Maintenance of Discovered Association Rules: When to update?

S.D. Lee
Department of Computer Science,
The University of Hong Kong,
Hong Kong.
sdlee@cs.hku.hk

David W. Cheung
Department of Computer Science,
The University of Hong Kong,
Hong Kong.
dcheung@cs.hku.hk

Abstract

In this paper, we devise an algorithm with which we can estimate the difference between the association rules in a database before and after it is updated. The estimated difference can be used to determine whether we update the mined association rules or not. If the estimated difference is large, then it is time to update the mined association rules in order to discover and learn the new rules and discard the old ones. If the estimated difference is small, then the rules in the original database is still a good approximation for those in the updated database. We do not have to spend the resources to update the rules. We can accumulate more updates before actually updating the rules, thereby avoiding the overheads of updating the rules too frequently.

1 Introduction

Data mining has been attracting much attention from practitioners and researchers in recent years. Combining techniques from the fields of machine learning, statistics and databases, data mining enables us to find out useful and invaluable information from huge databases. It has wide applicability and a promising future.

Mining of association rules is an important research topic among the various data mining problems. Many interesting works have been recently published on this problem and its variants [AIS93, AS94, CHNW96, CHN+96, HF95, HKMT95, KMR+94, PCY95b, PCY95a, SA96]. A classical example is about the retail industry or supermarket sales database. Typically, a record in the sales database describes all the items that are bought in a single transaction, together with other information such as the transaction time, customer-id, etc. The classical association rule mining problem is to find out all the rules like "A customer who buys item X and item Y is also *likely* to buy item Z in the same transaction" from the sales database, where X, Y and Z are initially unknown. This is a very important problem because knowing such rules helps marketers to develop and implement customized marketing programs and strategies.

We will concentrate on the problem of maintaining discovered association rules in this paper. The sales database is not a static database, because updates are constantly being applied to it. New records are frequently added to record purchase activities. Old records in the database are deleted from the database to save storage space or because they are out of interest. In addition, existing records may be edited or changed to correct manual operational errors or for other reasons. Because of these update activities, the database keeps on changing. Consequently, new association rules may appear in the database and at the same time, some existing association rules would become invalid. Thus, maintenance of discovered association rules is an important problem.

The problem of maintaining discovered association rules was first studied in [CHNW96]. That paper proposes the FUP algorithm, which can update the discovered association rules in a database when new transactions are *added* to the database. A more general algorithm FUP$_2$ was proposed in [CLK97]. This latter algorithm can update the discovered association rules when new transactions are added to, deleted from, or modified in the database.

Although the FUP and FUP$_2$ algorithms are very efficient, in the long run they still have certain performance overhead. We can of course apply FUP$_2$ each time a new transaction is added to or deleted from the database. However, the overhead of such a run of FUP$_2$ is very high. The total long-term overhead will be large because the algorithm is applied too frequently. On the other hand, if we always wait until a large amount of updates has accumulated before applying FUP$_2$, we will not be able to discover the newest association rules quickly. This is not desirable, especially in time-critical applications where decisions has to be made as soon as a new phenomenon is observed. So, applying the FUP$_2$ algorithm too infrequently, we would not be able to discover the new association rules in time. Thus, we have a challenge. To have best results, the FUP and FUP$_2$ algorithms should be run at suitable times.

To tackle this challenge, we attempt to find out some measures which can suggest to us when the

FUP2 algorithm should be applied to find out the new association rules in the updated database. A first intuition is to use the amount of the updates as an indicator. When the amount of accumulated updates exceeds a certain threshold, FUP2 is applied. This naïve idea does not work well. This is because if the updates follow the same association rule patterns as the old database, then both the old and new databases would exhibit the same association rules. In this case, however large the amount of updates, few useful new association rules can be found. Another intuitive approach is to find out the new association rules in the updated database, and then compare them with the old rules to find out their difference. The size of the difference is a good indicator for the necessity to discover the new association rules. However, calculating this size requires us to find out the new association rules first, and this operation involves great overhead.

In this paper, we propose a third approach. We propose a new algorithm DELI (Difference Estimation for Large Itemsets) which uses a sampling technique to estimate the difference between the old and new association rules. This estimate is used as an indicator for whether the FUP2 algorithm should be applied to the database to accurately find out the new association rules. If the estimated difference is large enough (w.r.t. some user specified thresholds), the algorithm signals the need of an update operation, which can be accomplished by using the FUP2 algorithm. If the estimated difference is small, then we do not run FUP2 immediately. In this case, since the difference between the old and new association rules is small, we can take the old rules as an approximation of the new rules. So, we wait until more changes are made to the database, and then reapply the DELI algorithm. Experimental results show that this new approach is applicable and can bring savings on machine resources. There have been studies on the application of sampling techniques in the mining of association rules [MT96, MTV94, Toi96]. However, there is not much research work on applying these techniques to the rule maintenance problem.

In the remaining of this paper, we will first give a more precise description of the problem. Next, we briefly go through two previous algorithms on which our new algorithm is based. Then, a detailed description of our new algorithm is given. After discussing some implementation issues, we present our experimental results. Finally, we give a conclusion at the end of this paper.

2 Problem Definitions

2.1 Mining of association rules

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals (called *items*) and D be a database of transactions. Each transaction

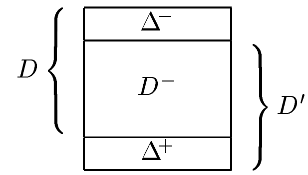

Figure 1: Definitions of D, D', Δ^-, D^- and Δ^+

$T \in D$ is a non-empty set of items (i.e. $T \neq \emptyset$ and $T \subseteq I$). Given an *itemset* X (i.e. $X \subseteq I$) and a 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, which is 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 total 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* $\varsigma\%$ in D if $\sigma_{X \cup Y} = |D| \times \varsigma\%$. 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 [AIS93].

Thus, if $s\%$ is the given support threshold, the mining problem is reduced to the problem of finding the set $L = \{X | X \subseteq I \ \wedge \ \sigma_X \geq |D| \times s\%\}$. We call an itemset that contains exactly k items a *k-itemset* and 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. So, without loss of generality, we assume that there are no transaction modifications. All updates are either transaction deletion or insertion. Let Δ^- be the set of deleted transactions and Δ^+ be the set of newly added transactions. We assume 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 $D^- = D - \Delta^- = D' - \Delta^+$. The relationships between these data sets are illustrated in Figure 1.[1]

We again use σ_X to denote the support count of itemset X in the original database D. The set of large

[1]Since the ordering of the transactions within the data sets does not affect the results of the association rule mining problem, we have moved the deleted transactions to the top and newly added transactions to the bottom for illustration purpose.

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

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: $\sigma'_X = \sigma_X + \delta_X^+ - \delta_X^- = \sigma_X + \delta_X$.

For the update problem, L and $\sigma_X\ \forall X \in L$ are information available as the result of a previous mining operation done on the old database D. Thus, the update problem is to find L' and $\sigma'_X\ \forall X \in L'$ efficiently, given the knowledge of D, D', Δ^-, D^-, Δ^+, L and $\sigma_X\ \forall X \in L$.

2.3 Estimation of the difference between the old and new association rules

Before actually doing the update to find L' and $\sigma'_X\ \forall X \in L'$, we would like to estimate the difference between L and L'. If the difference is small, then we do not update the association rules, but instead wait for more updates to come. If, however, the difference is larger than a given threshold, then enough updates have been accumulated and an update operation is necessary. To do this, we have to define a measure for the difference between L and L'.

The difference between the old large itemsets L and new large itemsets L' can be measured by the set symmetric difference between them. We use the notation $L \ominus L'$ to denote the symmetric difference between L and L'. Note that

$$L \ominus L' = L' \ominus L = (L' - L) \cup (L - L')$$

Depending on the similarity between L and L', the size of $L \ominus L'$ can vary between zero and $|L| + |L'|$. It is zero when $L = L'$ and it is $|L| + |L'|$ when L and L' are disjoint. The smaller the size of $L \ominus L'$, the greater the similarity between L and L'.

It is clear that the ratio $\frac{|L \ominus L'|}{|L|+|L'|}$ can be used as a relative measurement of the difference between L and L'. Its value varies between 0 and 1 and hence it is convenient to specify a threshold for this ratio. Unfortunately, the term $|L'|$ appears in the denominator. In this estimation problem, we do not know L'; therefore, we cannot calculate the above ratio. We have chosen to use the ratio $\frac{|L \ominus L'|}{|L|}$ instead. We use this as a difference measure for the old and new large itemsets. Since L is known to us from the results of the last mining, it remains to estimate the value of $|L \ominus L'|$.

So, the estimation problem is to efficiently estimate the size $|L \ominus L'|$ without finding out L', given the knowledge of D, D', Δ^-, D^-, Δ^+, L and $\sigma_X\ \forall X \in L$.

3 Previous Works

In this section, we will briefly go through two algorithms: The Apriori algorithm [AS94] and the FUP2 algorithm [CLK97]. Readers are referred to the cited papers for detailed information. The former is an algorithm for mining association rules. It does not address the maintenance problem, which latter algorithm solves.

3.1 The Apriori Algorithm

The Apriori algorithm [AS94] was designed to solve the problem of mining association rules, which was first introduced in [AIS93]. Apriori is one of the most successful algorithms for solving this problem, which is formulated in section 2.1.

The Apriori algorithm finds out the large itemsets iteratively. In the k-th iteration, it finds out L_k, i.e. the set all large itemsets of size k. To do this, in each iteration the algorithm first generates a set of candidate itemsets of size k, denoted by C_k. For the first iteration, C_1 contains all the 1-itemsets. For the subsequent iterations, C_k is generated by applying the `apriori_gen` function [AS94] on the set L_{k-1}, i.e. the set of all large $(k-1)$-itemsets, which have been found in the previous iteration. The `apriori_gen` function generates all those k-itemsets X satisfying the condition that all the size $(k-1)$ subsets of X be large (i.e. $\in L_{k-1}$). Since this is a *necessary* condition for a k-itemset to be large [AS94], the `apriori_gen` function guarantees that all large k-itemsets are included in C_k, i.e. $C_k \supseteq L_k$. Having found the set of candidates C_k, Apriori scans the database D in order to obtain the support counts σ_X for all $X \in C_k$. Next, all candidates $X \in C_k$ with support count $\sigma_X \geq |D| \times s\%$ are added to the set L_k. Thus L_k is found. This completes one iteration. The iterations go on until L_j is empty for some j. The set of all large itemsets L is then the union $\bigcup_{1 \leq k < j} L_k$. The key success of the Apriori algorithm is its use of the `apriori_gen` function to generate a small number of candidate itemsets.

3.2 The FUP2 algorithm

The FUP2 algorithm [CLK97] was designed to address the maintenance problem, formulated in section 2.2. Suppose that the association rules of an existing database has already been mined, and that after the mining, some updates have been performed on the database. To find out the new association rules L' in the updated database D', one approach is to apply the Apriori algorithm to the updated database D'. However, this method is not efficient, because it fails to reuse the results of the previous mining. Thus, the FUP2 algorithm was introduced. FUP2 makes use of the results of the previous mining to reduce the amount of work that needs to be done. FUP2 also improves performance by generating a smaller number of candidate itemsets. It works more efficiently.

The FUP2 algorithm works as follows. Like Apriori, it generates the large itemsets iteratively. In the k-th iteration, it first generates a set of candidate itemsets C_k for the updated database D'. For the first iteration, C_1 is again the set of all 1-itemsets. In the subsequent iterations, C_k is generated by applying `apriori_gen` on L'_{k-1}, the new large itemsets (w.r.t D') found in the previous iteration. The properties of `apriori_gen` guarantees that $C_k \supseteq L'_k$. Next, C_k is divided into two partitions: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. Note that L_k is available from the results of the last mining, and so are $\sigma_X \ \forall X \in L_k$.

The candidates in the two partitions of C_k are handled differently. For $X \in P_k \subseteq L_k$, we know the old support count σ_X from the old mining results. So, we can scan Δ^+ and Δ^- to find out δ_X and hence calculate the new support count $\sigma'_X = \sigma_X + \delta_X$. If this is greater than or equal to $|D'| \times s\%$, then X is large, and hence it is added to the set L'_k. For each candidate $X \in Q_k$, we do not know its old support count σ_X, but we know that $\sigma_X < |D| \times s\%$. So, such a candidate can be large *only if* $\delta_X > (|\Delta^+| - |\Delta^-|) \times s\%$, according to Lemma 4 in [CLK97]. Hence, in the scan of Δ^+ and Δ^-, we can obtain also the counts δ_X for the candidates $X \in Q_k$. Then, we prune away from Q_k those candidates with $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$. For the remaining candidates $X \in Q_k$, we scan the unchanged transactions D^- to find out their counts there, and then add this count to δ^+_X to get σ'_X. Those with $\sigma'_X \geq |D'| \times s\%$ are added to L'_k. As a result, all the candidates in $C_k = P_k \cup Q_k$ can be handled, and the candidates that are large in D' are added to L'_k. Hence an iteration is completed, and the algorithm proceeds to the next iteration unless L'_k is empty.

4 The DELI algorithm

We have already briefly gone through the Apriori and FUP2 algorithms in Section 3. FUP2 is efficient because it makes use of the result of a previous mining. This reduces the amount of work to be done as well as the number of candidate itemsets generated. However, FUP2 still has to rescan the old database several times in order to obtain the support counts of new itemsets. In order to find out accurate results, it has to spend much time. In fact, before doing the update, we can first estimate the amount of changes of the association rules caused by the updates to the database. If the changes are not significant, we can ignore the changes for the moment, and wait until more changes are accumulated before actually applying FUP2 to update the association rules. This is the idea behind the DELI algorithm. It uses a sampling technique to do the estimation.

4.1 Drawing a random sample

Consider the original database D and an arbitrary itemset $X \subseteq I$. Since exactly σ_X of the $|D|$ transactions in the database contain X (by the definition of σ_X), the probability that a transaction randomly selected[2] from D contains X is $p_X = \frac{\sigma_X}{|D|}$.

Now, suppose we draw m transactions from the database D with replacement to form a sample S. Each transaction is drawn independently. As a result, each transaction in S has a probability of p_X of containing the itemset X. Let the total number of transactions in S containing X be T_X (i.e. T_X is the support count of X in S.). Then T_X is a binomially distributed random variable with parameters m and p_X. With m sufficiently large (≥ 30), T_X can be approximated by a normal distribution with mean $m \cdot p_X$ and variance $m \cdot p_X(1-p_X)$. [Tri88] We can estimate the value of σ_X by the point estimator $\widehat{\sigma_X} = \frac{T_X}{m} \cdot |D|$. This is an unbiased estimator because $\widehat{\sigma_X}$ is normally distributed with mean $m \cdot p_X \cdot \frac{|D|}{m} = \sigma_X$. Its variance is $m \cdot p_X(1-p_X) \cdot (\frac{|D|}{m})^2 = \sigma_X(|D| - \sigma_X)/m$. Thus we can obtain a $100(1-\alpha)\%$ confidence interval $[a_X, b_X]$ for σ_X where

$$a_X = \widehat{\sigma_X} - z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D| - \widehat{\sigma_X})}{m}}$$
$$b_X = \widehat{\sigma_X} + z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D| - \widehat{\sigma_X})}{m}}$$

and $z_{\alpha/2}$ is the critical value such that the area under the standard normal curve beyond $z_{\alpha/2}$ is exactly $\alpha/2$.[3] The value of α is chosen by the user. The $100(1-\alpha)\%$ confidence interval for σ_X has the property that

$$\Pr(\sigma_X \in [a_X, b_X]) = 100(1-\alpha)\%$$

So, there is $100(1-\alpha)\%$ chance that the actual value of σ_X lies on the interval. In the above derivations, X is an arbitrary itemset. So, the above result generalizes to any itemset $X \subseteq I$.

We have seen how an estimate of σ_X can be obtained from a sample. In the next subsection, we will see how the confidence interval $[a_X, b_X]$ is employed. For efficiency, we will use the same sample S throughout the algorithm. The sample S is drawn from the original database D as a part of the initialization of the algorithm. After this initialization, we do not need to scan D^- at all.

[2] Each transaction in D is selected with equal probability.

[3] In other words, $z_{\alpha/2}$ is the value such that the probability that a normally distributed random variable with zero mean and unit variance exceeds $z_{\alpha/2}$ is $\alpha/2$. i.e. $\Pr(Z > z_{\alpha/2}) = \alpha/2$.

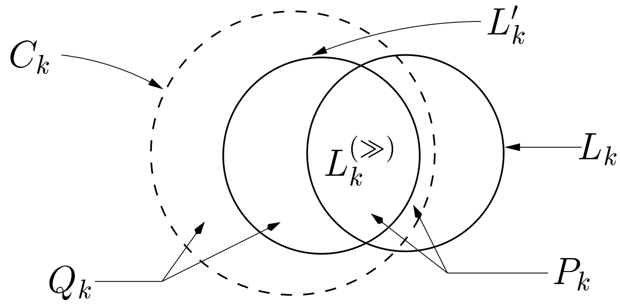

Figure 2: The relationship between various sets of itemsets

4.2 Estimating the symmetric difference

The DELI algorithm is also an iterative algorithm, like Apriori and FUP2. In the k-th iteration, the algorithm first generates a set C_k of candidate itemsets. For the first iteration, C_1 is the set of 1-itemsets. For subsequent iterations ($k \geq 2$), C_k is calculated by applying the `apriori_gen` function [AS94] on the set $\widehat{L_{k-1}}$ calculated in the previous iteration as described below. Note that $\widehat{L_{k-1}}$ is an approximation of L'_{k-1}. So, the candidate set C_k so generated may not cover all the itemsets in L'_k. However, itemsets missed by the approximation $\widehat{L_{k-1}}$ most likely have very small support counts, and hence candidates generated by them are unlikely to be large. So, C_k should cover most of the itemsets in L'_k, and the missed ones are unimportant ones (in the sense that they have low support counts).

After its generation, the set C_k is partitioned into two parts as in the FUP2 algorithm: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. (See Figure 2.) Thus, P_k contains all the candidates that were large in the old database, while Q_k contains those that were not large. Candidates in these two partitions are treated differently. For each itemset $X \in P_k$, its old support count σ_X in the old database can be retrieved from the previous mining results. Then, we find out δ_X^+ and δ_X^- by scanning the updates Δ^+ and Δ^-. Thus, the new support count σ'_X can be calculated. Those itemsets X with $\sigma'_X \geq |D'| \times s\%$ are large in the updated database. We add them to a set $L_k^{(\gg)}$. Note that the itemsets $Y \in (L_k - C_k) \cup (P_k - L_k^{(\gg)}) = L_k - L'_k$ are those that were large in the old database but not large in the new database. (Refer to Figure 2.) We count the number of such itemsets and denote this number by η_k.

For the candidates X in Q_k, we do not know σ_X. However, we know that $\sigma_X < |D| \times s\%$. We can apply the technique mentioned in [CLK97] to prune away some candidates. We can first find out δ_X by scanning the updates Δ^+ and Δ^-. Then, we are able to prune away some of the candidates before going on. This is because according to Lemma 4 in [CLK97], if $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$, then $\sigma'_X = \sigma_X + \delta_X < (|D| + |\Delta^+| - |\Delta^-|) \times s\% = |D'| \times s\%$ and hence X cannot be large. So, none of the itemsets $X \in Q_k$ satisfying $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$ can be large. They can be safely pruned away from Q_k. For each of the remaining candidates X in Q_k, we use the method as described in the previous subsection to find out a $100(1-\alpha)\%$ confidence interval $[a_X, b_X]$ for σ_X. This corresponds to the interval $[a_X + \delta_X, b_X + \delta_X]$ for σ'_X. Since the criterion for being "large" is $\sigma'_X \geq |D'| \times s\%$, We compare this threshold value with the interval for σ'_X. There are 3 possibilities:

1. If $|D'| \times s\% < a_X + \delta_X$, then σ'_X has a large chance to exceed the threshold. Thus, it is likely to be large. We add this itemset X to the set $L_k^{(>)}$.
2. If $a_X + \delta_X \leq |D'| \times s\% \leq b_X + \delta_X$, then σ'_X may or may not exceed the threshold. So, it may be large or not. Such a candidate is added to the set $L_k^{(\approx)}$.
3. If $b_X + \delta_X < |D'| \times s\%$, then σ'_X has little chance to exceed the threshold. So, X is very likely to be not large. So, such a candidate is dropped.

After all the candidates in Q_k has been handled as described above, we calculate the value $\xi_k = \left|L_k^{(>)}\right| + \left|L_k^{(\approx)}\right|$. Note that for an itemset $X \in L'_k - L_k$, X will have a high chance of being captured in cases 1 and 2 above. It has a very little chance of escaping via case 3. On the other hand, not all itemsets in $L_k^{(\approx)}$ are actually large in the new database. Some of them may be false hits. So, there is a high chance that $|L'_k - L_k|$ is less than ξ_k. So, we can confidently treat ξ_k as if it were an upper bound for $|L'_k - L_k|$.

This completes one iteration. We have not found the exact value of the set of new large itemsets L'_k and we do not intend to do so. We can make a conservative guess $\widehat{L_k} = L_k^{(\gg)} \cup L_k^{(>)} \cup L_k^{(\approx)}$. The set $\widehat{L_k}$ should contain most of the large itemsets in L'_k. This guess is conservative because there is statistically more false hits than misses. Whether the next iteration is performed depends on several criteria. Our first criterion is the degree of uncertainty that is introduced by our estimation of σ_X with the confidence interval $[a_X, b_X]$ for the candidates in Q_k. The uncertainties so introduced are gathered in the set $L_k^{(\approx)}$. So, we can compare the uncertainty factor $u_k = \frac{\left|L_k^{(\approx)}\right|}{\left|\widehat{L_k}\right|}$ against a pre-specified threshold $\overline{u}$. If this factor is too large, then the amount of uncertainty introduced in this iteration is too large to reliably generate the candidates for the next iteration. So, we declare that a good estimation cannot be made and we resort to FUP2 to do an accurate update.

The second criterion is the amount of changes in the set of large itemsets caused by Δ^- and Δ^+. As mentioned in Section 2.3, we examine the symmetric difference $L \ominus L'$. Note that the value of η_k found above gives the exact size of $L_k - L'_k$ while the value of ξ_k is a reliable approximate "upper bound" on the size of $L'_k - L_k$. So, the value of $\eta_k + \xi_k$ is an approximate "upper bound" on $|L_k \ominus L'_k| = |L_k - L'_k| + |L'_k - L_k|$. Thus, we test the ratio $d_k = \sum_{j=1}^{k}(\eta_j + \xi_j)/|L|$ against a user-specified threshold $\overline{d}$. If this threshold is exceeded, then the set of old large itemsets are so different from the set of new large itemsets that an accurate update is deemed necessary. So, FUP2 should be called to do the update.

The third criterion is to see if the number of large itemsets generated in this iteration is sufficient to generate any candidates for the next iteration. If so, the algorithm continues with the next iteration. If not, the algorithm terminates. If the algorithm comes to this point of termination, the amount of uncertainty and the size of the symmetric difference $L \ominus L'$ has not exceeded their thresholds, since $u_k < \overline{u} \;\wedge\; d_k < \overline{d} \;\; \forall k$. So, we can conclude that the we have enough certainty to say that the large itemsets in the new database are not too different from those in the old database. Hence, it is acceptable to take the large itemsets of the old database as an approximation of the updated database. Below is a summary this algorithm.

1. Obtain a random sample S of size m from the original database D. Set $k := 1$.
2. Generate a candidate set C_k. For $k = 1$, $C_1 := I$. For $k \geq 2$, $C_k = \texttt{apriori\_gen}(\widehat{L_{k-1}})$.
3. Divide C_k into 2 parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$.
4. Scan Δ^+ and Δ^- to obtain δ_X for all $X \in C_k$.
5. For each $X \in P_k$, retrieve σ_X from the results of the previous mining operation. Then, calculate $\sigma'_X = \sigma_X + \delta_X$. If $\sigma'_X \geq |D'| \times s\%$, add X to $L_k^{(\gg)}$.
6. Calculate $\eta_k = |L_k - C_k| + |P_k - L_k^{(\gg)}|$.
7. For each $X \in Q_k$, if $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$, delete it from Q_k.
8. For the remaining $X \in Q_k$, obtain a $100(1-\alpha)\%$ confidence interval $[a_X, b_X]$ for σ_X by examining the sample S. Then, do one for the following for X:
   (a) If $|D'| \times s\% < a_X + \delta_X$, add X to $L_k^{(>)}$.
   (b) If $a_X + \delta_X \leq |D'| \times s\% \leq b_X + \delta_X$, add X to $L_k^{(\approx)}$.
   (c) If $b_X + \delta_X < |D'| \times s\%$, drop X.
9. Calculate $\xi_k = \left|L_k^{(>)}\right| + \left|L_k^{(\approx)}\right|$.
10. Let $\widehat{L_k} = L_k^{(\gg)} \cup L_k^{(>)} \cup L_k^{(\approx)}$.
11. If $u_k \geq \overline{u}$, signal the need for an update operation.
12. If $d_k \geq \overline{d}$, signal the need for an update operation.
13. If $\widehat{L_k}$ is non-empty, increment k and goto step 2. Otherwise, conclude that $L \approx L'$ and hence use L as an approximation for L'.

Note that the DELI algorithm only scans the original database D once, in order to pick a sample S. The size of the sample should be small enough so that it fits in main memory. Thereafter, the algorithm only examines the updates Δ^+ and Δ^-, whose sizes are not very large. So, the DELI algorithm has very little I/O overhead.

5 Discussions

5.1 Sample size

In Section 4.1, we have derived that for a sample S of size m, the $100(1-\alpha)\%$ confidence interval for σ_X is

$$\widehat{\sigma_X} \pm z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D| - \widehat{\sigma_X})}{m}}$$

So, the width of this interval is $2z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D|-\widehat{\sigma_X})}{m}}$. This value gives us an idea of how accurate the estimation is. The smaller the interval width, the more accurate the estimation. So, the interval width gives us a guideline for choosing the suitable value of m.

From the above expression for the interval width, it can be observed that a larger value of m gives a the small interval width. So, by increasing the sample size, we can narrow the width of the confidence interval. We may impose a requirement on the interval width, e.g. requiring it to be no wider than $\frac{|D| \times s\%}{10}$, and then establish an inequality for the interval width. Solving this inequality gives a critical value for the sample size m. Whenever m exceeds this critical value, the interval width is smaller than the required value, and hence the desired accuracy is achieved.

5.2 Availability of Δ^+, D^- and Δ^-

Note that our new algorithm DELI requires that the transactions in Δ^+, D^- and Δ^- be available. This means we have to be able to divide the updated database D' into D^- and Δ^+, while the transactions in Δ^- has to be kept. How can we achieve this?

We remark that a database system that supports recovery keeps all update activities into the log files. Consequently, we are able to retrieve the deleted transactions from the log files to construct Δ^-. Moreover, from the log files, we can also know which transactions are newly added to the database. So, the transactions belonging to Δ^+ can be identified in D' and hence D' can be partitioned into D^- and Δ^+. Another approach is to use techniques such as time-stamping or "archive" bit to keep track of which transactions in D' are in D^-. Thus, Δ^+, D^- and Δ^- can all be made available to DELI.

5.3 Consecutive runs

Suppose we have applied the Apriori algorithm to find out the association rules in a database. Some time later, a batch of update arrives and we use the DELI algorithm to estimate whether an updating of the rules is necessary. Suppose the answer is negative, so we continue to take the old association rules as valid rules. Now, suppose a second batch of update has arrived. We have to reapply the DELI algorithm to estimate whether the 2 batches of updates together has introduced significant changes to the association rules in the database. Suppose the result is still negative. Later, when a third batch of updates is available, we will have to rerun the DELI algorithm a third time on all the 3 batches of updates.

In these consecutive runs of the DELI algorithm, much work is indeed duplicated. It is possible to avoid such repeated work by storing some intermediate results of the earlier runs, so that the later runs can reuse the old results without having to recalculate them from scratch.

6 Experimental Results

We have implemented our new algorithm DELI in order to assess its performance. A series of experiments have been carried out to compare its performance with FUP2. We used synthetic databases for the experiments. The generation method of the synthetic databases is the same as in [CLK97]. Owing to the lack of space, we are only giving brief descriptions of the results in this paper.

In our experiments, DELI is around 3 times faster than FUP2 when they are run on the same databases. This speedup ratio slightly decreases with the size of updates (i.e. $|\Delta^+|$ and $|\Delta^-|$). However, for update sizes of up to 10% of the original database (D), the speedup is still significant. This speedup ratio is an indicator of the amount of machine resources that can be saved by running DELI instead of FUP2. So, for practical situations where the size of updates is much smaller, much resource can be saved. Furthermore, our scaleup experiments show that the speedup ratio increases significantly with very large database sizes ($|D^-|$). This demonstrates that our new algorithm is very highly scalable.

Concerning the accuracy of the estimates, DELI shows encouraging results in our experiments. The uncertainty factors u_k's remained below 0.01. This means that DELI usually have very sufficiently certain information to work on. Hence, the new algorithm is feasible. We have also compared the value of d_k's, estimates for $|L \ominus L'|/|L|$, against the actual value obtained by direct measurements. The estimates are always larger than the actual value, confirming our claim that the estimates give *upper bounds* to the actual values. Hence, DELI does not fail to signal a rule-update operation when it is desired. Moreover, we have found that the estimates never overshoot by a factor 2.5. So, the conservatively estimated upper-bounds are reasonably tight. Rule-update operations are not triggered very often.

7 Conclusions

In this paper, we have introduced an algorithm for estimating the difference between the association rules of a database before and after updates. The algorithm uses sampling and statistical methods to make a reliable estimation. If the estimated difference is small, the old association rules can be taken as a good approximation of the new ones. If the estimated difference is large, the necessity of updating of the association rules in the database is signaled. We have also discussed about the size of the sample and how consecutive runs of the algorithm can be made more efficient. The feasibility and applicability of this new approach to the maintenance of association rules is confirmed by our experimental results.

References

[AIS93] 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*, page 207, Washington, DC, May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.

[CHN+96] David W. Cheung, Jiawei Han, Vincent T. Ng, Ada Fu, and Yongjian. Fu. A fast distributed algorithm for mining association rules. In *Proc. Fourth International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[CHNW96] David W. Cheung, Jiawei Han, Vincent T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the Twelfth International Conference on Data Engineering*, New Orleans, Louisiana, 1996. IEEE Computer Society.

[CLK97] David W. L. Cheung, S. D. Lee, and Benjamin Kao. A general incremental technique for maintaining discovered association rules. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, 1–4 April 1997.

[HF95] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st VLDB Conference*, pages 420–431, Zurich, Switzerland, 1995.

[HKMT95] Marcel Holsheimer, Martin Kersten, Heikki Mannila, and Hannu Toivonen. A perspective on databases and data mining. In *First International Conference on Knowledge Discovery and*

Data Mining (KDD'95), pages 150–155, Montreal, Canada, August 1995. AAAI Press.

[KMR+94] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding interesting rules from large sets of discovered association rules. In Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, editors, *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401–407. ACM Press, November 1994.

[MT96] Heikki Mannila and Hannu Toivonen. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems Research '96*, 973–978, Vienna, Austria, April 1996. Austrian Society for Cybernetic Studies.

[MTV94] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Databases (KDD'94)*, pages 181 – 192, Seattle, Washington, July 1994. AAAI Press.

[PCY95a] J. S. Park, M. S. Chen, and P. S. Yu. Efficient parallel data mining for association rules. In *Proc. 1995 International Conference on Information and Knowledge Management*, Baltimore, MD, November 1995.

[PCY95b] Jong Soo Park, Ming-Syan Chen, and Philip 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.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proc. ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[Toi96] Hannu Toivonen. Sampling large databases for finding association rules. In *Proceedings of the 22th International Conference on Very Large Databases (VLDB'96)*, pages 134–145, Mumbay, India, 3–6 September 1996. Morgan Kaufmann.

[Tri88] Kishor Shridharbhai Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* Prentice Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001, 1988.