efficiently discover association rules from a database, the rules maintenance problem is not addressed. The problem of maintaining association rules is first studied in [4]. That paper proposes the FUP algorithm, which can update the association rules in a database when new transactions are *added* to the database. It is based on the framework of Apriori and it also finds new large itemsets iteratively. The idea is to store the counts of all the large itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the algorithm can generate a very small number of candidate new large itemsets. The overall count of these candidate itemsets are then obtained by scanning the original database. Consequently, all *new* large itemsets are found. The FUP algorithm is very efficient. However, the algorithm does not handle the case of deleting transactions from the database. The modification of transactions is not addressed, either.

In this paper we propose a new algorithm, called FUP2, that can update the existing association rules when transactions are added and deleted from the database. It is a generalization of the FUP algorithm [4]. Like FUP, FUP2 makes use of the previous mining result to cut down the amount of work that has to be done to discover the new set of rules.

The remaining of this paper is organized as follows. Section 2 gives a detailed description of the problem. The new algorithm is described in Section 3. A performance study of FUP2 is presented in Section 4. We discuss some implementation issues of FUP2 in Section 5 and this paper is ended with a conclusion in Section 6.

2 Problem Description

2.1 Mining of association rules

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. Let D be a database of transactions, where each transaction T is a set of items such that $T \subseteq I$. For a given *itemset* $X \subseteq I$ and a given transaction T, we say that T *contains* X if and only if $X \subseteq T$. The *support count* of an itemset X is defined to be σ_X = the number of transactions in D that contain X. We say that an itemset X is *large*, with respect to a *support threshold* of $s\%$, if $\sigma_X \geq |D| \times s\%$, where $|D|$ is the number of transactions in the database D. An *association rule* is an implication of the form "$X \Rightarrow Y$", where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$. The association rule $X \Rightarrow Y$ is said to hold in the database D with *confidence* $c\%$ if no less than $c\%$ of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* $s\%$ in D if $\sigma_{X \cup Y} = |D| \times s\%$. For a given pair of confidence and support thresholds, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. This problem can be reduced to the problem of finding all large itemsets for the same support threshold [2].

Thus, if $s\%$ is the given support threshold, the mining problem is reduced to the problem of finding the set $L = \{X | X \subseteq I \wedge \sigma_X \geq |D| \times s\%\}$. For the convenience of subsequent discussions, we call an itemset that contains exactly k items a k-*itemset*. We use the symbol L_k to denote the set of all k-itemsets in L.

2.2 Update of association rules

After some update activities, old transactions are deleted from the database D and new transactions are added. We can treat the modification of existing transactions as deletion followed by insertion. Let Δ^- be the set of deleted transactions and Δ^+ be the set of newly added transactions. We assume, naturally, that $\Delta^- \subseteq D$. Denote the updated database by D'. Note that $D' = (D - \Delta^-) \cup \Delta^+$. We denote the set of unchanged transactions by $D^- = D - \Delta^- = D' - \Delta^+$.

database	support count of itemset X	Large k-itemsets
Δ^+	δ_X^+	—
D^-	—	—
Δ^-	δ_X^-	—
$D = \Delta^- \cup D^-$	σ_X	L_k
$D' = D^- \cup \Delta^+$	σ'_X	L'_k

Table 1: Definitions of several symbols

As defined in the previous subsection, σ_X is the support count of itemset X in the original database D. The set of large itemsets in D is L

and L_k is the set of k-itemsets in L. Define σ'_X to be the new support count of an itemset X in the updated database D', and L' to be the set of large itemsets in D'. L'_k is the set of k-itemsets in L'. We further define δ^+_X to be the support count of itemset X in the database Δ^+ and δ^-_X to be that of Δ^-. These definitions are summarized in Table 1. We define $\delta_X = \delta^+_X - \delta^-_X$ which is the change of support count of itemset X as a result of the update activities. Thus, we have:

Lemma 1 $\sigma'_X = \sigma_X + \delta^+_X - \delta^-_X = \sigma_X + \delta_X$

Proof. By definition. ∎

As the result of a previous mining on the old database D, we have already found L and σ_X $\forall X \in L$. Thus, the update problem is to find L' and σ'_X $\forall X \in L'$ efficiently, given the knowledge of D, D', Δ^-, D^-, Δ^+, L and σ_X $\forall X \in L$.

3 The FUP2 algorithm

In this section, we introduce the FUP2 algorithm step by step. We first focus on the special case for transaction deletion only ($\Delta^+ = \emptyset$). This special case can be considered as a complement of the FUP algorithm [4], which handles transaction insertion only. Next, we generalize the deletion algorithm to handle the case for transaction deletion as well as insertion ($|\Delta^+| \geq 0$).

3.1 The special case for transaction deletion only ($\Delta^+ = \emptyset$)

For the delete-only case, we have $\Delta^+ = \emptyset$ and hence $D' = D^- \cup \emptyset = D^- = D - \Delta^-$. Note also that $\delta^+_X = 0 \; \forall \; X \subseteq I$.

To discover the large itemsets in the updated database D', the FUP2 algorithm executes iteratively. In the k-th iteration, all the large k-itemsets in D' are found as follows. As in Apriori [3], we form a set of *candidates* C_k which is a superset of L'_k. In the first iteration, C_1 is exactly the set I. In subsequent iterations, C_k is calculated from L'_{k-1}, the large itemsets found in the previous iteration, using the same `apriori_gen` function as in Apriori [3]. All the itemsets in L'_k are guaranteed to be contained in C_k.

Next, we use the old large k-itemsets L_k from the previous mining result to divide the candidate set C_k into 2 parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. In words, P_k (Q_k) is the set of candidate itemsets that are *previously* large (small) with respect to D. Again, our goal is to select those itemsets that are *currently* large (w.r.t. D'). We treat the candidates in these two partitions separately.

With this partitioning, for all candidates $X \in P_k$, we already know its support count σ_X from the previous mining results. We find out δ^-_X by scanning Δ^-. Then, we can obtain the updated support count σ'_X using Lemma 1. Thus, a candidate X from P_k goes to L'_k if and only if $\sigma'_X \geq |D'| \times s\%$.

For the candidates in Q_k, we only know that they were not large in the original database D. We do not know their support counts. However, since they were not large, we know that $\sigma_X < |D| \times s\% \; \forall \; X \in Q_k$. We can make use of this information to tell which candidates from Q_k *may* be large and which *will not*.

Lemma 2 *If* $X \notin L$ *and* $\delta^+_X = 0$*, then* $\sigma'_X < |D'| \times s\%$ *if* $\delta^-_X \geq |\Delta^-| \times s\%$.

Proof. Since $X \notin L$, we have $\sigma_X < |D| \times s\%$. Hence, $\sigma'_X = \sigma_X - \delta^-_X + \delta^+_X < |D| \times s\% - |\Delta^-| \times s\% + 0 = (|D| - |\Delta^-|) \times s\% = |D'| \times s\%$ ∎

That is to say, for each candidate in Q_k, if it is *large* in Δ^-, then it cannot be *large* in L'_k. We first scan Δ^- and obtain δ^-_X for each $X \in Q_k$. Then, we delete those candidates for which $\delta^-_X \geq |\Delta^-| \times s\%$, thus leaving in Q_k those that are *small* in Δ^-. Note that we are not checking all *small* itemsets in Δ^-. We are only checking for those small itemsets of Δ^- that are in Q_k. The number of such itemsets is not large, as $Q_k \subseteq C_k$. For the candidates X that remain in Q_k, we scan D^- to obtain their new support counts σ'_X. Finally, we add to L'_k those candidates X from Q_k for which $\sigma'_X \geq |D'| \times s\%$.

Thus, we have discovered which candidates from P_k and Q_k are large and put them into L'_k. Moreover, we have also found out σ'_X for each $X \in L'_k$. We have completed one iteration. In the subsequent iterations, large itemsets of larger sizes are discovered. The iterations go on until either $C_k = \emptyset$ or $|L'_k| < k + 1$ for some k. The steps of the kth-iteration are summarized as follows.

1. Obtain a candidate set C_k of itemsets. Halt if $C_k = \emptyset$.
2. Partition C_k into P_k and Q_k, where $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$.
3. Scan Δ^- to find out δ^-_X for each $X \in P_k \cup Q_k$.
4. For each $X \in P_k$, Calculate σ'_X.
5. Delete from Q_k those candidates X where $\delta^-_X \geq |\Delta^-| \times s\%$. (Application of Lemma 2.)
6. Scan D^- to find out σ'_X of the remaining candidates $X \in Q_k$.
7. Add to L'_k those candidates X from $P_k \cup Q_k$ for which $\sigma'_X \geq |D'| \times s\%$.
8. Halt if $|L'_k| < k + 1$.

As found in previous works, the speed of the Apriori algorithm depends very much on the size of the candidate set. To improve performance, our FUP2 algorithm makes use of the information L_k and σ_X $\forall$ $X \in L_k$ to reduce the size of the candidate set. It scans the updated database D' with a candidate set ($Q_k \subseteq C_k$) which is significantly smaller than that (C_k) of Apriori (see Section 4). The algorithm, however, does scan the deleted transactions Δ^- with the same number of candidates as Apriori does. We can reduce this number of candidates by the following optimization.

3.2 An optimization

Observe that in step 5 of the FUP2 algorithm, we are removing from Q_k those candidates which are *large* in Δ^-. Those that are *small* in Δ^- always remain. Now, there is a way to determine whether a candidate itemset X is *small* prior to knowing δ_X^-, thus saving the work of finding the value of δ_X^-.

Lemma 3 *For any itemsets X and Y such that $X \subseteq Y$, $\delta_X^- \geq \delta_Y^-$*

Proof. Any deleted transaction that contains Y must also contain X. ∎

Corollary 1 *All supersets of a* small *1-itemset are* small.

Hence, if we remember which 1-itemsets are small in Δ^- during the first iteration, then in the subsequent iterations, we can quickly determine if a candidate from Q_k is small in Δ^- without finding its support count in Δ^-. Thus, we can optimize the above algorithm by adding the following steps:

2.5 For each candidate $X \in Q_k$, if X contains any item which is a *small* 1-itemset in Δ^-, move it to the set R_k. All the candidates so moved to R_k are those that are small in Δ^- by corollary 1.

5.5 Move all candidates from R_k to Q_k.

This modification significantly reduces the number of candidates during the scan of Δ^-. The only additional cost is to remember the set of *small* 1-itemset in the first iteration. This requires extra memory space of size linear to $|I|$. The extra CPU time required is negligible, since we have to find δ_X^- for all 1-itemsets X anyway. So, the additional cost of this optimization is relatively inexpensive. The number of the candidates for scanning D^- is not affected by this optimization, but the number of candidates for scanning Δ^- is significantly reduced. So, this optimization speeds up the performance of the algorithm at negligible cost.

Let us illustrate this special case of the FUP2 algorithm with the example shown in Table 2. The original database D contains 5 transactions and we set the support threshold to 25%. So, itemsets with a support count σ_X no less than $5 \times 25\% = 1.25$ are large. The large itemsets in L are shown in the same table. For convenience, we write XYZ for the itemset $\{X, Y, Z\}$ when no ambiguity arises. One transaction is deleted, leaving 4 transactions in the final database D'. Now, let us apply the FUP2 algorithm to see how L' is generated.

Transactions: $(I = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\})$

D	Δ^-	$\mathcal{A}$	$\mathcal{B}$			$\mathcal{E}$	
D	D^-	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$			D'
D	D^-	$\mathcal{A}$			$\mathcal{D}$		D'
D	D^-		$\mathcal{B}$		$\mathcal{D}$		D'
D	D^-			$\mathcal{C}$	$\mathcal{D}$		D'

Large itemsets (support threshold $s = 25\%$) in $D = \Delta^- \cup D^-$:

Itemsets(X)	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{AB}$
σ_X	3	3	2	3	2

Table 2: An example for $\Delta^+ = \emptyset$

In the first iteration, $C_1 = I = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$. Of these candidates, only $\mathcal{E}$ was not large in D. So, after partitioning, $P_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$ and $Q_1 = \{\mathcal{E}\}$. Next, we scan Δ^- and update the support counts of the candidates in P_1. Only $\mathcal{A}$ and $\mathcal{B}$ occur in Δ^-. So, the counts are updated as $\sigma'_{\mathcal{A}} = \sigma'_{\mathcal{B}} = \sigma'_{\mathcal{C}} = 2$, $\sigma'_{\mathcal{D}} = 3$. In the same scan of Δ^-, we find out that $\delta_{\mathcal{E}}^- = 1 \geq 0.25 = |\Delta^-| \times 25\%$. Hence, $\mathcal{E}$ is large in Δ^-, and it was *small* in D. It cannot be *large* in D' by Lemma 2. So, it is removed from Q_1. This leaves Q_1 empty, hence we need *not* scan D^- at all in this iteration. Since all the remaining candidates in $P_1 \cup Q_1$ have a support count in D^- no less than $|D'| \times 25\% = 1$, they all fall into L'_1. We remember that $\mathcal{C}$ and $\mathcal{D}$ are small in Δ^- for optimization.

In the second iteration, we first obtain C_2 by applying the `apriori_gen` function on L'_1. This gives $C_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$, of which only $\mathcal{AB}$ was large in D. So, the partitioning results in $P_2 = \{\mathcal{AB}\}$ and $Q_2 = \{\mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$. Next, we scan Δ^- and update the count $\sigma'_{\mathcal{AB}} = \sigma_{\mathcal{AB}} - \delta_{\mathcal{AB}}^- = 2 - 1 = 1$. All the candidates in Q_2 contain either item $\mathcal{C}$ or $\mathcal{D}$, which are small in Δ^-. So, we know that all the candidates in Q_2 are definitely small in Δ^- (corollary 1) and hence potentially large in D' (Lemma 2). There is no need to find out δ_X^- for these candidates $X \in Q_2$. So, the next job is to scan D^- to obtain σ'_X for the candidates in Q_2. This gives $\sigma'_{\mathcal{AC}} = \sigma'_{\mathcal{AD}} = \sigma'_{\mathcal{BC}} = \sigma'_{\mathcal{BD}} = \sigma'_{\mathcal{CD}} = 1$. Consequently, $\mathcal{AB}$, $\mathcal{AC}$, $\mathcal{AD}$, $\mathcal{BC}$, $\mathcal{BD}$, $\mathcal{CD}$ are large in D' and hence are included in L'_2.

In the third iteration, `apriori_gen` gives a candidate set $C_3 = \{\mathcal{ABC}, \mathcal{ABD}, \mathcal{ACD}, \mathcal{BCD}\}$. None of these candidates were large. So, $P_3 = \emptyset$ and $Q_3 = C_3$. Since all the candidates contain item $\mathcal{C}$ or item $\mathcal{D}$, we know that they are all small in Δ^- without having to find out their support counts in Δ^-. So, there is no need to scan Δ^- in this iteration! We only have to scan D^- to obtain the support counts in D' for the candidates. The results are $\sigma'_{\mathcal{ABC}} = 1$, $\sigma'_{\mathcal{ABD}} = \sigma'_{\mathcal{ACD}} = \sigma'_{\mathcal{BCD}} = 0$. Only $\mathcal{ABC}$ goes to L'_3.

There is only 1 large itemset found in this iteration. This is insufficient to generate any candidates in the next iteration. Hence, the algorithm stops after 3 iterations.

Table 3 compares the size of candidates when Apriori is applied on D' and when FUP2 is employed. While Apriori scans D^- three times, with a total of 15 candidate itemsets, FUP2 scans D^- only twice, with a total of 9 candidates only. Although FUP2 has to scan Δ^- with 6 candidate sets, the time spent on this is insignificant, as $|D^-| \gg |\Delta^-|$ in most practical applications. In our example, FUP2 has reduced the number of candidates for scanning D^- by $\frac{15-9}{15} = 40\%$—a significant improvement.

Iteration	**Apriori**	**FUP2**	
	scan D^-	scan D^-	scan Δ^-
1	5	0	5
2	6	5	1
3	4	4	0
Total	15	9	6

Table 3: Size of candidates for the example

3.3 The general case for transaction deletion and insertion

Now, we extend the algorithm introduced in the previous subsection to handle the general case for transaction insertion as well as deletion. We no longer assume that $\Delta^+ = \emptyset$. So, δ_X^+ may be positive for any $X \subseteq I$. Consequently, Lemma 2 cannot be applied and corollary 1 is no longer useful.

As before, we find out L'_k and $\sigma'_X \ \forall \ X \in L'_k$ in the kth-iteration. In each iteration, we first form a candidate set C_k and then partition it into two parts P_k and Q_k as before. Again for each candidate $X \in P_k$, we know σ_X from the previous mining result. So, we only have to scan Δ^- and Δ^+ to update the support count for the candidates in P_k. In the FUP2 algorithm, we choose to scan Δ^- to find δ_X^- for each candidate X first. As we scan Δ^-, we can deduct the support count at the same time, and remove a candidate from P_k as soon as its support count drops below $|D'| \times s\% - |\Delta^+|$. This is because such a candidate has no hope to have $\sigma'_X \geq |D'| \times s\%$, as $\delta_X^+ \leq |\Delta^+|$. Next, we scan Δ^+ to find δ_X^+ for each candidate X that remains in P_k. Finally, we calculate σ'_X for each candidate in P_k using Lemma 1, and add those with $\sigma'_X \geq |D'| \times s\%$ to L'_k.

For the candidates $X \in Q_k$, again we do not know σ_X, but we know that $\sigma_X < |D| \times s\%$. By a generalization of Lemma 2, we are able to prune some candidates from Q_k without knowing their counts in D^-.

Lemma 4 *If $X \notin L$ and $\delta_X = \delta_X^+ - \delta_X^- \leq (|\Delta^+| - |\Delta^-|) \times s\%$, then $X \notin L'$.*

Proof. If $X \notin L$, then $\sigma_X < |D| \times s\%$. Hence, $\sigma'_X = \sigma_X + (\delta_X^+ - \delta_X^-) < |D| \times s\% + (|\Delta^+| - |\Delta^-|) \times s\% = (|D| + |\Delta^+| - |\Delta^-|) \times s\% = |D'| \times s\%$. Thus $X \notin L'$ by the definition of L'. ∎

So, for each candidate X in Q_k, we obtain the values of δ_X^+ and δ_X^- during the scans of Δ^+ and Δ^-. Then, we calculate δ_X and remove those with $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$, because Lemma 4 tells us that they will not fall into L'_k. For the remaining candidate in Q_k, we scan D^- and obtain their support counts in D^-. Adding this count to δ_X^+ gives σ'_X. We add those candidates with $\sigma'_X \geq |D'| \times s\%$ from Q_k to L'_k. This finishes the iteration. This algorithm scans Δ^- and Δ^+ with a candidate set of size $|P_k \cup Q_k| = |C_k|$, the same as Apriori does. However, it scans D^- with a candidate set much smaller than the initial size of $Q_k \subseteq C_k$, thereby saving time, assuming that $|D^-| \gg |\Delta^-|$. The algorithm saves a lot of time when compared to Apriori.

3.4 Optimizations

In the algorithm described in the above paragraphs, the databases Δ^- and Δ^+ have to be scanned with a candidate set of size $|C_k|$. As an improvement to the algorithm, we can reduce this size by finding bounds on the values of δ_X^+ and δ_X^- for each candidate X prior to the scans of Δ^- and Δ^+. The idea comes from an extension of Lemma 3.

Lemma 5 *For any itemsets X and Y such that $X \subseteq Y$, $\delta_X^- \geq \delta_Y^-$ and $\delta_X^+ \geq \delta_Y^+$.*

Proof. Any transaction in Δ^- that contains Y also contains X, $\forall X \subseteq Y$. The same is true for transactions in Δ^+. ∎

Using this lemma, at the k-th iteration ($k \geq 2$), we can obtain an upper bound b_Y^- for δ_Y^- of candidate Y before scanning Δ^-. The bound is taken to be the minimum of δ_X^- for all $X \subset Y \ \wedge \ |X| = |Y| - 1$. Note that since Y is a candidate generated by `apriori_gen` [3], all its size-$(k-1)$ subsets X must be in L'_{k-1} and hence C_{k-1}; thus δ_X^- has been found in the previous iteration. A bound b_Y^+ can be similarly obtained for δ_Y^+ for each candidate Y before scanning Δ^+. Lower bounds for δ_Y^+ and δ_Y^- are zero, of course.

Now, before we scan Δ^-, we do not know the values of δ_X^- and δ_X^+ for each candidate X. We cannot apply Lemma 4 directly. However, combining lemmas 4 and 5, we can do some pruning at this stage: For each candidate X in Q_k, if $b_X^+ \leq (|\Delta^+| - |\Delta^-|) \times s\%$, then X cannot be in L'. So, we can remove such X from Q_k. Similarly, we may use the bound b_X^+ to remove the candidates X in P_k which satisfy $\sigma_X + b_X^+ < |D'| \times s\%$. This reduces the number of candidates before scanning Δ^- at a negligible cost.

After scanning Δ^- and before scanning Δ^+, we know the value of δ_X^-, but not δ_X^+ for a candidate X. Combining lemmas 4 and 5 gives us the following pruning: Delete from Q_k those candidate for which $b_X^+ - \delta_X^- \leq (|\Delta^+| - |\Delta^-|) \times s\%$. For the candidates X in P_k, those satisfying $\sigma_X + b_X^+ - \delta_X^- < |D'| \times s\%$ are deleted. Thus, the number of candidates is reduced at a negligible cost before scanning Δ^+.

We still have not made use of the bound b_X^-. It is employed in the following optimization which corresponds to the optimization introduced in Section 3.2. We note that for a candidate X in Q_k, we actually do not need to find δ_X^-, since the deleted transactions contribute nothing to the final support count σ'_X. However, the value of δ_X^- helps us to remove some candidates from Q_k before scanning D^-. So, it helps to improve performance. For a candidate X satisfying $b_X^+ - b_X^- \geq (|\Delta^+| - |\Delta^-|) \times s\%$, whatever the value of δ_X^- be, we have $b_X^+ - \delta_X^- \geq b_X^+ - b_X^- \geq (|\Delta^+| - |\Delta^-|) \times s\%$. Thus, we do not need to find δ_X^- for these candidates. As in Section 3.2, we remove such candidates to the set R_k before we scan Δ^-, thus reducing the size of the candidates in this scan.

A candidate X in R_k may be finally found to be in L'_k. However, δ_X^- is not available for such a candidate. This causes troubles in the calculation of b_X^- for the candidates in the *next* iteration. As a remedy, we assign b_X^- to δ_X^- for all candidates X in R_k. The bounds so calculated will still be valid, though not optimal. In the scan of Δ^+, we cannot directly apply Lemma 4 to the candidates in R_k directly. We can only prune out those candidates X from R_k for which $\delta_X^+ \leq (|\Delta^+| - |\Delta^-|) \times s\%$.

In the deletion-only case (Section 3.2), we introduced the set R_k to optimize the algorithm without paying much cost. This is not true for the general case. Although R_k reduces the candidate set for the scan of Δ^-, it also causes the candidate set in the scan of D^- to be larger. Moreover, a candidate X that gets moved to R_k will not have its count (δ_X^-) in Δ^- tallied. If we want to apply Lemma 4 to test if X can be ignored in the scan of D^-, only a trivial lower bound (zero) of δ_X^- is used. Therefore, the test and thus the pruning is less effective. The tradeoffs of whether to use R_k is thus on the amount of work saved in scanning Δ^- and the effectiveness of the pruning (Lemma 4). Naturally, if $|\Delta^-|$ is large, using R_k can save much work. Our algorithm therefore applies R_k only when $|\Delta^-| > |\Delta^+|$. Here is the final version of our FUP2 algorithm, for iteration k where $k \geq 2$. For the first iteration, set $C_1 = I$, $b_X^+ = |\Delta^+|$ and $b_X^- = |\Delta^-|$.

1. Obtain a candidate set C_k of itemsets. Halt if $C_k = \emptyset$.
2. Calculate b_X^+ for each $X \in C_k$.
3. Partition C_k into P_k and Q_k.
4. For each $X \in P_k$, remove it if $\sigma_X + b_X^+ < |D'| \times s\%$.
5. For each $X \in Q_k$, remove it if $b_X^+ \leq (|\Delta^+| - |\Delta^-|) \times s\%$.
6. If $|\Delta^-| \leq |\Delta^+|$, let $R_k = \emptyset$. Otherwise, calculate b_X^- for each $X \in Q_k$ and if $b_X^+ - b_X^- \geq (|\Delta^+| - |\Delta^-|) \times s\%$, move it to R_k and assign b_X^- to δ_X^-.
7. Scan Δ^- to find out δ_X^- for each $X \in P_k \cup Q_k$.
8. Delete from P_k those candidates X where $\sigma_X + b_X^+ - \delta_X^- < |D'| \times s\%$.
9. Delete from Q_k those candidates with $b_X^+ - \delta_X^- \leq (|\Delta^+| - |\Delta^-|) \times s\%$.
10. Scan Δ^+ to find δ_X^+ for each $X \in P_k \cup Q_k \cup R_k$.
11. For each candidate $X \in P_k$, calculate σ'_X.
12. For each candidate $X \in Q_k$, delete X if $\delta_X^+ - \delta_X^- \leq (|\Delta^+| - |\Delta^-|) \times s\%$.
13. For each candidate $X \in R_k$, delete X if $\delta_X^+ \leq (|\Delta^+| - |\Delta^-|) \times s\%$.
14. Scan D^- and get the count of each $X \in Q_k \cup R_k$. Then, add this count to δ_X^+ to get σ'_X.
15. Add to L'_k those candidates X from $P_k \cup Q_k \cup R_k$ where $\sigma'_X \geq |D'| \times s\%$.
16. Halt if $|L'_k| < k + 1$.

It is interesting to note that this algorithm reduces to Apriori [3] if we set $\Delta^- = D^- = \emptyset$, FUP [4] if we set $\Delta^- = \emptyset$ and the transaction deletion algorithm in Section 3.2 if we set $\Delta^+ = \emptyset$. So, it is a generalization of these three algorithms.

A further improvement can be made by applying the DHP technique [12]. The technique can be introduced into the FUP2 algorithm to hash the counts of itemsets in D'. This brings the benefits of the DHP algorithm into the FUP2 algorithm immediately. We call this DHP-enhanced update algorithm FUP2$_\mathcal{H}$, to distinguish it from FUP2.[1]

Let us illustrate this final FUP2 algorithm with the example in Table 4. This example is the same as the previous one, except that we have Δ^+ containing one transaction $\{\mathcal{C}, \mathcal{D}\}$ this time. Large itemsets and their counts in D' are shown in the same table.

In the first iteration, we have $C_1 = I = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$, $P_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$, $Q_1 = \{\mathcal{E}\}$ and $R_1 = \emptyset$. Note that for this iteration, $b_X^+ = |\Delta^+| = 1$ and $b_X^- = |\Delta^-| = 1$ for all $X \subseteq I$. Next, Δ^- is scanned and we find $\delta_\mathcal{A}^- = \delta_\mathcal{B}^- = \delta_\mathcal{E}^- = 1$ and $\delta_\mathcal{C}^- = \delta_\mathcal{D}^- = 0$. After pruning (step 9), $Q_1 = \emptyset$. Then, Δ^+ is scanned and we have $\delta_\mathcal{A}^+ = \delta_\mathcal{B}^+ = 0$; $\delta_\mathcal{C}^+ = \delta_\mathcal{D}^+ = 1$. Since both Q_k and R_k are now empty, steps 12–14 can be skipped. D^- need not be scanned in this iteration. Finally, $\sigma'_\mathcal{A} = \sigma'_\mathcal{B} = 2$; $\sigma'_\mathcal{C} = 3$ and $\sigma'_\mathcal{D} = 4$. All of them are large in D'. So, $L'_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}\}$.

[1] We remark that the DHP algorithm requires extra memory to store a big hash table. Algorithm FUP2$_\mathcal{H}$ should therefore be applied only when memory is plentiful.

Transactions: ($I = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$)

		Transactions	
D	Δ^-	$\mathcal{A}\ \mathcal{B}\ \mathcal{E}$	
D	D^-	$\mathcal{A}\ \mathcal{B}\ \mathcal{C}$	D'
D	D^-	$\mathcal{A}\ \mathcal{D}$	D'
D	D^-	$\mathcal{B}\ \mathcal{D}$	D'
D	D^-	$\mathcal{C}\ \mathcal{D}$	D'
	Δ^+	$\mathcal{C}\ \mathcal{D}$	D'

Large itemsets (support threshold $s = 25\%$) in $D' = D^- \cup \Delta^+$:

Itemsets(X)	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{CD}$
σ'_X	2	2	3	4	2

Table 4: An example for $|\Delta^+| > 0$

In the second iteration, $C_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$, $Q_2 = \{\mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$ and $P_2 = \{\mathcal{AB}\}$. Since $b^+_{\mathcal{AC}} = b^+_{\mathcal{AD}} = b^+_{\mathcal{BC}} = b^+_{\mathcal{BD}} = 0$ (because $\delta^+_{\mathcal{A}} = \delta^+_{\mathcal{B}} = 0$), the corresponding candidates are removed from Q_2 in step 5, leaving $Q_2 = \{\mathcal{CD}\}$. Next, we scan Δ^- and obtain $\delta^-_{\mathcal{AB}} = 1$, $\delta^-_{\mathcal{CD}} = 0$. Since $b^+_{\mathcal{AB}} = 0$, $\mathcal{AB}$ is removed from P_2 in step 8, leaving $P_2 = \emptyset$. Then, Δ^+ is scanned to get $\delta^+_{\mathcal{CD}} = 1$, followed by the scan of D^-. Finally $\sigma'_{\mathcal{CD}}$ is found to be 2, enough for $\mathcal{CD}$ to be large. Thus $L'_2 = \{\mathcal{CD}\}$. This is the last iteration, since $|L'_2| = 1 < 3$ is insufficient to generate a C_3 in the third iteration.

Hence, we find that in the updated database D', $L' = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{CD}\}$. The large itemset $\mathcal{AB} \in L$ is now obsolete and the new large itemset $\mathcal{CD}$ is added to L'. Note that FUP2 scans D^- only once, to obtain the count of $\mathcal{CD}$ in the second iteration. If we apply the Apriori algorithm on D' instead, we will have to scan D^- twice, with 11 candidates from C_1 and C_2. So, FUP2 reduces the candidate size by $\frac{11-1}{11} = 91\%$, a very significant improvement over Apriori.

4 Performance Analysis

To assess the performance of our new algorithms, Apriori, DHP, FUP2 and FUP2$\mathcal{H}$ are implemented on an RS/6000 workstation (model 410) running AIX. Several experiments are conducted to compare their performance.

4.1 Generation of synthetic data

In the experiments, we used synthetic data as the input databases to the algorithms. The data are generated using the same technique as introduced in [3] and modified in [12]. Readers are referred to these papers for a detailed description. Table 5 gives a list of the parameters used in the data generation method. To model the change of association rules as a result of inserting and deleting transactions, we slightly modified the data generation method as follows.

$\|\Delta^-\|$	number of deleted transactions
$\|D^-\|$	number of unchanged transactions
$\|\Delta^+\|$	number of added transactions
$\|T\|$	mean size of transactions
$\|\mathcal{I}\|$	mean size of potentially large itemsets
$\|\mathcal{L}\|$	number of potentially large itemsets
N	number of items

Table 5: Parameters for data generation

We split the data generation procedure into 2 steps. In the first step, a set $\mathcal{L}$ of potentially large itemsets is generated. In the second step, a subset of $\mathcal{L}$ is used to generate the database transactions. To model a change of association rules, we choose two integers p and q in the range from zero to $|\mathcal{L}|$, such that $p+q \geq |\mathcal{L}|$.[2] We use the first p potentially large itemsets from $\mathcal{L}$ to generate Δ^- and the last q potentially large itemsets from $\mathcal{L}$ to generate Δ^+. D^- is generated from the whole $\mathcal{L}$. As a result, the first p potentially large itemsets in $\mathcal{L}$ have a higher tendency to be large in $D = \Delta^- \cup D^-$ than in $D' = D^- \cup \Delta^+$. They correspond to large itemsets that turn *obsolete* due to the updates. Similarly, the last q potentially large itemsets have a higher tendency to be large in D' than in D. They correspond to *new* large itemsets in the updated database. The middle $p + q - |\mathcal{L}|$ potentially large itemsets take part in the generation of all of Δ^-, D^- as well as Δ^+. So, they would be large in both D and D'. They represent the association rules that remain unchanged as a result of the update. By varying the values of p and q, we can control the degree of similarity between D and D'.

In the following we use the notation Tx.Iy.D$i$$-$$j$$+$$k$, modified from the one used in [3], to denote an experiment using databases with the following sizes: $|D| = |\Delta^-| + |D^-| = i$ thousand, $|\Delta^-| = j$ thousand, $|\Delta^+| = k$ thousand, $|T| = x$ and $|\mathcal{I}| = y$. In the experiments, we set $|\mathcal{L}| = 2100$, $N = 1000$ and $p = q = 2000$.[3] For DHP and FUP2$\mathcal{H}$, we use a hash table of 4096 entries. The hash table is used to prune size-2 candidates (i.e. C_2) *only*.

In each experiment, we first use DHP to find out the large itemsets in D. Then, we run FUP2 and FUP2$\mathcal{H}$, supplying to them the databases Δ^-, D^- and Δ^+ and the large itemsets and their support

[2] This condition is used to model rules that stay in the old as well as new databases.

[3] There are several other parameters for the data generation procedure reported in [4] and [12]. For example, S_q is the clustering size used in the generation of potentially large itemsets, P_s is the size of the pool of potentially large itemsets from which transactions are generated, M_f is the multiplying factor associated with the pool. Following [4], we set $S_q = 5$, $P_s = 50$, $M_f = 200$. Readers are referred to [12] for a detailed explanation of these parameters.

counts in D. The time taken is noted. To compare with the performance of Apriori and DHP, we run these two algorithms on the updated database D', and note the amounts the time they have spent. The time taken by the algorithms are then compared.

4.2 Comparing the four algorithms

The four algorithms are tested against the setting T10.I4.D100−5+5. The support thresholds is varied between 1.0 and 3.0. The results are plotted in Figure 1. It is found that FUP2 is 1.83 to 2.27 times faster than Apriori, while FUP2$\mathcal{H}$ is 1.99 to 2.96 times faster than DHP and is 2.05 to 3.40 times faster than Apriori. To explain the performance gain, let us examine the number of candidate itemsets generated by each algorithm in the scan of D^- for the particular instance with support threshold 2.0 (see Table 6). The total number of candidates generated by FUP2 is 38% of that of Apriori. The candidate size of FUP2$\mathcal{H}$ is 28% of that of DHP and is 21% of that of Apriori. This significant reduction in the number of candidates is the main reason for the performance gain. Clearly, FUP2$\mathcal{H}$ is very efficient because it combines the techniques of both FUP2 and DHP to greatly reduce the number of candidates.

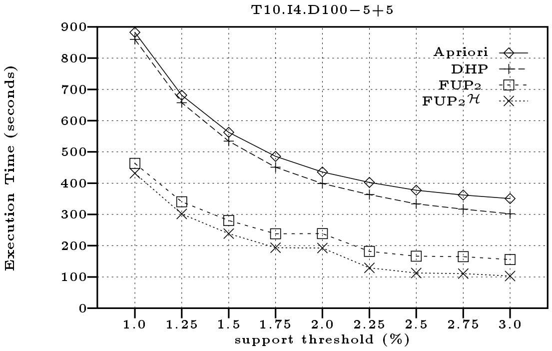

Figure 1: Comparison of the four algoritms

Iteration	Apriori	DHP	FUP2	FUP2$\mathcal{H}$
1	99	99	2	2
2	4095	1615	1794	75
3	3814	3814	2018	2018
4	805	805	34	34
5	697	697	29	29
6	495	495	20	20
7	258	258	7	7
8	92	92	1	1
9	20	20	0	0
10	2	2	0	0
Total	10377	7897	3905	2186
$\frac{\text{Total}}{\text{Apriori}}$	100%	76.1%	37.6%	21.1%
$\frac{\text{Total}}{\text{DHP}}$	131%	100%	49.4%	27.7%

Table 6: Number of candidate itemsets

4.3 Effect of the size of updates

Our next experiment is to find out how the size of Δ^- and Δ^+ affects the performance of the algorithms. We use the setting T10.I4.D100−x+x for the experiment, with a support threshold of 2%. In other words, we use an initial database of 100 thousand transactions. From this database, x thousand transactions are deleted and another x thousand are added to it. Figure 2 shows the results of this experiment. As expected, both FUP2 and FUP2$\mathcal{H}$ have to spend more and more time as the size of updates increases. On the other hand, since the size of the final database D' is constant (100 thousand transactions), the amounts of time spent by Apriori and DHP algorithms are not sensitive to x. Note that FUP2 is faster than Apriori as long as $x \leq 30$ and FUP2$\mathcal{H}$ is faster than DHP for $x < 40$. As Apriori and DHP do not have to scan through Δ^-, their performances are better when $|\Delta^-|$ is very large. These results indicate that the incremental update algorithms are very efficient for a small to moderate size of updates. When the size of the updates exceeds 40% of the original database, Apriori and DHP perform better. This is primarily because that as the amount of changes to the original database becomes large, the updated database is so different from the original one that the previous mining results are not helpful. So, we are better off mining the updated database from scratch when the amount of updates is too large.

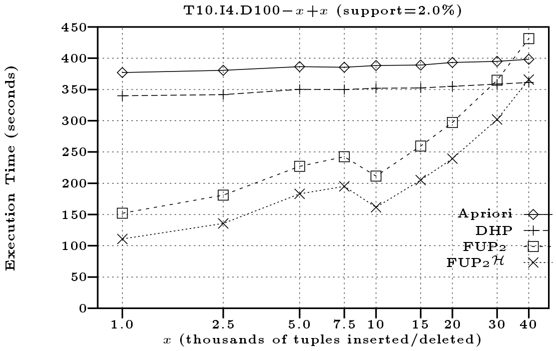

Figure 2: Effect of $|\Delta^-|$ and $|\Delta^+|$

4.4 Varying the number of deleted and added transactions independently

Another experiment is conducted to find out how the size of Δ^- affects the performance of the algorithms. We use the setting T10.I4.D100−x+10 for the experiment. The support threshold is 2%. In other words, we use an initial database of 100 thousand transactions. Ten thousand transactions are added to the database and x thousand are deleted. Figure 3 shows the results of this experiment. As the number of deleted transactions increases, the amounts of time taken by Apriori and DHP decrease, since the size of the final database decreases. For example, at $x = 1.0$, FUP2 is 4.5 times faster than Apriori. As x increases, the number of transactions that FUP2 and FUP2$\mathcal{H}$ have to handle increases; therefore, these algorithms take more and more time as x grows. However, FUP2 and FUP2$\mathcal{H}$ still outperform Apriori and DHP for $x < 30$. Beyond that, Apriori and DHP take less time to finish. This means that as long as the number of deleted transactions is less than 30% of the original database, the incremental algorithms win. Practically, the original database D in a data mining problem is very large. The amount of updates should be much less than 30% of D.

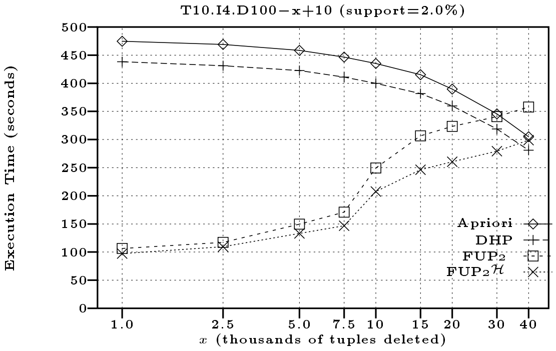

Figure 3: Effect of $|\Delta^-|$

A similar experiment is done using the setting T10.I4.D100−10+x and the same support threshold of 2%. This time, we keep the size of Δ^- constant and vary the size of Δ^+. The results are plotted in Figure 4. As x increases, $|D'|$ increases. So, the execution time of Apriori and DHP increases with x. They do not run faster than FUP2 and FUP2$^{\mathcal{H}}$ even when x is as large as 40.

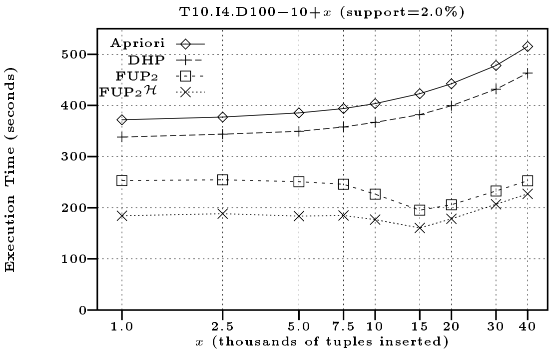

Figure 4: Effect of $|\Delta^+|$

Examining Figure 4 more closely, we notice that the execution time of FUP2 and FUP2$^{\mathcal{H}}$ is quite steady in the range $1.0 \leq x \leq 7.5$. For $x > 15$, the execution time increases with x. This is because the greater the value of x, the more the transactions the algorithms have to handle. However, in the range $7.5 \leq x \leq 15$, the execution time drops as x increases! Also, if we examine Figure 3 more carefully, we can also notice sharper rises in the execution times of FUP2 and FUP2$^{\mathcal{H}}$ in that range of x.

To understand this phenomenon, recall that in iteration k, if an itemset V was not large in D but is in C_k, it is put in Q_k. Suppose that V is also small in D'. Then, since V is small in both D and D', it does not occur frequently in D and D'. Statistically, δ_V^+ and δ_V^- are small in magnitude and they are close to each other. So, $\delta_V = \delta_V^+ - \delta_V^-$ has a very small magnitude. It may be positive or negative. When Lemma 4 is applied to prune Q_k in step 12 of FUP2, a candidate X in Q_k is pruned if $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$. So, when $|\Delta^+| - |\Delta^-| \gg 0$, V has a very high change of being deleted from Q_k. If $|\Delta^+| - |\Delta^-| \geq 0$ but is small in magnitude, V may escape the pruning if δ_V is large enough, although there is still a high chance that V is pruned away. If, however, $|\Delta^+| - |\Delta^-| < 0$, then V will only be pruned away if δ_V is negative enough, but the chance of this is low. Hence, as $|\Delta^+| - |\Delta^-|$ increases from a negative value to a small positive value (e.g., as x in Figure 4 varies from 10 to 15 thousands), the chance that V gets pruned increases.

As there are many itemsets that behave like V, the drop in execution time of FUP2 and FUP2$^{\mathcal{H}}$ is very dramatic when $|\Delta^+|$ increases from slightly below $|\Delta^-|$ to slightly above $|\Delta^-|$. A similar result occurs as $|\Delta^-|$ decreases from slightly above $|\Delta^+|$ to slightly below it.

4.5 Scale-up experiment

To find out if FUP2 and FUP2$^{\mathcal{H}}$ work also for large databases, experiments with scale-up databases are conducted. We use the setting T10.I4.D$x-\frac{x}{10}+\frac{x}{10}$. Again, we use a support threshold of 2%. The results are shown in Figure 5. The execution times of all the four algorithms increase *linearly* as x increases. This shows that FUP2 and FUP2$^{\mathcal{H}}$ are scalable and can work with very large databases.

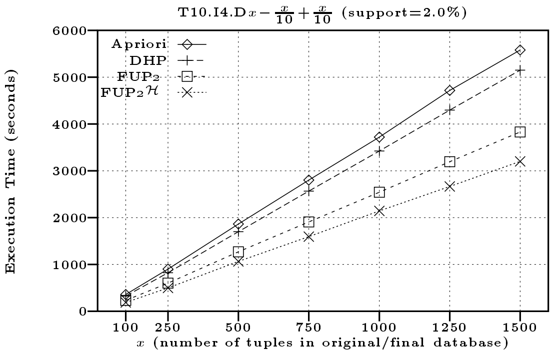

Figure 5: Scale-up experiment

.

5 Discussion

Our new incremental algorithms make use of certain information to achieve their high performance. This information includes the *old* large itemsets and their support counts, the transactions that are not changed by the update (D^-), and the transactions that are inserted or deleted (Δ^+,Δ^-). Is it reasonable to assume that such information be available? The answer is yes.

The large itemsets and their support counts in the original database come from the results of a previous mining activity. We assume that this information is stored. As the association rules can be calculated from these counts efficiently, it is more desirable to store the counts rather than the association rules. Storing the counts enables us to maintain of the association rules efficiently.

A database system that supports recovery keeps all updates into the log files. Consequently, it is possible to retrieve from the log files all the deleted and newly added transactions since the last mining. By identifying the newly inserted transactions in the current *updated* database (e.g. with the help of transaction IDs), we can select from the *updated* database those transactions which have remained unchanged since the last mining activity. Thus, we can obtain the set of unchanged transactions. Hence Δ^-, D^- and Δ^+ are available.

6 Conclusions

We studied an efficient incremental updating technique for the maintenance of association rules discovered by database mining. This technique updates the association rules when old transactions are removed from the database and new transactions are added to it. It uses the information available from a previous mining to reduce the amount of work that has to be done to discover the association rules in the updated database. It is a generalization of two previous algorithms: Apriori [3] and FUP [4]. Performance studies show that the new technique is significantly faster than mining the updated database from scratch. The new technique works well over wide ranges of system parameter values. In particular, it works well for updates of a wide range of insertion sizes and small to moderate deletion sizes.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. 18th Int. Conf. Very Large Data Bases*, pp. 560–573, Vancouver, Canada, August 1992.

[2] R. Agrawal, T. Imielinski and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD International Conference on Management of Data*, Washington, DC, May 1993.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proc. 20th Int. Conf. on Very Large Databases*, pp. 487–499, Santiago, Chile, 1994.

[4] 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 *Proc. 12th Int. Conf. on Data Engineering*, New Orleans, Louisiana, 1996.

[5] D. W. Cheung, J. Han, V. T. Ng, A. Fu and Y. Fu. A Fast Distributed Algorithm for Mining Association Rules. In *Proc. 4th Int. Conf. on Parallel and Distributed Information Systems*, Miami Beach, Florida, Dec. 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, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, pp. 29–40, 1993.

[8] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, Zürich, Switzerland, pp. 420–431, Sept. 1995.

[9] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia and O. R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. International Conf. on Data Mining and Knowledge Discovery (KDD-96)*, Portland, Oregon, August 1996.

[10] 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. Conf. on Information and Knowledge Management*, pp. 401–408, Gaithersburg, Maryland, Nov. 1994.

[11] H. Lu, R. Setiono, and H. Liu, NeuroRule: A Connectionist Approach to Data Mining In *Proc. 21th Int. Conf. Very Large Data Bases*, pp. 478–489, Zürich, Switzerland, Sept 1995.

[12] J. S. Park, M. S. Chen and P. S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. ACM SIGMOD International Conference on Management of Data*, San Jose, California, May 1995.

[13] J. S. Park, M. S. Chen, and P. S. Yu, Efficient Parallel Data Mining for Association Rules. In *Proc. 1995 Internation Conference on Information and Knowledge Management*, Baltimore, MD, Nov 1995.

[14] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 1–12, Montréal, Canada, June, 1996.