Is Sampling Useful in Data Mining?
A Case in the Maintenance of Discovered Association Rules.

S.D. Lee David W. Cheung Ben Kao

Department of Computer Science,
The University of Hong Kong,
Hong Kong.
{*sdlee,dcheung,kao*}*@cs.hku.hk*

Abstract

By nature, sampling is an appealing technique for data mining, because approximate solutions in most cases may already be of great satisfaction to the need of the users. We attempt to use sampling techniques to address the problem of maintaining discovered association rules. Some studies have been done on the problem of maintaining the discovered association rules when updates are made to the database. All proposed methods must examine not only the changed part but also the unchanged part in the original database, which is very large, and hence take much time. Worse yet, if the updates on the rules are performed frequently on the database but the underlying rule set has not changed much, then the effort could be mostly wasted. In this paper, we devise an algorithm which employs sampling techniques to estimate the difference between the association rules in a database before and after the database is updated. The estimated difference can be used to determine whether we should update the mined association rules or not. If the estimated difference is small, then the rules in the original database is still a good approximation to those in the updated database. Hence, 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. Experimental results show that our algorithm is very efficient and highly accurate.

1 Introduction

Data mining has recently attracted considerable attention from database practitioners and researchers because of its applicability in many areas, such as decision support, market strategy and financial forecasts. 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.

Recent developments in technology have enabled many organizations to collect massive amounts of data from bar-code labels, credit cards, OCRs, cash dispensers, etc. Meanwhile, the rapidly decreasing cost of storage devices allows the storage of such databases at a low cost. These databases are now seen by the organizations as valuable assets. Much new, previously unknown knowledge can be discovered from these databases. Such new knowledge is very important because it enables marketers to develop and implement customized marketing programs and strategies. To extract such new information from large databases is the task of data mining.

Mining of association rules is a research topic that has received much attention among the various data mining problems. Many interesting works have been published recently on this problem and its variations [1–4, 6, 9, 12–14]. Here, we give a classical example concerning the retail industry. Typically, a sales database of a supermarket stores, for each transaction, all the items that are bought in that transaction, together with other information such as the transaction time, customer-id, etc. The association rule mining problem is to find out all the rules which have the form "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 not known beforehand.

This problem was first introduced in [1] and subsequently many studies have been carried out on it. Among the proposed algorithms for solving this problem, the Apriori algorithm [2] and the DHP algorithm [13] are the most efficient and successful. However, these algorithms examine the whole database in order to discover the association rules *exactly*. A large amount of time has to be spent on examining the huge database. In practice, a 100% correct mining result would not be much more useful than a 95% accurate result, because the data are collected from real-world observations, in which random error is deemed to be incurred. So, can we sacrifice that 5% of accuracy for a higher mining speed? And if so, how? These questions can be answered by *sampling techniques* [8, 10, 11, 15]. Studies have shown that by examining a small[1] sample of the database, association rules can be discovered very efficiently with reasonably high accuracy and confidence. The sampling methods are efficient for two main reasons. First, they only examine a small sample instead of the entire, huge database. Secondly, since the sample is small in size, it can often fit entirely into main memory, thus reducing the I/O overhead of repeated database scanning.

Closely linked to the original mining problem is the association rule maintenance problem. For example, a sales database is not a static one. From time to time, new records are appended to record new purchase activities. 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. We have to update the set of discovered association rules to reflect the changes in the underlying database. Thus, maintenance of discovered association rules is also an important problem.

The maintenance problem was first studied in [4], which proposed the FUP algorithm to find out new association rules when new transactions are appended to the database. Another algorithm FUP2, proposed in [5], is able to update the discovered association rules efficiently when transactions are added to, deleted from, or modified in the database. As with Apriori and DHP, the FUP and FUP2 algorithms find exact answers and they have to scan the old database several times. This hinders the performance of the database system in responding to other queries. The high overhead in rule maintenance thus induces an important problem of *how often* or *when* an update algorithm (such as FUP2) should be applied. As an extreme case, if we apply FUP2 too often, such as after each single transaction is added to the database, we will cripple the system to an extent that virtually no useful transactions can be executed. On the other hand, if we wait to accumulate a large number of updates before applying FUP2 onto the batch, we take the risk of using stale rules in between batch updates. Potentially harmful decisions might be made as a consequence. What is needed and is missing here is a strategy to determine when the batch of transaction updates has created a non-trivial distortion to the original set of association rules that a rule-update algorithm needs to be applied. The goal of this paper is to discover such a strategy.

We would like to remark that simple general rules such as "update the rules when x% of the transactions in the database have been changed" are hard to set and may not result in the best resource saving. The reason is that the *amount* of transaction update activity does not directly indicate when a rule update is warranted. It is the *content* of the update that counts.

[1] with respect to the huge size of the entire database.

In this paper, we propose an algorithm DELI (Difference Estimation for Large Itemsets) to determine when rule update should be applied. Instead of finding out the new set of association rules, or an approximation, in the updated database, the algorithm uses sampling techniques to estimate the *maximum amount of changes* in the set of rules due to the newly added transactions. The estimate gives us an indication of *when* the association rules need updating. If the estimate is small, then, so is the amount of changes in the original set of association rules. So, the old set of rules is still a good approximation to the new set of rules. It is not necessary to find out the new rules precisely. We simply wait until more changes are made to the database, and then reapply the DELI algorithm. Only when the estimate becomes large do we need to apply FUP2 to refresh the set of rules. Our experimental results show that DELI gives a very good upper bound on the amount of changes in the set of rules. Moreover, as we will see later in this paper, since DELI examines only a small portion of the database, it is much more efficient than FUP2. We thus can achieve great saving in the database resources. The saving can be used on serving other database queries, thus greatly increases the usability of the system.

In the remaining of this paper, a formal description of the problem is first given. Next, we briefly go through two previous algorithms, namely Apriori and FUP2, on which our new algorithm DELI is based. Then, a detailed description of our new algorithm is given. Following that, we present our experiments for studying the performance of DELI and the results are given. Comparisons with several other data mining algorithms employing sample are made. Finally, we discuss about some future directions and then conclude this paper.

2 Problem Definitions

In this section we give formal definitions of the *association rule mining problem*, the *rule maintenance problem*, and the *rule-set-change estimation problem*. These definitions will facilitate our description and analysis of the DELI algorithm.

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 $T \in D$ 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, denoted by σ_X, is defined to be the number of transactions in D that contain X. We say that an itemset X is *large*, with respect to a user-specified *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 has been shown to be reducible to the problem of finding all large itemsets for the same support threshold [1].

Thus, if $s\%$ is the given support threshold, the association rule 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.

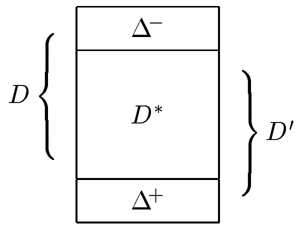

Figure 1: Definitions of D, D', Δ^-, D^* and Δ^+

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

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.[2]

For consistency, we again use σ_X to denote 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: $\sigma'_X = \sigma_X + \delta_X^+ - \delta_X^- = \sigma_X + \delta_X$.

For the update problem, L and σ_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 σ'_X $\forall X \in L'$ efficiently, given the knowledge of D, D', Δ^-, D^*, Δ^+, L and σ_X $\forall X \in L$.

2.3 Finding an upper bound on the size of the difference between the old and new association rules

Before actually doing the update to find L' and σ'_X $\forall X \in L'$, we want to know something about the size of the difference between L and L'. If the difference is small, then we do not update the set of 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 a rule update operation is necessary. To do this, we have

[2] 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.

to define a measure for the difference between L and L'.

But how should we measure the amount of changes in the set of association rules? Does it increase with the number of updates to the database? The answer is *no*. If the new transactions follow the same association rule patterns as the old database, then both the old and new database would exhibit the same association rules. In this case, however numerous the updates are, few useful new association rules will be found. An alternative would be to find out the new set of association rules in the updated database, and then compare it with the old set of rules to find out their *set symmetric difference*. The size of the symmetric difference is of course a good indicator for the necessity to discover the new set of association rules.

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 is the similarity between L and L'.

Note, however, that calculating the symmetric difference, or even just its size, requires us to find out the new set of association rules first, and this operation is itself expensive. So, instead of computing the symmetric difference exactly, we estimate its *size* by examining samples of the database.

It is clear that the ratio $\frac{|L \ominus L'|}{|L|+|L'|}$ can be used as a normalized measurement of the difference between L and L'. Its value varies between 0 and 1 and hence is more convenient when one needs to specify a threshold for this ratio. Unfortunately, the term $|L'|$ appears in the denominator, which we do not know; therefore, we cannot calculate the above ratio. So, we 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 $|L \ominus L'|$.

In this paper, instead of finding an accurate value for $|L \ominus L'|$, we take the approach to find an approximate *upper bound* on this value. The bound is approximate, but there should be *little*[3] chance that the amount of changes in the set of association rules is larger than the bound. 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 σ_X $\forall X \in L$.

3 Related Works

In this section, we briefly describe two previous algorithms that are related to our work. For details, the reader is referred to the corresponding references.

3.1 The Apriori Algorithm

The original problem of mining association rules (see Section 2.1) was first posed in [1]. The Apriori algorithm [2] is one of the most successful algorithms designed for solving the problem.

The Apriori algorithm finds out the large itemsets iteratively. In the k-th iteration, it finds out L_k, i.e. the set of all large itemset 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 [2] 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 are large

[3] w.r.t. the level of confidence.

$|D| = 10^6$ $\quad |\Delta^-| = 9000$ $\quad |\Delta^+| = 10000$ $\quad s\% = 2\%$

X	σ_X	δ_X^-	δ_X^+	σ'_X	in L?	in L'?
$\mathcal{A}$	24803	497	512	24818	Yes	Yes
$\mathcal{B}$	31305	690	823	31438	Yes	Yes
$\mathcal{C}$	24323	760	847	24410	Yes	Yes
$\mathcal{D}$	27887	192	185	27880	Yes	Yes
$\mathcal{E}$	21208	503	496	21201	Yes	Yes
$\mathcal{F}$	19887	120	137	19904	No	No
$\mathcal{AB}$	23766	80	99	23785	Yes	Yes
$\mathcal{AC}$	22302	92	98	22308	Yes	Yes
$\mathcal{AD}$	21188	113	135	21210	Yes	Yes
$\mathcal{AE}$	20100	194	103	20009	Yes	No
$\mathcal{BC}$	22321	108	137	22350	Yes	Yes
$\mathcal{BD}$	25086	172	153	25067	Yes	Yes
$\mathcal{BE}$	19803	202	436	20037	No	Yes
$\mathcal{CD}$	23847	125	152	23874	Yes	Yes
$\mathcal{CE}$	14467	480	438	14425	No	No
$\mathcal{DE}$	16782	168	152	16766	No	No
$\mathcal{ABC}$	21033	79	96	21050	Yes	Yes
$\mathcal{ABD}$	13744	58	90	13776	No	No
$\mathcal{ACD}$	20387	85	96	20398	Yes	Yes
$\mathcal{BCD}$	20211	87	120	20244	Yes	Yes

Table 2: Support counts for the examples

(i.e. $\in L_{k-1}$). Since this is a *necessary* condition for a k-itemset to be large [2], 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 lies in its use of the `apriori_gen` function to generate a small number of candidate itemsets.

Let us illustrate this algorithm with an example. Suppose we have a database D of 10^6 transactions with $I = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}\}$. Let us use a support count of 2%. Suppose the relevant support counts, which are unknown to us before we run Apriori, are as shown in Table 2. To find out the large itemsets in D, Apriori first generates the candidate set $C_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}\}$.[4] Then, it scans D to obtain the support counts of the itemsets in C_1. The counts are shown in Table 2. Next, Apriori finds that $\sigma_{\mathcal{F}} < |D| \times s\% = 20000$. So, $\mathcal{F}$ is not large. All other candidates are large. So, $L_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$. In the second iteration, Apriori computes $C_2 = \texttt{apriori\_gen}(L_1) = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{AE}, \mathcal{BC}, \mathcal{BD}, \mathcal{BE}, \mathcal{CD}, \mathcal{CE}, \mathcal{DE}\}$. Then, D is scanned to obtain the support counts of these itemsets. The resulting support counts are shown in Table 2. So, we have $L_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{AE}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$. In the third iteration, $C_3 = \texttt{apriori\_gen}(L_2) = \{\mathcal{ABC}, \mathcal{ABD},$

[4] We will write "$\mathcal{ABC}$" for the itemset $\{\mathcal{A},\mathcal{B},\mathcal{C}\}$ when no ambiguity arises.

$\mathcal{ACD}$, $\mathcal{BCD}\}$. Apriori scans D and the support counts given in the table. Thus, $L_3 = \{\mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$. Next, Apriori finds out that $C_4 = \texttt{apriori\_gen}(L_3) = \emptyset$. So, the algorithm terminates and we have $L = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{AE}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}, \mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$. The large itemsets and their support counts in D are thus found.

3.2 The FUP2 algorithm

The FUP2 algorithm [5] 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 σ_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 [5]. 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.

Let us trace how the algorithm works with an example continued from Section 3.1. Suppose 9000 original transactions in D are deleted ($|\Delta^-|$=9000) and 10000 new transactions are added ($|\Delta^+|$=10000). Please refer to Table 2 for the corresponding support counts in Δ^+ and Δ^-. In the first iteration, $P_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$ and $Q_1 = \{\mathcal{F}\}$. FUP2 scans the new transactions Δ^+ and deleted transactions Δ^- to find out the counts δ^+_X and δ^-_X, respectively, for all $X \in P_1 \cup Q_1$. For every itemset $X \in P_1$, FUP2 retrieves the old support count σ_X from the previous mining result, and calculate the counts σ'_X. All of them are large. So, they all go to L'_1. For $\mathcal{F} \in Q_1$, since it was not large in the database D, we know that its new count is at most $20000 + \delta^+_{\mathcal{F}} - \delta^-_{\mathcal{F}} = 20017 < |D'| \times s\% = 20020$. Therefore, $\mathcal{F}$ cannot be large, and it is pruned away from Q_1[5]. Thus, $L'_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$. In the second iteration, $P_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{AE}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$ and $Q_2 = \{\mathcal{BE}, \mathcal{CE}, \mathcal{DE}\}$. Next, FUP2 scans Δ^+ and Δ^- to find δ^+_X and δ^-_X for all $X \in C_2$. For those $X \in P_2$,

[5]The FUP2 algorithm actually checks whether $\delta^+_{\mathcal{F}} - \delta^-_{\mathcal{F}} \leq (|\Delta^+| - |\Delta^-|) \times s\%$. Since $\mathcal{F}$ satisfies this condition, it is pruned away.

FUP2 retrieves σ_X from the previous mining results and calculates σ'_X. The itemsets $\mathcal{AB}$, $\mathcal{AC}$, $\mathcal{AD}$, $\mathcal{BC}$, $\mathcal{BD}$, and $\mathcal{CD}$ are large. $\mathcal{AE}$ is no longer large in D'. For $\mathcal{CE} \in Q_2$, since $\delta^+_{\mathcal{CE}} - \delta^-_{\mathcal{CE}} = -42 \leq (|\Delta^+| - |\Delta^-|) \times s\% = 20$, it cannot be large. It is pruned away from Q_2. $\mathcal{DE} \in Q_2$ is also pruned away for the same reason. For the remaining candidate $\mathcal{BE}$ in Q_2, FUP2 scans D^* to obtain its old support count $\sigma_{\mathcal{BE}}$. Next, FUP2 computes the new count $\sigma'_{\mathcal{BE}} = 20037 \geq 20020 = |D'| \times s\%$. So, $\mathcal{BE}$ is large. Thus $L'_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{BE}, \mathcal{CD}\}$. In the third iteration, $P_3 = \{\mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$ and $Q_3 = \{\mathcal{ABD}\}$. FUP2 scans Δ^+ and Δ^- to find δ^+_X and δ^-_X for these candidates. The counts are as shown in Table 2. FUP2 calculates the new support counts of the itemsets in P_3 and finds that all of them are still large in D'. For itemset $\mathcal{ABD}$ in Q_3, FUP2 scans D^* to find out its old support count $\sigma_{\mathcal{ABD}}$. The new support count $\sigma'_{\mathcal{ABD}}$ is then calculated. It is found to be not large in D'. Therefore, $L'_3 = \{\mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$. In the next iteration $C_4 = \emptyset$. So, the algorithm terminates. The result is $L' = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{BE}, \mathcal{CD}, \mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$. The old large itemset $\mathcal{AE}$ is now obsolete and we have found a new large itemsets $\mathcal{BE}$.

4 The DELI algorithm

The FUP2 algorithm is an efficient way of maintaining an *accurate* set of association rules. Unfortunately, accuracy has its price. FUP2 still has to rescan the old database several times in order to obtain the support counts of new large itemsets. In this section, we present the DELI algorithm, which applies a sampling technique to estimate the support counts and then gives an approximate upper bound on how much changes in the set of association rules are introduced by the new transactions. If the changes are not significant, we can ignore them and wait until more are accumulated, contenting with the old set as a good approximation.

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[6] from D contains X is $p_X = \frac{\sigma_X}{|D|}$.

Now, suppose we randomly 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 $m \cdot p_X(1 - p_X)$. [16] 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

$$\begin{aligned} a_X &= \widehat{\sigma_X} - z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D| - \widehat{\sigma_X})}{m}} \\ b_X &= \widehat{\sigma_X} + z_{\alpha/2}\sqrt{\frac{\widehat{\sigma_X}(|D| - \widehat{\sigma_X})}{m}} \end{aligned} \tag{1}$$

and $z_{\alpha/2}$ is the critical value such that the area under the standard normal curve beyond $z_{\alpha/2}$ is exactly $\alpha/2$.[7] The value of α is chosen by the user. Typical values of $z_{\alpha/2}$ are 1.645 (for $\alpha = 0.10$), 1.960 (for $\alpha = 0.05$)

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

[7] 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. $\Pr(Z > z_{\alpha/2}) = \alpha/2$.

and 2.576 (for $\alpha = 0.01$). The $100(1-\alpha)\%$ confidence interval for σ_X has the property that

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

This means that there is $100(1-\alpha)\%$ chance that the actual value of σ_X lies within the interval $[a_X, b_X]$. 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 $(\widehat{\sigma_X})$ of σ_X can be obtained from a sample. Next, we will see how the confidence interval $[a_X, b_X]$ is employed. For efficiency, we will use the same sample S for every iteration of 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. Thus, the algorithm scans the old database D only once, so as to extract a sample.

4.2 Determining the sample size

In Section 4.1, we 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 a guideline for choosing a suitable size of m.

As we have shown above, the DELI algorithm uses the confidence interval to estimate σ_X *only when* X is not large in the original database D (see Section 4.3). So, for such X, $\sigma_X < |D| \times s\%$. This gives an upper bound on the possible values of σ_X when we do the estimation. Practical values for s is much less than 50. So, we can assume that $|D| \times s\% < \frac{|D|}{2}$. Now, observe also that the function $f(\widehat{\sigma_X}) = \widehat{\sigma_X}(|D| - \widehat{\sigma_X})$ is a quadratic polynomial in $\widehat{\sigma_X}$. It is increasing in the interval $(-\infty, \frac{|D|}{2})$ but decreasing in $(\frac{|D|}{2}, \infty)$. For practical values of s, $\widehat{\sigma_X} \approx \sigma_X \in [0, |D| \times s\%] \subseteq (-\infty, \frac{|D|}{2})$. Since the function $f(x)$ is increasing in this range, it is bounded above by $f(|D| \times s\%) = |D|^2 s\%(1 - s\%)$ for the practical range of s. Hence, the widths of all our confidence intervals are no more than

$$2z_{\alpha/2}\sqrt{\frac{|D|^2 s\%(1-s\%)}{m}} = 2|D|z_{\alpha/2}\sqrt{\frac{s\%(1-s\%)}{m}}$$

From this bound, it is obvious that the greater the size of the sample, the narrower the confidence intervals are, and hence the more accurate our estimates are. We can vary the sample size m to control the maximum width of the intervals. For example, suppose that we want the widths of the intervals not to exceed $\frac{|D| \times s\%}{5}$. We can establish the inequality

$$2|D|z_{\alpha/2}\sqrt{\frac{s\%(1-s\%)}{m}} \leq \frac{|D| \times s\%}{5}$$

and solve it to get a minimum value for m. For example, if $s = 2$ and $\alpha = 0.05$, then $z_{\alpha/2} = 1.96$ and solving the above inequality gives $m \geq 18823.84$. Note that this value is independent of the size of the database D. So, while D may contain billions of transactions, a sample of around 19 thousand transactions is large enough to give the accuracy desired in this example.

4.3 Giving an approximate upper bound on the amount of changes in large itemsets

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

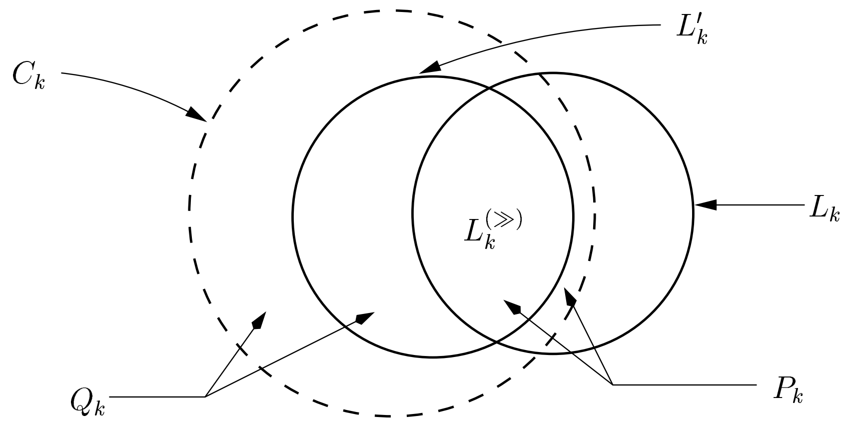

Figure 2: The relationship between various sets of itemsets

subsequent iterations ($k \geq 2$), C_k is calculated by applying the `apriori_gen` function [2] 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}; therefore, the candidate set C_k so generated may not cover all the itemsets in L'_k. However, we will see later in this section that statistically, misses are rare and the itemsets missed by the approximation $\widehat{L_{k-1}}$ most likely have very small support counts. Hence, candidates generated by them are unlikely to be large. This is supported by our experimental results (see Section 5). Therefore, C_k should cover most of the itemsets in L'_k, and the missed ones are unimportant (in the sense that they have low support counts). On the other hand, there may be false hits in $\widehat{L_{k-1}}$, but they would not generate too many false candidates in C_k, and even if they do so, the resulting false candidates are very likely to be pruned away in the subsequent steps because of their small support counts.

After its generation, the set C_k is partitioned in the same way as in FUP2: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. This is illustrated in 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 $L_k^{(\gg)}$.[8] 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 calculate the number of such itemsets and denote this number by η_k. The value of η_k will be used in computing the size of the symmetric difference $L \ominus L'$ (see Section 2.3).

For the candidates X in Q_k (i.e. those that were not large in the old database D), we do not know σ_X. However, we do know that $\sigma_X < |D| \times s\%$. We can apply the technique mentioned in [5] to prune away some candidates. We first find out δ_X during the scan of 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 [5], if $\delta_X \leq (|\Delta^+| - |\Delta^-|) \times s\%$, we have the new support count of X in the new database, $\sigma'_X = \sigma_X + \delta_X < (|D| + |\Delta^+| - |\Delta^-|) \times s\% = |D'| \times s\%$ and hence X cannot be large in D'. 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 scan the sample S and use the method as described in Section 4.1 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

[8]The superscript $(\gg)$ indicates that the support counts of the itemsets in $L_k^{(\gg)}$ are certainly larger than the threshold $|D'| \times s\%$.

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 $|D'| \times s\% < a_X + \delta_X$, the support threshold is below X's support count's confidence interval. In this case, we are very confident (to $100(1-\alpha)\%$) that X is large in the new database D'. We add X to the set $L_k^{(>)}$. (The set $L_k^{(>)}$ thus contains all the itemsets that were originally not large in D but are most probably large in the new database D'.)

2. If $a_X + \delta_X \leq |D'| \times s\% \leq b_X + \delta_X$, the support threshold lies within X's support count's confidence interval. In this case, we are not sure if X is large or not in D'. However, even if X is not large in D', since its support count is close to the threshold, we can say that X is *almost* large in D'. We add X to the set $L_k^{(\approx)}$. (The set $L_k^{(\approx)}$ thus contains all the itemsets that were originally not large but are either large or are close to be large in the new database D'.)

3. If $b_X + \delta_X < |D'| \times s\%$, i.e., the support threshold is above the confidence interval, we are very confident (to $100(1-\alpha)\%$) that X is not large in D'. We simply drop these candidates.

After all the candidates in Q_k are handled as described above, we calculate the value $\xi_k = \left|L_k^{(>)}\right| + \left|L_k^{(\approx)}\right|$. Note that an itemset in $L'_k - L_k$ (i.e., it was originally not large but becomes large in the new database) would be captured into either the set $L_k^{(>)}$ or the set $L_k^{(\approx)}$ with a very high probability. As our experiment results show (see Section 5), *misses* are very rare. Moreover, $L_k^{(\approx)}$ also captures a few itemsets that are not large (but are almost large) in D' (*false hits*). It is thus reasonable to use ξ_k as an upper bound of $|L'_k - L_k|$. Similar to η_k, we will use ξ_k to estimate the size of the symmetric difference $L_k \ominus L'_k$.

To complete the k-th iteration of the algorithm, we set $\widehat{L_k}$, the set that approximates the real set of large k-itemsets (L'_k) as $\widehat{L_k} = L_k^{(\gg)} \cup L_k^{(>)} \cup L_k^{(\approx)}$. The set $\widehat{L_k}$ should contain most of the large itemsets in L'_k, plus a few non-large itemsets that are almost large. Experimental results (see Section 5) reveal that the size of $L_k^{(\approx)}$ is usually very small. So, the number of false hits is very small. As we have argued, and will be shown in our experimental results, the number of large itemsets in L'_k that are missing in $\widehat{L_k}$ is even smaller. $\widehat{L_k}$ is thus a good approximation to L'_k.

After we complete the k-th iteration, we will check whether we should continue with the $(k+1)$-th iteration. Three criteria are considered.

The 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. As we have discussed, itemsets collected in $L_k^{(\approx)}$ may or may not be large in D'. Adding them to $\widehat{L_k}$ introduces potential error (and thus uncertainty) in our estimations. The higher this uncertainty is, the less reliable is the result. To measure this uncertainty, we compare the uncertainty factor $u_k = \frac{\left|L_k^{(\approx)}\right|}{\left|\widehat{L_k}\right|}$ against a user-specified threshold $\overline{u}$. If $u_k \geq \overline{u}$, then the amount of uncertainty introduced in this iteration is too much for DELI to reliably generate the candidates for the next iteration. So, we declare that a good estimation cannot be made and DELI stops. 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^+—the changes to the original database D. 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 $L_k - L'_k$ and 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 $|L_k \ominus L'_k| = |L_k - L'_k| + |L'_k - L_k|$. To determine whether this bound is so large that we need to discover the new set of association rules, 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 necessary. So, FUP2 should be invoked and DELI terminates.

The third criterion is basically the same as the termination condition of Apriori and FUP2. We check if $\widehat{L_k}$ is empty. If not, the algorithm continues with the next iteration. If it is empty, the algorithm terminates. Note that if the algorithm is terminated by this criterion, the values of u_k and d_k would not have exceeded their thresholds, since $u_k < \overline{u} \wedge d_k < \overline{d} \ \forall k$. So, we can conclude with *high certainty*, that the set of large itemsets in the new database is *not too different* from that in the old database. Hence, it is acceptable to take the set of large itemsets of the old database D as an approximation to that of the updated database D'. Below is a summary of the DELI algorithm.

Algorithm: DELI
Inputs: I, D, D^*, Δ^+, Δ^-, $s\%$, L, $\sigma_X \ \ \forall X \in L$, m, $z_{\alpha/2}$, $\overline{u}$, $\overline{d}$.
Output: a Boolean value indicating whether a rule-update operation is needed.
Requires: the `apriori_gen` function.

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, check the following conditions 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 $u_k = \frac{\left|L_k^{(\approx)}\right|}{\left|\widehat{L_k}\right|} \geq \overline{u}$, signal the need for a rule-update operation and then halt.
12. If $d_k = \sum_{j=1}^{k} (\eta_j + \xi_j)/|L| \geq \overline{d}$, signal the need for a rule-update operation and then halt.
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' and halt.

Note that the DELI algorithm scans the original database D only 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 should be small in practice. So, the DELI algorithm has very little I/O overhead. The algorithm takes I, D^*, Δ^+, Δ^-, $s\%$, L, $\sigma_X \ \forall X \in L$, m, $z_{\alpha/2}$, $\overline{u}$ and $\overline{d}$ as input, and generates the Boolean output of whether the new association rules need to be discovered. If the algorithm reports that it is necessary, then we can use the FUP2 algorithm to find out the new rules. Otherwise, we can take L as an approximation of L'. In this latter case, we have run DELI instead of FUP2. Since running DELI is less expensive than running FUP2, as revealed by our experimental results, we have saved machine resources.

4.4 An improvement

We can increase the accuracy of estimation and the performance of DELI by storing the support counts of all 1-itemsets, i.e. $\sigma_X \ \forall X \in I$. If these counts are stored during a previous mining, then in the first iteration of DELI, we need not estimate the values of σ_X. Instead, these counts can be retrieved. Then, DELI can find out the counts σ'_X accurately for all 1-itemsets X without scanning the database D or the sample S. Consequently, L'_1 can be accurately determined and hence C_2 will include all itemsets in L'_2. This can further increase the accuracy of the estimations in iteration 2. The only extra cost is the extra storage, which has a size of $O(|I|)$. This is both affordable and scalable.

4.5 An example

Let us illustrate the DELI algorithm with an example continued from Section 3.2. The support counts are again as shown in Table 2. Suppose that we use a sample size $m = 10000$ and we set $\alpha = 0.05$, i.e. $z_{\alpha/2} = 1.96$. We set the thresholds $\overline{u} = \overline{d} = \frac{1}{5}$. Suppose also that the support counts of the 1-itemsets in D have been stored as described in Section 4.4.

In the first iteration, we have $C_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}\}$. Since the old support counts are all stored, we can directly retrieve them and update them by scanning Δ^- and Δ^+. So, we can find the new set of large itemsets $L'_1 = \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$. We assign $\widehat{L_1} = L'_1$. Since no approximations are made in this iteration, we have $u_1 = 0$. Moreover, since $L_1 = L'_1$, we have $\eta_1 = \xi_1 = 0$ and hence $d_1 = 0$. Further, $\widehat{L_1}$ is non-empty. So, the algorithm goes on with the second iteration.

In the second iteration, C_2 is generated from $\widehat{L_1}$. It contains those ten 2-itemsets shown in Table 2. After partitioning, we have $P_2 = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{AE}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$ and $Q_2 = \{\mathcal{BE}, \mathcal{CE}, \mathcal{DE}\}$. For the itemsets in P_2, the stored old support counts are retrieved and updated against Δ^- and Δ^+. Of these seven itemsets, $\mathcal{AE}$ is no longer large, because $\sigma'_{\mathcal{AE}} = 20009 < 20020 = |D'| \times s\%$. All the other six are still large and hence they are put into $L_2^{(\gg)}$. Therefore, $L_2^{(\gg)} = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{CD}\}$ and thus $\eta_2 = 1$. For the itemsets in Q_2, $\mathcal{CE}$ and $\mathcal{DE}$ are pruned, because $\delta^+_{\mathcal{CE}} - \delta^-_{\mathcal{CE}} = -42 \le (|\Delta^+| - |\Delta^-|) \times s\%$ and $\delta^+_{\mathcal{DE}} - \delta^-_{\mathcal{DE}} = -16 \le (|\Delta^+| - |\Delta^-|) \times s\%$. The itemset $\mathcal{BE}$, however, is not pruned away. So, we examine S for this itemset. Suppose the support count of $\mathcal{BE}$ in S is 202. Then, $\widehat{\sigma_{\mathcal{BE}}} = 20200$ and hence according to equations 1, the 95% confidence interval for $\sigma_{\mathcal{BE}}$ is 20200 ± 2757. Since $\delta^+_{\mathcal{BE}} = 436$ and $\delta^-_{\mathcal{BE}} = 202$, the corresponding confidence interval for $\sigma'_{\mathcal{BE}}$ is $[17677, 23191]$. As $20020 = |D'| \times s\%$ falls within this interval, $\mathcal{BE}$ is inserted into $L_2^{(\approx)}$. $L_2^{(>)}$ remains empty. Thus, $\xi_2 = 1$. We have $\widehat{L_2} = \{\mathcal{AB}, \mathcal{AC}, \mathcal{AD}, \mathcal{BC}, \mathcal{BD}, \mathcal{BE}, \mathcal{CD}\}$. Since $u_2 = \frac{1}{7} < \frac{1}{5} = \overline{u}$ and $d_2 = \frac{2}{15} < \frac{1}{5} = \overline{d}$, the algorithm continues into the third iteration.

In the third iteration, $P_3 = \{\mathcal{ABC}, \mathcal{ACD}, \mathcal{BCD}\}$ and $Q_3 = \{\mathcal{ABD}\}$. The old support counts for the itemsets in P_3 are again retrieved and then updated against Δ^- and Δ^+. All of them remain large and hence are inserted into $L_3^{(\gg)}$. So, $L_3^{(\gg)} = P_3$ and $\eta_3 = 0$. For the itemset $\mathcal{ABD}$ in Q_3, we obtain its count in the sample S. Suppose the values is 155. Then, we get the interval $[13111, 17953]$ for $\sigma'_{\mathcal{ABD}}$. The interval is below 20020 and hence $\mathcal{ABD}$ is dropped. This gives $L_3^{(\approx)} = L_3^{(>)} = \emptyset$, $\xi_3 = 0$ and $\widehat{L_3} = P_3$. Thus $u_3 = 0 < \overline{u}$ and $d_3 = \frac{2}{15} < \overline{d}$. So, the algorithm continues into iteration 4. However, $C_4 = \texttt{apriori\_gen}(\widehat{L_3}) = \emptyset$. As a result, DELI terminates and concludes that $L \approx L'$. So, we can confidently use L as an approximation of L'.

Note that in our example, the values of u_k have remained very small. So, the estimations are quite reliable. Moreover, comparing the estimations $\widehat{L_k}$ to the actual sets L'_k (which are not accurately found by DELI), we can note that they are the same. So, although DELI is doing approximations, it can yield results quite close to the actual answer. By using the sample S, we have avoided repeated scanning of the unchanged transactions D^*. We only have to scan D once in order to collect the sample S.

5 Experimental Results

To assess the performance of DELI, we have implemented the algorithms Apriori, FUP2 and DELI on an RS/6000 workstation (model 410) running AIX. The improvement described in Section 4.4 has been incorporated into our implementation of DELI. We have done a series of experiments to study the performance of the new algorithm and compare its performance with the other algorithms.

In the experiments, we follow the approach in [2, 5, 12] and use synthetic data as the input databases to the algorithms. We will briefly explain the data generation method in Section 5.1. In each experiment, we first generate the required transaction databases D, Δ^+ and Δ^-. Then, we use Apriori to find out the large itemsets L. Next, FUP2 is invoked to find out L', and the time taken is noted. After that, we run the DELI algorithm, and note the time taken. For experimental purpose, we have set the thresholds $\overline{u}$ and $\overline{d}$ to infinity. This is to "switch off" the first two termination criteria and hence make the algorithm always terminate by the third criterion. Then, the maximum values of u_k and d_k are noted. We denote these maximum values by $\overline{u}_c$ and $\overline{d}_c$, respectively. Note that if we had set the threshold $\overline{u}$ to any value above $\overline{u}_c$, then it would never be exceeded by any u_k because $\overline{u}_c$ is the maximum value of u_k $\forall k$. However, if we had set the threshold $\overline{u}$ to any value below or equal to $\overline{u}_c$, then there would be at least one $u_k \geq \overline{u}$. In that case, DELI would signal a need for an update, because of the first criterion. So, $\overline{u}_c$ is the *critical* value for the threshold $\overline{u}$. If the threshold is set above the critical value, the uncertainty factors u_k will not trigger the signal for update. If the threshold is below the critical value, the uncertainty factors will trigger the signal. A similar relationship holds for $\overline{d}$ and $\overline{d}_c$. Thus, it is much more useful to examine the critical values $\overline{u}_c$ and $\overline{d}_c$ rather than the Boolean output of DELI.

For each of the experiment, we compare the value of $\overline{d}_c$, which gives an approximate upper bound on $\frac{|L \ominus L'|}{|L|}$, against the actual value of $\frac{|L \ominus L'|}{|L|}$, which is calculated by comparing L with L'. We give the ratio of the former value to the latter. This is an indication of how good DELI's bounds are. The closer to 1 this ratio, the better the bounds. However, since the d_k's are *approximate* upper bounds on $\frac{|L \ominus L'|}{|L|}$, we expect the ratio to be above 1 most of the time. A value of the order of magnitude of 1 is acceptable for $\overline{d}_c$. A value of $\overline{d}_c$ closer to unity is of course more favourable. The value of $\overline{u}_c$, on the other hand, gives us the level of uncertainty of DELI's estimations. Since $\overline{u}_c$ is the maximum value of the uncertainty factors u_k over all iterations, it bounds the factors from above. The smaller this critical value, the smaller the amount of uncertain information that DELI is relying on (because u_k is directly proportional the size of the uncertain set $L_k^{(\approx)}$) and hence the more reliable DELI's conclusion (on whether FUP2 should be invoked) is. The algorithm guarantees that the need for a rule-update operation is signaled when the uncertainty factor of any iteration exceeds the threshold $\overline{u}$.

In addition, we compare the time taken by FUP2 against the time taken by DELI in each experiment. The ratio of the former to the latter is plotted. We call this the "speedup" ratio because it measures how much faster DELI is. Note that the time taken by DELI in our experiments is the *maximum* time taken for the same experimental settings, with the maximum taken over all possible threshold values of $\overline{u}$ and $\overline{d}$. This is because for other threshold values, the algorithm may be terminated by the first or the second criterion, causing the algorithm to stop *earlier*. So, for practical values of $\overline{u}$ and $\overline{d}$, DELI may take less time to finish. In that case, the speedup ratio is even higher. So, the speedup ratios measured in our experiments are values for the worst cases. It is important to note that while FUP2 finds exact values for L and $\sigma'_X \ \ \forall X \in L'$, DELI only gives an approximate upper bound on $\frac{|L \ominus L'|}{|L|}$. They compute different things and we should not directly compare their speeds. Nevertheless, the speedup ratio shows the amount of savings, in terms of machine resources, that we gain if we run DELI instead of FUP2. In practice, DELI will not signal for the need of using FUP2 very often. Suppose that on average, the need for using FUP2 is signaled once in every n runs of DELI, then the fraction of machine resources saved would be $1 - \frac{1}{n} - \frac{1}{\text{speedup ratio}}$. This saving is positive when $n > 1$ and speedup > 1. Moreover, when n is large, this saving approaches $1 - \frac{1}{\text{speedup ratio}}$, which could be substantial.

5.1 Generation of synthetic databases

In the following experiments, we use synthetic databases. The synthesis procedure was introduced in [2] and subsequently modified in [5, 12]. The reader is referred to these papers for detailed explanation of the procedure. Below, we go through the procedure briefly.

To generate the transaction database D^* and the transactions in Δ^+ and Δ^-, we first generate a pool $\mathcal{L}$ of potentially large itemsets. Each itemset in $\mathcal{L}$ is generated by first determining the itemset size from a Poisson distribution with mean $|\mathcal{I}|$; then, the itemset is filled with some items from the previous itemset, and the remaining slots are filled by randomly picking from N distinct items. After $\mathcal{L}$ is generated, it is used to generate the database D^*. Each transaction in D^* first has its size determined from a Poisson distribution with mean $|T|$. Next, a random itemset from $\mathcal{L}$ is chosen and its items are added to the transaction being generated. If the transaction has acquired the desired size, we're done and we go to generate the next transaction. If not, we pick another itemset from $\mathcal{L}$ and repeat the above until the transaction has got the desired number of items. In this way, transactions in D^* are generated. The transactions in Δ^+ are generated similarly, except that they are generated from only a subset of p itemsets in $\mathcal{L}$, rather than the whole $\mathcal{L}$. This is to make Δ^+ contain large itemsets not identical to those of D. Similarly, the transactions in Δ^- are generated from only a subset of q itemsets in $\mathcal{L}$, rather than the whole $\mathcal{L}$. Hence, we can model the change of association rules when new transactions are added.

Following [5], we have set the parameters of the data generation procedure as follows: $N = 1000$, $|\mathcal{I}| = 4$, $|T| = 10$, $|\mathcal{L}| = 2000$, and $p = q = 1500$. In these generated databases, our measurements show that $\frac{|L \ominus L'|}{|L|}$ ranges from 1% to 18% of the size of L. In other words, 1–18% of the large itemsets are changed by the updates.

5.2 Effects of the sample size

The first experiment is to study the effect of the sample size m on the behaviour of DELI. We fix $|D| = 5000$, $|\Delta^+| = |\Delta^-| = |D| \times 5\% = 5000$, $s\% = 2\%$ and $z_{\alpha/2} = 1.96$ ($\alpha = 0.05$). As explained in Section 4.2, if we want the maximum width of the confidence not to exceed $\frac{|D| \times s\%}{5}$, we should set m to around 19 thousands. In this experiment, we vary m between 5000 and 30000 to see its effect on DELI. We use the same databases D^*, Δ^+ and Δ^- for all these values of m. So, the value of $\frac{|L \ominus L'|}{|L|}$ remains the same for all values of m in the experiment. The results are plotted in Figure 3.

The speedup ratio is between 2.2 and 4.0 (see Figure 3). It can be observed from the figure that the speedup ratio decreases as m increases. This is expected: The speed of FUP2 is not affected by the sample size. However, as the sample size increases, DELI has to examine more and more transactions. So, it consumes more time and hence the speedup ratio decreases. From the same graph, it can be observed that the critical value $\overline{d}_c$ varies between 1.1 and 1.5 times the actual value of $\frac{|L \ominus L'|}{|L|}$. This means the approximate upper bounds given by DELI are reasonably good. The ratio $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ comes closer to 1 as m increases. This is because with a larger sample, DELI can make better estimations and hence the bounds are closer to the actual values. This observation confirms the results of our analysis in Section 4.2: The accuracy of the estimations can be improved by using a larger sample. The price to pay is the increase in the amount of

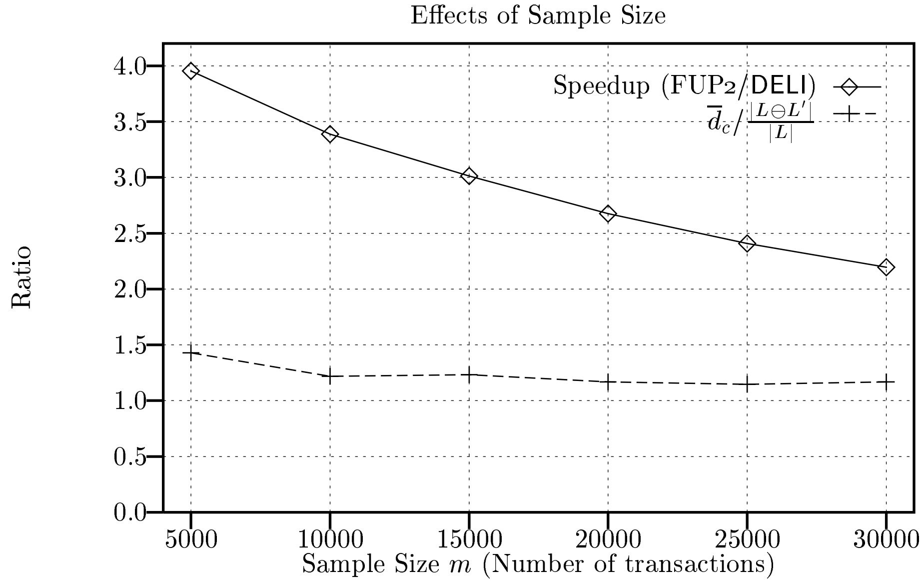

Figure 3: Effects of the sample size m

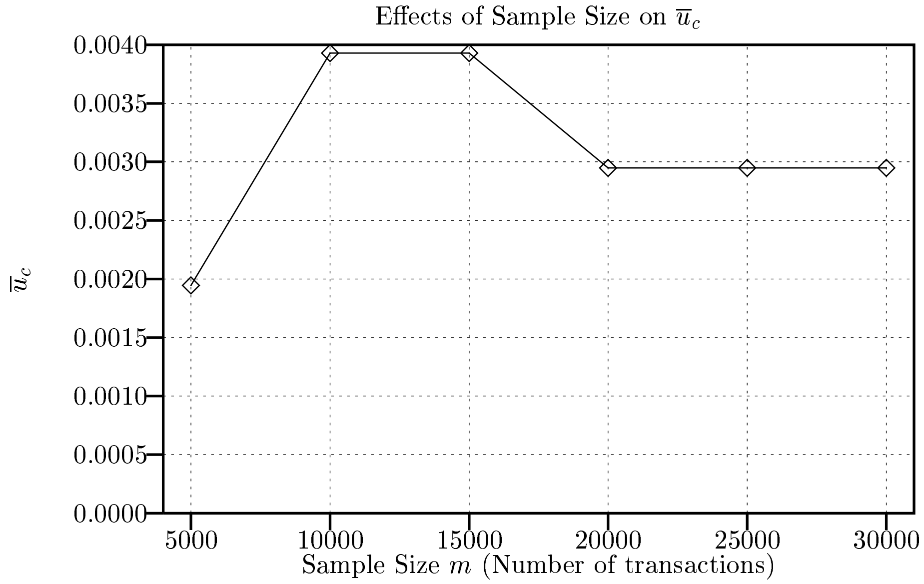

Figure 4: Effects of the sample size m on $\overline{u}_c$

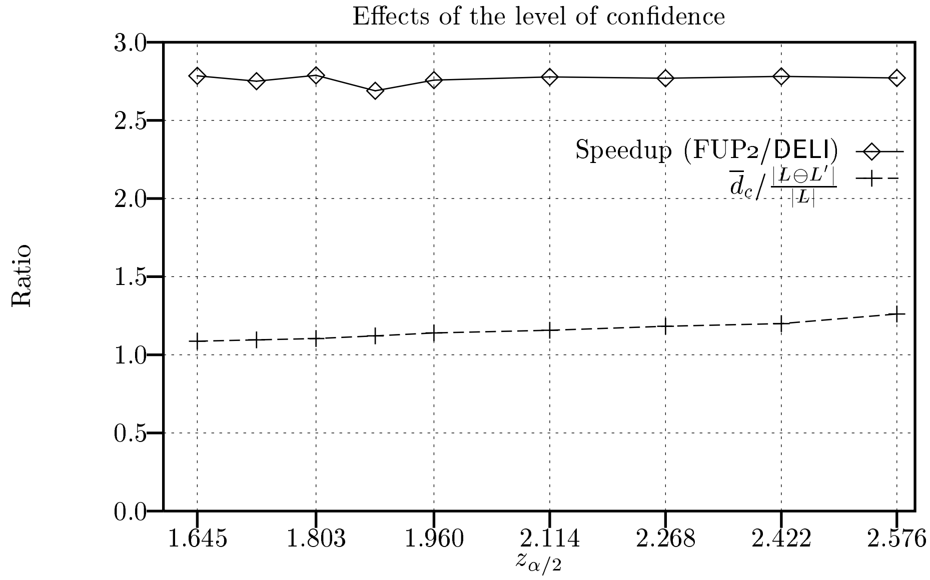

Figure 5: Effects of the level of confidence

time taken. So, we have a tradeoff between machine resource and accuracy. The critical value $\overline{u}_c$ is plotted against the sample size in Figure 4. Its value remains below 0.0040 with minor fluctuations. This is a very small value. So, the results of DELI are very reliable. In each iteration, no more than 0.4% of the new large itemsets would be false hits.

Thus, very reliable conclusions can be made by DELI even when the sample size is relatively small (w.r.t. $|D|$). Higher reliability can be achieved by using a larger sample, at the expense of consuming more machine resources.

5.3 Effects of the level of confidence

Our next experiment is to study the effects of α on the performance of DELI. We fix $|D| = 100000$, $|\Delta^+| = |\Delta^-| = 5000$, $s\% = 2\%$, $m = 20000$ and vary the value of $z_{\alpha/2}$ between 1.645 (for $\alpha = 0.10$) and 2.576 (for $\alpha = 0.01$). The same databases Δ^-, D^* and Δ^+ are used throughout this experiment. Hence, the value of $\frac{|L \ominus L'|}{|L|}$ remains the same for all values of $z_{\alpha/2}$ in the experiment. The results are shown in Figures 5 and 6.

Figure 5 shows that the value $\overline{d}_c$ slowly increases with $z_{\alpha/2}$. This is because at greater values of $z_{\alpha/2}$, the level of confidence $100(1-\alpha)\%$ increases and hence DELI is making more conservative estimations. The resulting critical value $\overline{d}_c$ is larger. So, we should have a larger chance of having $\overline{u}$ and $\overline{d}$ exceeded. A similar explanation applies for the increase of $\overline{u}_c$ against $z_{\alpha/2}$ as revealed in Figure 6. However, the level of confidence has no significant effects on the speedup ratio (see Figure 5).

This result suggests that we can make the estimations more conservative by setting $z_{\alpha/2}$ to higher values (i.e. lower values of α). This makes DELI generate signals for rule-update operations earlier and hence more frequently. This would consume more resources, but in return, we will accurately find out the new rules earlier. On the other hand, if we can tolerate a larger change in the set of association rules, we can use a smaller value of $z_{\alpha/2}$. Then, DELI would generate the update signals less frequently, and hence more machine resources can be saved.

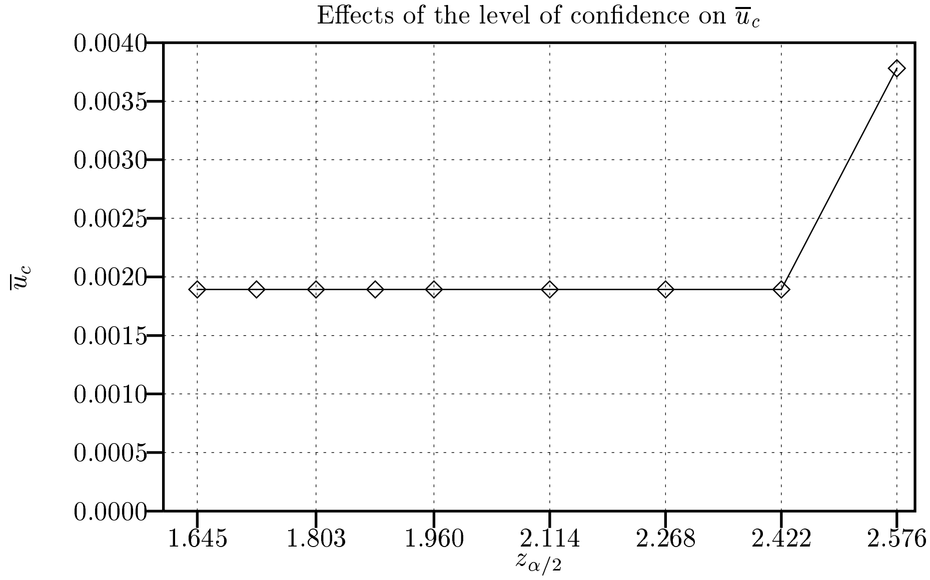

Figure 6: $\overline{u}_c$ against level of confidence

5.4 Effects of the size of updates

This experiment is to find out how the size of updates, i.e. $|\Delta^+|$ and $|\Delta^-|$, affects the performance of DELI. We set the parameters as follows: $|D| = 100000$, $s\% = 2\%$, $z_{\alpha/2} = 1.96$, $m = 20000$. We vary the value of $|\Delta^+| = |\Delta^-|$ from 5000 to 20000.

Figure 7 shows that the speedup ratio drops when the size of Δ^+ and Δ^- increases. This is because a major advantage of DELI over FUP2 is that it inspects a small sample of D instead of the entire D. However, both algorithms examine all the transactions in Δ^+ and Δ^-. So, the advantage of DELI diminishes as $|\Delta^+| = |\Delta^-|$ increases. Nevertheless, the maximum value of $|\Delta^+|$ and $|\Delta^-|$ that we use in the experiment is 20% of that of $|D|$ and in that case, the speed up ratio is 1.75 (i.e. we still save 75% of the system resources). Practically, DELI is applied to cases where $|\Delta^+|$ and $|\Delta^-|$ are much smaller. So, for practical applications, the speedup is significant.

On the same figure, it can be observed that the ratio $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ decreases as $|\Delta^+|$ and $|\Delta^-|$ increase. This is because the DELI algorithm gets information about the new database D' from two sources, namely the sample S and the new transactions in Δ^+ and Δ^-. The information derived from the former source are estimations and hence inaccurate, while the latter source provides accurate information. In this experiment, we fixed the sample size m. So, the amount of inaccurate information that DELI gets remains the same. Meanwhile, the amount of accurate information that DELI gets increases as $|\Delta^+|$ and $|\Delta^-|$ increase. Consequently, the accuracy of estimations increases. Thus, the ratio $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ approaches 1 as $|\Delta^+|$ and $|\Delta^-|$ increase. For similar reasons, the value of $\overline{u}_c$ (see Figure 8) decreases from 0.012 to 0.006 as $|\Delta^+|$ and $|\Delta^-|$ increases from 5000 to 20000.

Hence, for practical values of $|\Delta^+|$ and $|\Delta^-|$, where they are small, the speedup of DELI over FUP2 is significant. Moreover, even for these values of $|\Delta^+|$ and $|\Delta^-|$, the values of $\overline{u}_c$ and $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ are low enough so that DELI would not unnecessarily signal for the need of using FUP2. Thus, practically, DELI is quite efficient and it seldom generates the signal for a rule-update operation when it is unnecessary.

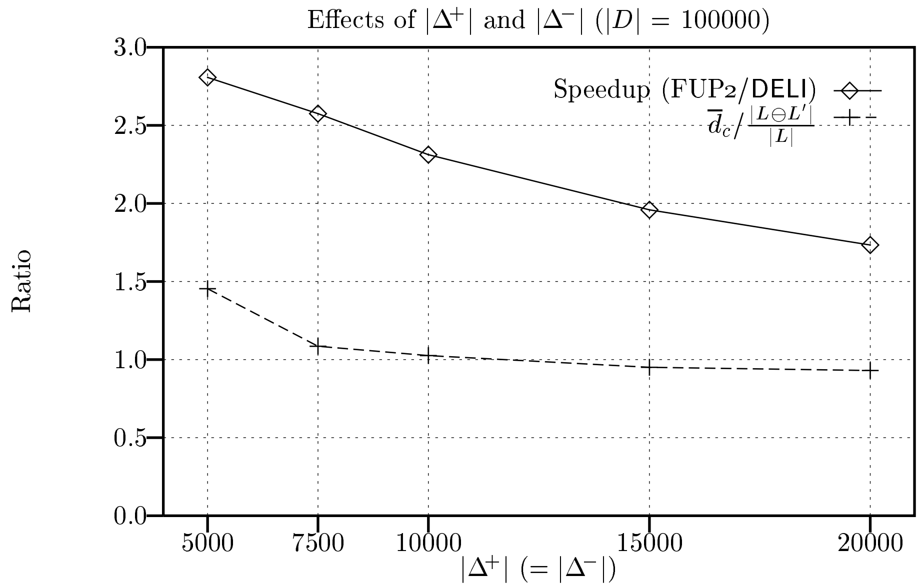

Figure 7: Effects of $|\Delta^+|$ and $|\Delta^-|$

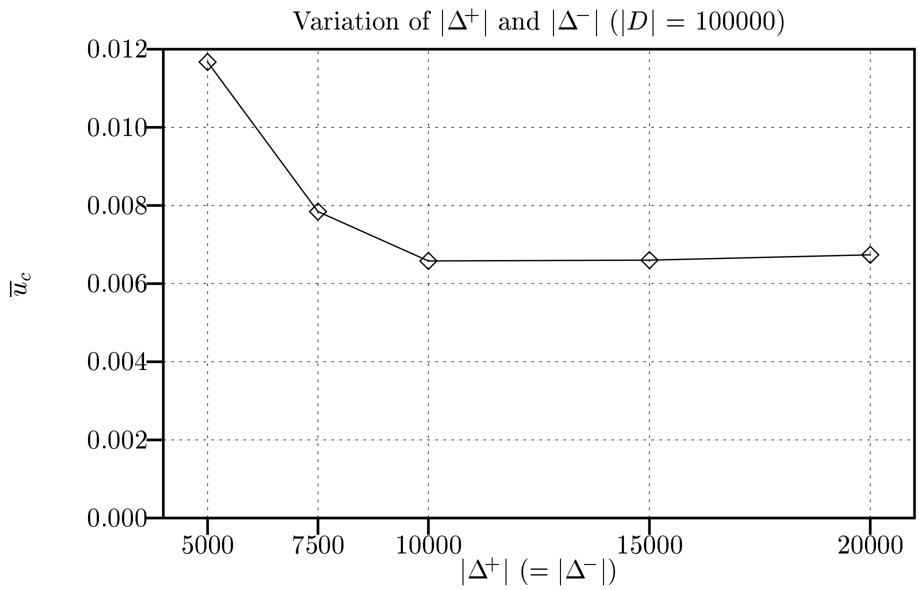

Figure 8: Effects of $|\Delta^+|$ and $|\Delta^-|$ on $\overline{u}_c$

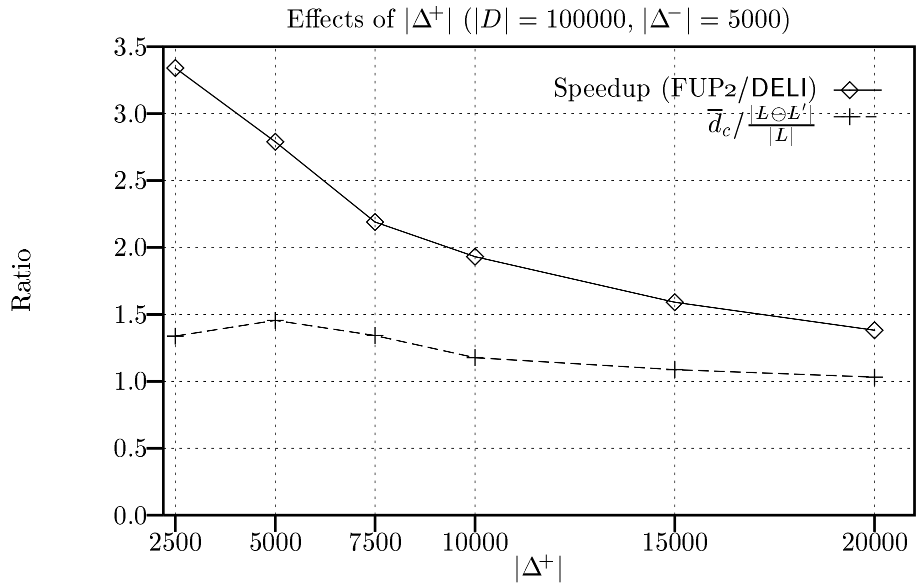

Figure 9: Effects of $|\Delta^+|$

5.5 Effects of the size of Δ^+

The above experiment only shows how DELI performs when $|\Delta^+| = |\Delta^-|$. To see how the size of Δ^+ alone would affect the performance, we do another experiment. We set the parameters as follows: $|D| = 100000$, $|\Delta^-| = 5000$, $s\% = 2\%$, $z_{\alpha/2} = 1.96$, $m = 20000$. We vary the value of $|\Delta^+|$ from 2500 to 20000.

From Figure 9, it can be observed that the speedup ratio drops as $|\Delta^+|$ increases. In fact, when the size of Δ^+ increases, the execution times of both FUP2 and DELI increases. This is because both FUP2 and DELI has to spend more time to examine a larger Δ^+. However, the increase in execution time of DELI is more significant. This is because the major saving of DELI over FUP2 is that it scans D^* only once. DELI still has to scan Δ^+ multiple times to extract information. So, the saving of DELI over FUP2 diminishes as Δ^+ increases in size relative to D^*. This increase in execution time, however, causes the estimations made by DELI to be more and more accurate as $|\Delta^+|$ increases. This is because DELI only does estimations for the transactions in D. It extracts exact information from Δ^+. So, as Δ^+ increases in size, DELI gets more and more accurate information, thus making more and more accurate deductions. This explains why the curve of $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ approaches 1.0 as $|\Delta^+|$ increases.

Figure 10 is a plot of $\overline{u}_c$ against $|\Delta^+|$. No simple relationship between these two quantities can be derived from the graph. However, we can see from the graph that the value of $\overline{u}_c$ is below 0.012, which is very low. So, although we cannot observe any trends of $\overline{u}_c$ as $|\Delta^+|$ is varied, we know that the level of uncertainty of DELI is very low.

5.6 Effects of the size of Δ^-

For the sake of completeness, a complementary experiment is performed to see how the size of Δ^- alone would affect the performance of the algorithm. The parameters are set as follows: $|D| = 100000$, $|\Delta^+| = 5000$, $s\% = 2\%$, $z_{\alpha/2} = 1.96$, $m = 20000$. We vary the value of $|\Delta^-|$ from 2500 to 20000.

We can see from Figure 11 that the speedup ratio initially increases as $|\Delta^-|$ increases but eventually decreases for large values of $|\Delta^-|$. The initial increase is caused by the amount of certain information that DELI can get. With small values of $|\Delta^-|$, DELI cannot get too much certain information, since it only get

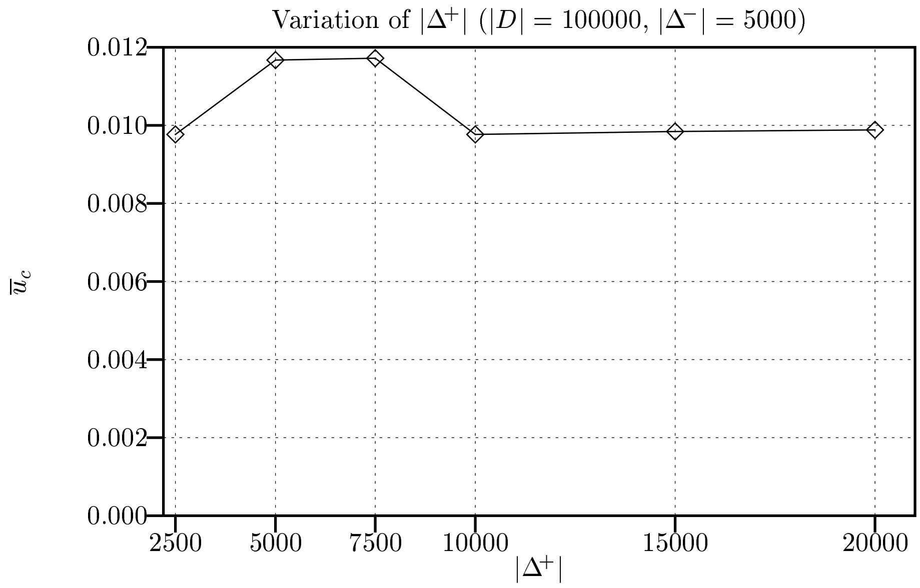

Figure 10: Effects of $|\Delta^+|$ on $\overline{u}_c$

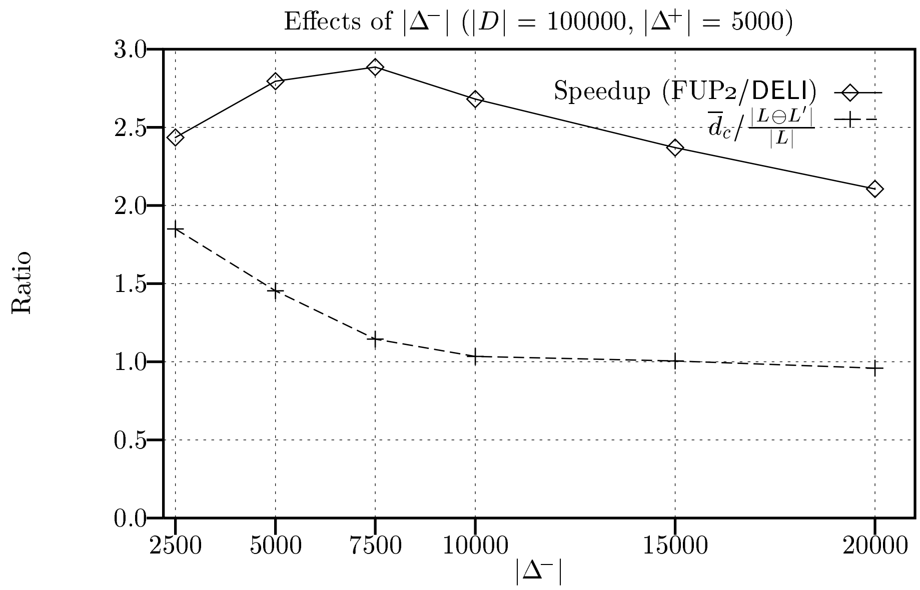

Figure 11: Effects of $|\Delta^-|$

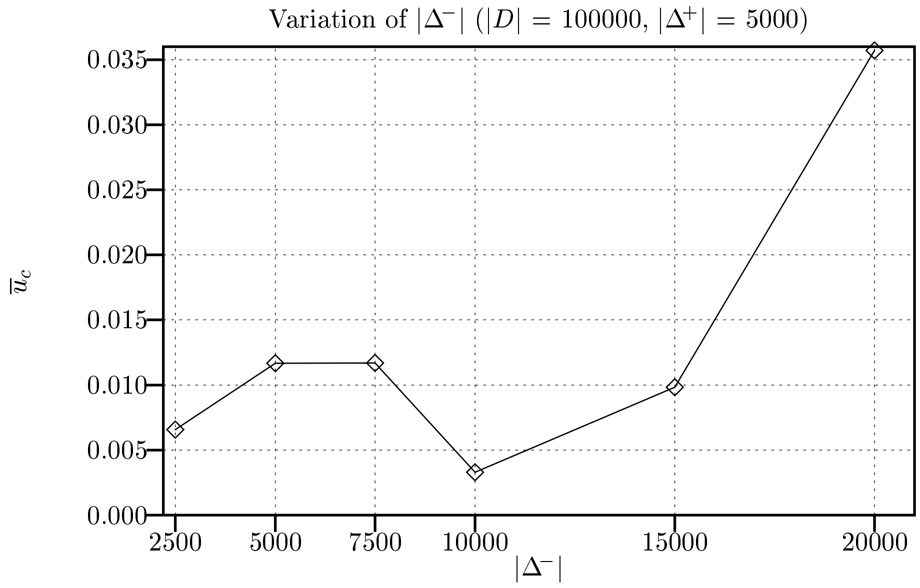

Figure 12: Effects of $|\Delta^-|$ on $\overline{u}_c$

exact counts from Δ^- and Δ^+. As a result, DELI cannot make very reliable estimations. Yet, it tends to be conservative. So, as $|\Delta^-|$ decreases over ranges of small values, DELI gets less and less accurate information. To be conservative, it generates more and more candidates, and hence it has to spend more and more time on obtaining support counts from Δ^- and Δ^+. Thus, the speedup decreases as $|\Delta^-|$ decreases. However, for larger $|\Delta^-|$, this effect starts to diminish. Once $|\Delta^-|$ is large enough, DELI can become quite confident with its estimations. Then, another effect starts to dominate in determining the speedup ratio. This effect is caused by the increase in the amount of transactions in Δ^- that both FUP2 and DELI has to examine. As mentioned in the previous subsection, DELI's speedup over FUP2 is mainly attributed to the examination of the sample S instead of D. Both algorithms still has to examine Δ^- several times. So, the saving resulting from the use of sampling in DELI diminishes as $|\Delta^-|$ increases. Thus, the speedup ratio drops as $|\Delta^-|$ increase when $|\Delta^-|$ is large enough. Anyway, the speedup ratio remains above 2.0 in the experiments. So, DELI runs faster than FUP2 in all the cases in the experiment. For the same reasons given in the last subsection, the estimations made by DELI increases in accuracy as $|\Delta^-|$ increases. This is exhibited by the curve of $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ in Figure 11.

Figure 12 shows how the value of $\overline{u}_c$ varies as $|\Delta^-|$ is varied. The roughly observed trend is that $\overline{u}_c$ increases as $|\Delta^-|$ increases. The explanation is that the larger the size of Δ^-, the smaller the similarity between D and D' and hence L and L' and hence the less reliable our estimations for the updated database D' derived from the information of the old database D. So, the certainty level of DELI decreases as $|\Delta^-|$ increases.

5.7 Effects of the support threshold

The next experiment shows the effects of the support threshold $s\%$. We set $|D| = 100000$, $|\Delta^+| = |\Delta^-| = 5000$, $z_{\alpha/2} = 1.96$, $m = 20000$ and vary the value of $s\%$ between 1.0% and 3.0%. The results are shown in Figure 13.

The figure does not reveal any general trend of the performance of DELI with respect to the support threshold. Neither do we find a simple pattern for $\overline{u}_c$ (Figure 14), which remained below 0.025. No simple relationships between these quantities and $s\%$ can be observed. This is because the size of the symmetric difference $L \ominus L'$ has no direct relationship with the sizes of L and L', even though we know that $|L|$ and $|L'|$

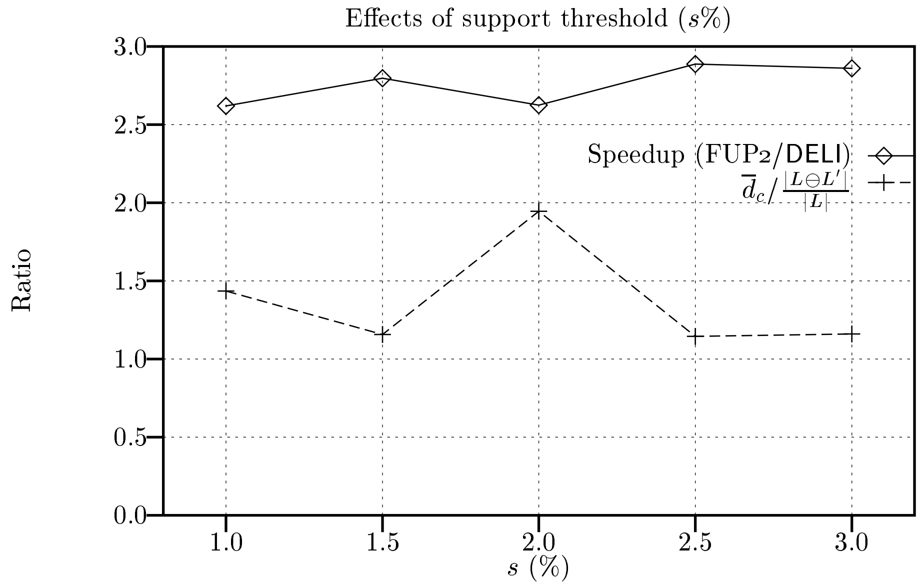

Figure 13: Effects of s%

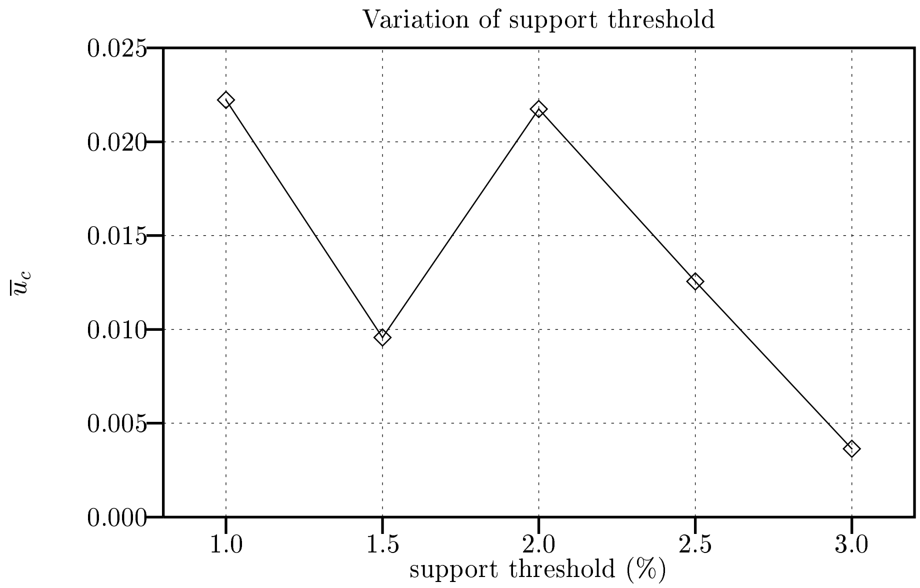

Figure 14: Effects of s% on $\overline{u}_c$

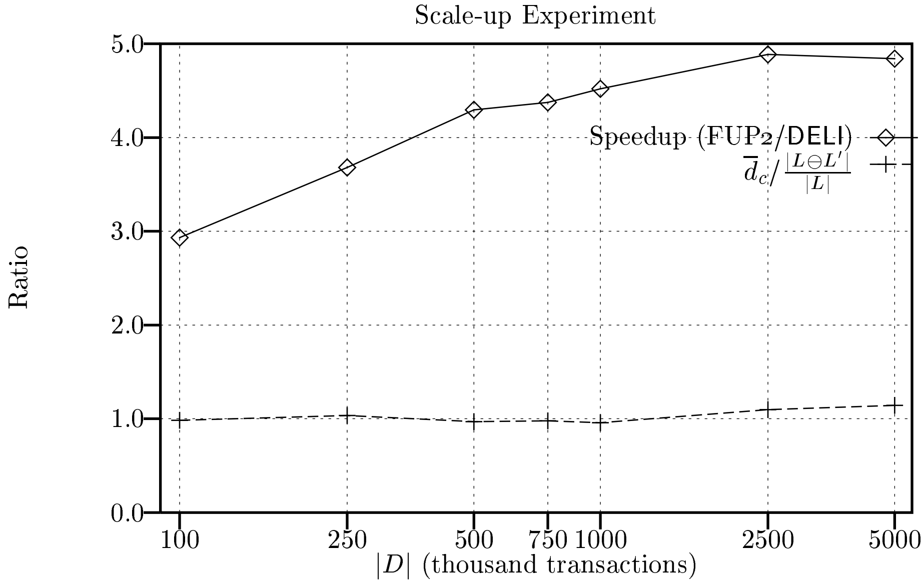

Figure 15: Scale-up Experiment

decreases as $s\%$ increases. Hence, we believe that the value of $s\%$ does not have any simple relationships with the performance of DELI.

5.8 Scale-up Experiment

We want to know if DELI still performs well when the size of the databases becomes orders of magnitudes larger than the above experiments. So, in this experiment, we increase $|D|$ from 100 thousand gradually up to 5000 thousand. The size of Δ^+ and Δ^- are increased accordingly, so that $|\Delta^+| = |\Delta^-| = |D| \times 5\%$ in each case. Other parameters are set as follows: $s\% = 2\%$, $z_{\alpha/2} = 1.96$ and $m = 20000$. The results are shown in Figures 15 and 16.

It is obvious from the figure that the speedup factor increases significantly as $|D|$, $|\Delta^+|$ and $|\Delta^-|$ increase. We account this to the I/O overhead of FUP2 and the effect of disk-caching of the operating system. Recall that FUP2 has to scan the old database D several times, while DELI does that only once. So, as the size of D increases, the I/O overhead of FUP2 increases rapidly. Also, for very large values of $|D|$, the disk-cache system will have a lot of cache misses during the scan of D. This effect is amplified by FUP2, which scans D several times.

Clearly, DELI performs much better than FUP2, which is already scalable [5], for very large databases. At the same time, the ratio $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ is not sensitive to the size of the databases. It remains around 1.0 with minor fluctuations. The critical value $\overline{u}_c$ (Figure 16) remained below 0.0030. The general tendency is that $\overline{u}_c$ decreases as the sizes of the databases increase. Therefore, the amount of uncertainty remains small. So, the accuracy of the estimations made by DELI is insensitive to the absolute sizes of the databases. These results show that our new algorithm DELI is highly scalable.

5.9 Summary

In the above experiments, we have observed that the critical values $\overline{u}_c$ and $\overline{d}_c$ are both very small. None of $\overline{u}_c$'s exceeds 0.036. This means the uncertainty level is very low. For example, if we had set the threshold $\overline{u}$ to as low as 0.05, i.e. tolerating with no more than 1 false hit in every 20 new large itemsets, the DELI

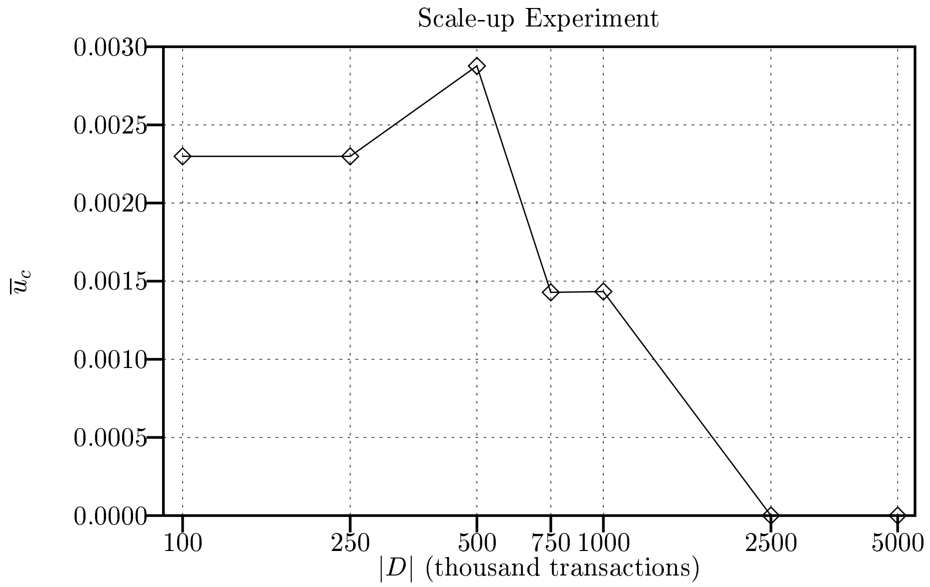

Figure 16: Scale-up Experiment (value of $\overline{u}_c$)

algorithm would still not be terminated by the first criterion. At the same time, the observed values of $\overline{d}_c$ are below 0.10 when $|\Delta^-| < 10000$. For larger values of $|\Delta^-|$, $\overline{d}_c$ is larger. In our experiments, the maximum value of $|\Delta^-|$ is 20000 and $\overline{d}_c$ never exceeds 0.21. This value is also very low. If we had set the threshold $\overline{d}$ to 0.10, i.e. requiring DELI to signal the need for a rule-update operation when no less than 10% of the set of large itemsets has changed, this threshold wouldn't be exceeded unless we have a large number of deleted transactions ($|\Delta^-| \geq 10000$). So, for these threshold values and moderate sizes of Δ^-, DELI would not suggest that FUP2 be invoked to discover the new large itemsets L'. (We have also measured the actual values of $\frac{|L \ominus L'|}{|L|}$ in the experiments, and none of them exceeded 0.18.) So, after running DELI, we wouldn't need to run FUP2 because DELI told us that the difference between L' and L were not significant. Since the speedup ratio is always above 1.0, we would have saved much time, when compared to running FUP2 directly.

The measured values of the ratio $\overline{d}_c / \frac{|L \ominus L'|}{|L|}$ in our experiments are all below 2.0. Even for a value as high as 2.0, provided that $\overline{u}_c < \overline{u}$, the need for invoking FUP2 will be signaled only when the actual value of $\frac{|L \ominus L'|}{|L|}$ is more than half of the value of $\overline{d}$. Otherwise, FUP2 is not invoked, thereby saving resources; and since $\frac{|L \ominus L'|}{|L|}$ is small, we may take L as a good approximation of L',

6 Comparing with some other algorithms that also employ sampling

Note that the Apriori algorithm described in Section 3.1 finds out all the large itemsets and their exact support counts from the database. This 100% accuracy is not required in many applications. It is possible to exchange this unnecessary accuracy for a higher performance. In [11], it is pointed out that even for fairly low values of support threshold $s\%$, a sample size of 3000 transactions gives an extremely good approximation of the large itemsets in the entire database. Working on such a small sample is much more efficient, because less I/O operations are required. In fact, the sample size is usually so small that the whole sample can reside in main memory. This reduces I/O overhead to a minimum. Moreover, examining a small sample requires much less computation than examining a huge database. Thus, sampling is a very powerful technique for association rule mining, and it should also be very useful for other data mining problems.

Several association rule mining algorithms employing sampling techniques are proposed in [15]. Algorithm 1 in that paper is a 1-pass algorithm. It scans the database only once. This algorithm is essentially the same as applying Apriori to a small sample S of the database D, with a reduced support threshold of $s'\%$ instead of $s\%$. After Apriori is completed, the large itemsets thus found are then pruned by the sampling algorithm against the original threshold $s\%$. The algorithm also checks if there are possibly missed large itemsets, using the concept of *negative border* [10]. So, the algorithm attempts to find out all the large itemsets from a small sample, and tells whether the result is accurate by reporting whether there are possible misses. Since our DELI algorithm aims at telling whether it is time to update the set of association rules, rather than finding the the rules or an approximation of it, it is difficult to compare it with Algorithm 1, which only finds an approximate answer. We would like to remark that DELI eventually aims at finding the *exact* rules by invoking another algorithm such as FUP2. The use of DELI is reduce the frequency of using FUP2 so as to save machine resources.

Algorithm 1 can only report the presence of possible misses, but it does not find out the missed itemsets. To obtain an accurate mining result, we need to find out the missed itemsets. Therefore, [15] gives Algorithm 2 for extending Algorithm 1 to find out all the missed itemsets and their support counts in a second pass. After the first pass as in Algorithm 1, those possibly missed itemsets are computed using again the concept of negative border. Then, the entire database is scanned a second time to verify if the possible misses are really large. Thus, this second algorithm is always able to find out all the large itemsets from the database. Unlike Apriori, which has to scan the database several times, the second algorithm scans the database at most twice. This avoids much I/O overhead and hence improves performance. We would like to point out that this algorithm has a very high requirement on the capacity of primary memory in the system. This is because it has to handle all its candidates in the final scan of the database. Moreover, the number of candidates is not small. We have in fact discovered that in case of the presence of possible misses, the candidate set generated by this algorithm for the final scan includes all the possible itemsets. This is because if an itemset X is not included, then either at least one of its size $|X|-1$ subsets is not in set of candidates (denoted by S), or all of its size $|X|-1$ subsets are inside the candidate set. The later case is impossible, because that would mean X is an itemset in $Br^-(S)$ (the negative border of S) and the algorithm would have included X into S. So, we're left with the former case only, which says that at least one of the size $|X|-1$ subset of X must not be in S. Continuing the application of this deduction to this subset and its own subsets and so on, we would eventually come to the conclusion that the empty-set is not in S, which would then contradict the correctness of the algorithm, since the empty-set is a (trivially) large itemset and hence it must be a candidate. So, it is impossible that an itemset X is not included in S. Thus, S has to contain all the possible itemsets, and hence its size is $2^{|I|}$. To handle such a large candidate set in one database scan, a lot of memory is needed.[9] Practically, such large amount of physical memory is not affordable. With highly inadequate memory, a virtual memory system would thrash severely. Alternatively, we may scan the databases several times, handling just a portion of the candidate set in each scan. The size of each portion handled should be such that the available memory can accommodate the support counts of all the itemsets in that portion. But then, the number of database scans is not much less than that of DELI. Therefore, we argue that our DELI algorithm is more applicable, because it has much lesser requirements on memory.

Since the memory requirement of Algorithm 2 is too large, the paper proposes Algorithm 3 to attempt to reduce it. Algorithm 3 tries to reduce the size of the candidate set S by allowing it to omit itemsets that

[9]Indeed, if we have that much memory, we can use the power set of I as our initial candidate set. Then, 1 pass of scanning the database will be sufficient to find out all the large itemsets.

are highly unlikely[10] to be large. In other words, Algorithm 3 *may* miss some large itemsets in its mining results, although the probability of this is controlled to be below a user-specified threshold. So, it does not find out the set of large itemsets *accurately*. Moreover, it focuses on controlling the miss probability but does not limit the number of the generated candidate itemsets. The candidate generation method is basically the same as Algorithm 1 extended by Algorithm 2. So, the number of generated candidates is comparable to the extended algorithm. When the probability threshold is small (so that the chances of missing large itemsets is small), Algorithm 3 behaves conservatively. It would generate a very large amount of candidates. For instance, step 7 of the algorithm examines all candidates in $S \cup Br^{-}(S)$, which are as numerous as all the candidates generated by Apriori. So, this algorithm also suffers from the problem of high memory requirement as explained above. If the probability threshold is large, then the memory requirement could be lessened, but we'll have a high chance of missing large itemsets. This is undesirable. Hence, these algorithms are not suitable for finding accurate mining results. DELI is useful for maintaining the set of large itemsets accurately.

7 Discussions

7.1 Why not sample the newly added transactions?

In the DELI algorithm, we use a sample S of the original database D to estimate the support counts σ_X of itemsets $X \in Q_k$. Why don't we do the same for the counts δ_X^+ in Δ^+ and δ_X^- in Δ^-? This is because DELI is usually applied to situations where $|\Delta^+|$ and $|\Delta^-|$ are small. It is often the case that the whole Δ^+ and Δ^- can reside in main memory or the disk-cache buffers of the operating system. For such small Δ^+ and Δ^-, we cannot gain much speed by the sampling technique, but the decreased accuracy of the estimated values of δ_X would affect the overall accuracy of DELI a lot. So, we choose to examine the whole Δ^+ and Δ^- for 100% accuracy of the counts δ_X.

7.2 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 update 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.

Note that this second run of the DELI algorithm is actually repeating much work that has already been done in the first run. So, if we can afford the storage, we can save some intermediate results from the first run of the DELI algorithm. This saved information can be reused in the subsequent runs to reduce the amount of work that has to be done. In particular, the values of δ_X^+ and δ_X^- can be saved to avoid scanning the first batch of updates again. This idea extends to the second and third runs. In general, in a sequence of consecutive runs of DELI, information can be accumulated gradually so that subsequent runs can make use of the information from the previous runs. The amount of information so accumulated will not be too large, because an update operation will be triggered eventually, at which time the accumulated information can be dumped completely.

[10] with respect to a user-specified threshold on the probability of this.

8 Conclusions

In this paper, we study the usefulness of sampling techniques in data mining. Our experience with the maintenance of association rules gives an affirmative answer. Applying sampling techniques, we devised a new algorithm DELI for finding an approximate upper bound on the size of the difference between association rules of a database before and after it is updated. The algorithm uses sampling and statistical methods to give a reliable upper bound. If the bound is low, then the amount of changes in association rules is small. So, the old association rules can be taken as a good approximation of the new ones. If the bound is high, the necessity of updating of the association rules in the database is signaled. Experiments show that DELI is not only efficient with high reliability, but also scalable. It is effective in saving machine resources in the maintenance of association rules. We have also discussed how the algorithm can be extended to handle the case of transaction deletions, and how it can be modified to make consecutive runs more efficient. These encouraging results assure that *sampling is useful in data mining*.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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*, pages 185–194, Melbourne, Australia, 1–4 April 1997.

[6] 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.

[7] 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.

[8] Jyrki Kivinen and Heikki Mannila. The power of sampling in knowledge discovery. In *13th Symposium — 1994 May: Minneapolis; MN*, volume 13 of *Proceedings of the ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems 1994*, pages 77–85, New York, NY 10036, USA, 1994. ACM Press.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.