Expert Systems with Applications 36 (2009) 10890–10895

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A decremental algorithm of frequent itemset maintenance for mining updated databases

Shichao Zhang [a,*], Jilian Zhang [b], Zhi Jin [c]

[a] Department of Computer Science, Zhejiang Normal University, PR China
[b] College of Computer Science and Information Technology, Guangxi Normal University, PR China
[c] College of Information Technology, Peking University, PR China

ARTICLE INFO

Keywords:
Incremental mining
Decremental mining
Dynamic database mining

ABSTRACT

Data-mining and machine learning must confront the problem of pattern maintenance because data update is a fundamental operation in data management. Most existing data-mining algorithms assume that the database is static, and a database update requires rediscovering all the patterns by scanning the entire old and new data. While there are many efficient mining techniques for data additions to databases, in this paper, we propose a decremental algorithm for pattern discovery when data is deleted from databases. We conduct extensive experiments for evaluating this approach, and illustrate that the proposed algorithm can well model and capture useful interactions within data when the data is decreasing.

1. Introduction

The dynamic of databases is represented in two cases: (1) the content updates over time and (2) the size changes incrementally. When some transactions of a database are deleted or modified, it says that the content of the database has been updated. This database is referred to an *updated database* and updated database mining is referred to *decremental mining*. When some new transactions are inserted or appended into a database, it says that the size of the database has been changed. This database is referred to *incremental database* and incremental database mining is referred to *incremental mining*. This paper investigates the issue of mining updated databases.[1]

When a database is updated on a regular basis, running the discovery program all over again when there is an update might produce significant computation and I/O loads. Hence, there is a need for algorithms that perform frequent patterns searches on incrementally updated databases without having to run the entire mining algorithm again (Parthasarathy, Zaki, Ogihara, & Dwarkadas, 1999). This leads to many efficient mining techniques for data additions to databases, such as the FUP algorithm (Cheung, Han, Ng, & Wong, 1996), ISM (Parthasarathy et al., 1999), negative-border based incremental updating (Thomas, Bodagala, Alsabti, & Ranka, 1997), incremental induction (Utgoff, 1994) and the weighting model (Zhang, Zhang, & Yan, 2003), and others (Zhang, Zhang, & Zhang, 2007).

Unfortunately, no work has been done for pattern discovery when data is deleted from databases. Indeed, the *delete* is one of the most frequently used operations in many DBMS systems, such as IBM DB2, MS SQL SERVER, and ORACLE. Usually, these DBMS systems use log files to record the committed changes in order to maintain the database consistency. Therefore we can easily obtain the data that is deleted from the original database by using the *delete* operation.

To solve the decrement problem, one can simply re-run an association rule mining algorithm on the remaining database. But this approach is very inefficient, wasting previous computation that has been done before. In this paper, we propose an algorithm **DUA** (*Decrement Updating Algorithm*) for pattern discovery in dynamic databases when data is deleted from databases. Experiments show that **DUA** is 2–13 times faster than re-running the Apriori algorithm on the remaining database. The accuracy of **DUA**, namely, the ratio of the frequent itemsets found by **DUA** in the remaining database to the itemsets found by re-running the Apriori, is more than 99%.

The rest of the paper is organized as follows. We provide the problem description in Section 2. In Section 3, the DUA algorithm is described in detail. Experimental results are presented in Section 4. Finally, conclusions of our study are given in Section 5.

2. Problem statement

2.1. Association rule mining

Let $I = \{i_1, i_2, \ldots, i_N\}$ be a set of N distinct literals called *items*. Let DB be a set of variable length transactions over I, where transaction

* Corresponding author.
E-mail addresses: zhangsc@it.uts.edu.au (S. Zhang), zhangjilian@yeah.net (J. Zhang).

[1] For simplifying description, this paper takes deletion as the unique update operation because a modification can be implemented by a deletion and an insert. And for the modification, we will study in another paper shortly.

0957-4174/$ - see front matter © 2009 Published by Elsevier Ltd.
doi:10.1016/j.eswa.2009.02.016

T is a set of items such that $T \subseteq I$. A transaction has an associated unique identifier called *TID*. An *itemset* X is a set of items, i.e., a subset of I. The number of items in an itemset X is the length of the itemset. X is called a *k-itemset* if the length of X is k. A transaction T contains X if and only if $X \subseteq T$. An association rule is an implication of the form $X \to Y$, where $X, Y \subset I$, $X \cap Y = \phi$. X is the *antecedent* of the rule, and Y is the *consequent* of the rule. The support of a rule $X \to Y$, denoted as $supp(X \cup Y)$, is s if s% transactions in DB contain X. and the confidence of rule $X \to Y$, denoted as $conf(X \to Y)$, is c if c% transactions in DB that contain X also contain Y. That is, $conf(X \to Y) = supp(X \cup Y)/supp(X)$.

The problem of mining association rules from database DB is to find out all the association rules whose support and confidence are greater than or equal to the minimum support (*minsupp*) and the minimum confidence (*minconf*) specified by the user respectively. Usually, an itemset X is called frequent, if $supp(X) \geqslant minsupp$. And X is infrequent if $supp(X) < minsupp$. The first step of association rule mining is to generate *frequent* itemsets using an algorithm like Apriori (Agrawal & Srikant, 1994) or the FP-Tree (Han, Pei, & Yin, 2000). And then generate association rules based on the discovered *frequent* itemsets. The second step is straightforward, so the main problem of mining association rules is to find out all frequent itemsets in DB.

2.2. Updated databases

The update operation includes deletion and modification on databases. Consider transaction database TD = {{A, B}; {A, C}; {A, B, C}; {B, C}; {A, B, D}} where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The update operation on TD can basically be

Case-1. Deleting transactions from the database TD. For example, after deleting transaction {A, C} from TD, the updated database is TD1 as

$$\mathrm{TD1} = \{\{A,B\};\{A,B,C\};\{B,C\};\{A,B,D\}\}$$

Case-2. Deleting attributes from a transaction. For example, after deleting B from transaction {A, B, C} in D, the updated database is TD2 as

$$\mathrm{TD2} = \{\{A,B\};\{A,C\};\{A,C\};\{B,C\};\{A,B,D\}\}$$

Case-3. Modifying existing attributes of a transaction in the database TD. For example, after modifying the attribute C to D for the transaction {A, C} in TD, the updated database is TD3 as

$$\mathrm{TD3} = \{\{A,B\};\{A,B\};\{A,B,C\};\{B,C\};\{A,B,D\}\}$$

Case-4. Modifying a transaction in the database TD. For example, after modifying the transaction {A, C} to {A, C, D} for TD, the updated database is TD4 as

$$\mathrm{TD4} = \{\{A,B\};\{A,C,D\};\{A,B,C\};\{B,C\};\{A,B,D\}\}$$

Mining updated databases generates a significant challenge: the maintenance of their patterns. To capture the changes of data, for each time of updating a database, we may re-mine the updated database. This is a time-consuming procedure. In particular, when a database mined is very large and the changed content of each updating transaction is relatively small, re-mining the database is not an intelligent strategy, where an updating transaction is a set of update operations. Our strategy for updated databases is decremental discovery. For simplifying description, this paper is focused on the above Case-1 and Case-2. And for the Case-3 and Case-4, we will study in another paper shortly.

2.3. Maintenance of association rules

Let DB be the original transaction database, db be a dataset randomly deleted from DB, and $DB - db$ be the remaining database. $|DB|$, $|db|$, $|DB - db|$ denote the size of DB, db, $DB - db$, respectively. Let S_0 be the minimal support specified by the user, L, L', L'' be the set of frequent itemsets in DB, db, and $DB - db$ respectively. Assume that the frequent itemsets L and the support of each itemset in L are available in advance. After a period of time, some useless data is deleted from DB, forming the deleted database db. Suppose there is an itemset X in DB, and we know that X could be frequent, infrequent or absent in db. And suppose the support of X in $DB - db$ is X_{supp}, with respect to the same minimal support S_0, and X is expected to be frequent only if $X_{supp} \geqslant S_0$. So, a frequent itemset X in DB will not be frequent again in $DB - db$ after deleting some data from DB. Similarly, an infrequent itemset X can be frequent in $DB - db$.

The tasks of maintenance for association rules are: (i) evaluating the approximate upper and lower support bounds between which the itemsets in DB are most likely to change their frequentness in the remaining database $DB - db$. (ii) finding out all the frequent itemsets in $DB - db$.

3. A decrement algorithm

We first present an analysis for the changes of frequent and infrequent itemsets in a database when some data is subtracted from the original database. And then we give a simple formula to approximate the support range in DB between which the itemsets have the chance to change their frequentness in $DB - db$.

3.1. Frequent and infrequent itemsets

It is commonly understood that the easiest way of finding the frequent itemsets in $DB - db$ is to re-run the Apriori algorithm (or the FP-Tree) on $DB - db$. When the size of db is larger than a half of the size of DB, this approach would be very effective. But this situation is not common. Usually, in many databases, in a period of time, the size of the deleted data is much less. So in this paper, we assume that the size of db is less than a half of the size of DB, i.e., $|db| < |DB|/2$.

Suppose X is an itemset in DB. S_1, S_2, and S_3 are the support of X in DB, db, and $DB - db$, respectively. With respect to the same minimal support S_0, we can obtain the relation between them by the definition of *support*.

$$S_3 = \frac{S_1 \times |DB| - S_2 \times |db|}{|DB| - |db|} \tag{1}$$

In order to discuss whether X is frequent or not in $DB - db$, namely, the frequentness of X, we introduce a function $f = S_3 - S_0$.

$$\begin{aligned} f &= \frac{S_1 \times |DB| - S_2 \times |db|}{|DB| - |db|} - S_0 \\ &= \frac{(S_1 - S_0) \times |DB| - (S_2 - S_0) \times |db|}{|DB| - |db|} \\ &= \frac{(S_1 - S_0) - (S_2 - S_0) \times \alpha}{1 - \alpha} \end{aligned} \tag{2}$$

where $\alpha = |db|/|DB|$, $0 < \alpha < 1/2$. From the above, we can see that X is frequent in $DB - db$ if and only if f is greater than or equal to 0, and X is infrequent if and only if f is less than 0. In fact, the sign of f is determined by the numerator of Formula 2. So we get another formula as a judgment of the frequentness:

$$(S_1 - S_0) - (S_2 - S_0) \times \alpha \tag{3}$$

Having got the above formula, the changes of frequentness of an itemset can be discussed in detail. We present some properties that are useful in the analysis of the frequentness of itemsets.

Lemma 1. *If an itemset X is frequent in DB, and X is infrequent in db, then X must be frequent in the remaining database DB − db.*

Proof. Based on Formula (3), when X is frequent in *DB*, i.e., $S_1 \geqslant S_0$, and X is infrequent in *db*, i.e., $S_2 < S_0$, the result of Formula 3 is greater than 0. Consequently, X is frequent in $DB - db$. □

Lemma 2. *If itemset X is frequent in DB, and X does not occur in db, then X is still frequent in the remaining database DB − db.*

Proof. Based on Formula (3), when X is frequent in DB, i.e., $S_1 \geqslant S_0$, and X does not occur in db, i.e., $S_2 = 0$, the result of Formula 3 is greater than 0. Consequently, X is frequent in $DB - db$. □

Lemma 3. *If itemset X is infrequent in DB, and X is frequent in db, then X must be infrequent in the remaining database DB − db.*

Proof. Based on Formula (3), when X is infrequent in *DB*, i.e., $S_1 < S_0$, and X is frequent in *db*, i.e., $S_2 \geqslant S_0$, the result of Formula 3 is less than 0. Consequently, X is infrequent in $DB - db$. □

Consider this situation. When $S_1 = S_2 = S_0$, i.e., the itemset X is frequent in both the database *DB* and *db*. From Formula 3, we know that X is also frequent in the remaining database $DB - db$. But consider the other situation when X is frequent in *DB* and *db*, i.e., $S_1 > S_0$ and $S_2 > S_0$, it is difficult to decide whether X is frequent or infrequent in $DB - db$ from Formula 3.

As a conclusion of Lemmas 1–3 and the discussion above, we present the changes of the frequentness of an itemset X in Table 1.

Using Properties 2, 3, and 4, whether an itemset X is frequent or infrequent in $DB - db$ could be determined in advance by the analysis of the frequentness of X in *DB* and *db* without computation using Formula 1. As for the other circumstances like 1, 5 and 6 in Table 1, the real support of X in *DB* and *db* must be obtained. Then formula 1 is used to compute the real support of X in $DB - db$.

Example 1. Consider a database *DB* with 10,000 transactions, and a *db* dataset with 1000 transactions deleted from *DB*. Suppose there are 600, 900, 490, 500 and 450 transactions in *DB* that contain itemsets *A*, *B*, *C*, *D* and *E* respectively, and there are 40, 80, 90, 0 and 0 transactions in *db* that contain *A*, *B*, *C*, *D* and *E*.

With respect to the same minimal support $S_0 = 5\%$, *A*, *B* and *D* are frequent in *DB*, *C* and *E* are infrequent. While in *db*, *A* is infrequent, *B* and *C* are frequent, and *D* and *E* do not occur. Based on the properties mentioned above, in the remaining database $DB - db$, *A* and *D* are frequent, and *C* is infrequent. The frequentness of *B* and *E* in $DB - db$ can only be determined by using Formula 1.

From Example 1, we can see that a lot of computation can be reduced when avoiding using Formula 1 to compute the frequentness of the itemsets in $DB - db$. For the infrequent itemsets in *DB* or *db*, they are not in L and L'. Their real supports cannot be available. So their frequentness in $DB - db$ is unknown. One possible method to solve this problem is to generate the infrequent itemsets as well as the frequent ones when using Apriori. This can easily conquer the problem as to which itemsets are frequent or infrequent in *DB-db*. However, this method is not feasible in practice, because the number of infrequent itemsets in a database is enormous, especially when the database is very large. In Section 3.3, we propose two approximate approaches to obtaining the real supports of some infrequent itemsets in *DB* and *db*.

Table 1
The changes of frequentness of an itemset.

	X In *DB*	X In *db*	X In *DB* − *db*
1	Frequent	Frequent	Unknown
2	Frequent	Infrequent	Frequent
3	Frequent	–	Frequent
4	Infrequent	Frequent	Infrequent
5	Infrequent	Infrequent	Unknown
6	Infrequent	–	Unknown

(– Denotes that the itemset X does not occur in *db*.)

3.2. Approximating the upper and lower support bounds for itemsets

Generally speaking, a database may have a distribution of supports of itemsets in which some itemsets have very high supports and some have very low supports. When a useless dataset *db* is subtracted from *DB*, a problem arises: Is there a support range in which the itemsets in *DB* are most likely to change their frequentness in $DB - db$? or (to put it in a different way), do there exist upper and lower support bounds larger or less than which the itemsets in *DB* do not change their frequentness in $DB - db$?

We have already proved that an itemset X is frequent in $DB - db$ if and only if Formula 3 satisfies $(S_1 - S_0) - (S_2 - S_0) \times \alpha \geqslant 0$.

As for X is frequent in both *DB* and *db*, we have

$$(S_1 - S_0) \geqslant (S_2 - S_0) \times \alpha \quad (4)$$

where $|s_1 - s_0|$ is the distance between the support of X and S_0 in *DB*, denoted as m, and $|s_2 - s_0|$ is the distance between the support of X and S_0 in *db*, denoted as n. The above formula means that itemset X is frequent in $DB - db$ if and only if m is greater than or equal to n multiplied by α. Suppose S_2 is the biggest support, then n is the maximal distance in *db*. Based on Formula 4, any frequent itemset in *DB* with a support greater than or equal to $S_0 + n \cdot \alpha$, is frequent in $DB - db$. And any frequent itemset with a support less than $S_0 + n \cdot \alpha$ is likely to be infrequent. So $S_0 + n \cdot \alpha$ is the upper support bound for itemsets in *DB*, denoted as S_{upper}.

When X is infrequent both in *DB* and *db*, we have

$$(S_1 - S_0) \leqslant (S_2 - S_0) \times \alpha \quad (5)$$

Similarly, we assume that S_2 is the smallest support in *db*. Theoretically, $S_2 = 0$. Then distance n reaches its maximum S_0. Based on Formula 5, any infrequent itemsets in *DB* with supports less than $S_0 \times (1 - \alpha)$, is infrequent in $DB - db$. And any infrequent itemsets with support greater than or equal to $S_0 \times (1 - \alpha)$ are likely to be frequent in $DB - db$. Obviously, the lower support bound for itemsets in *DB* is $S_0 \times (1 - \alpha)$, denoted as S_{lower}. The itemsets whose supports greater than S_0 and less than S_{upper} are called *unstable frequent itemsets*, denoted as *UFIS*. Similarly, itemsets whose supports less than S_0 and greater than S_{lower} are called *unstable infrequent itemsets* (*UIIS*). Fig. 1 illustrates the upper and lower support limits of itemsets.

3.3. Obtaining the supports of infrequent itemsets

In this section, we will discuss methods to be used to obtain the support of infrequent itemsets in *DB* and *db*.

First, consider the *delete* operation in DBMS. The grammar is described in SQL as follow:

delete from Database.TableName where Conditions

The deleted dataset *db* may be constructed by using many *delete* statements with different *conditions*, or mainly by using several *delete* statements that aim to delete some specified items contained in transactions of database *DB*. Generally, the size of *DB* could be very large, while the size of *db* is very small. Especially, when *db* is constructed by deleting transactions that contain several specified items from *DB*, the itemsets in *db* are only a fraction of the itemsets

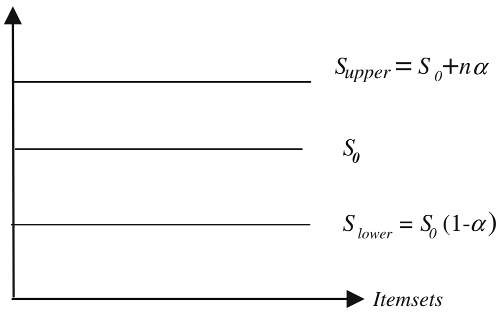

Fig. 1. The upper and lower support bounds for itemsets.

in *DB*. Consequently, a sampling technique is adopted to acquire some infrequent itemsets in *DB*. First, we randomly sample a set of transactions from *DB*, denoted as db', and then use minimal support S_0 to mine db'. By this way, we can obtain some infrequent itemsets in *DB*, which do not occur in *db*.

To acquire all the infrequent itemsets with their supports in a database is actually impossible, even in a small database, say *db*, because of the combinatorial number of them. We use a lowered minimal support threshold S' to mine *db* in order to obtain some of the infrequent itemsets. But a lower support threshold will lead to too much computation and a very large frequent itemsets. Choosing an appropriate support threshold is a tradeoff between the efficiency and accuracy when mining *db*.

Based on the previous work of the theoretical analysis for determining a lowered threshold (Toivonen, 1996). In DUA, we let S' be $0.62 \times S_0$. In our later experiments, we demonstrate that $S' = 0.62 \times S_0$ is an appropriate support threshold that satisfies the requirements both for efficiency and accuracy when mining *db*.

3.4. The DUA algorithm

Based on Lemmas 1–3 and the above discussion, the DUA algorithm is designed as follows.**Algorithm *DUA***

Input: *DB*: *the original database with size* $|DB|$; *db*: *the deleted dataset from DB with size* $|db|$; *L*: *the set of frequent itemsets in DB*; S_0: *the minimal support specified by the user*;
Output: L'': *the set of frequent itemsets in* $DB - db$;
1. **compute** S';
2. **mine** *db* with minimal support S' and **put** the frequent itemsets into L';
3. **compute** S_{upper} and S_{lower};
4. **for** each itemset in *L* **do**
5. **for** each itemset in L' **do**
6. use Table 1, formula 1, S_{upper} and S_{lower} to **identify** the frequent itemsets in $DB - db$, **eliminate** them from L' and **store** to L'';
7. randomly **sample** a set of transactions from *DB*, denoted as db';
8. **mine** db' with the threshold S_0, and obtain frequent itemsets L''';
9. **eliminate** the itemsets from L''' that occur in *L* and L';
10. **for** each remaining itemsets in L' and L'''
11. **scan** *DB* to **obtain** their support in *DB*;
12. use Table 1, formula 1, S_{upper} and S_{lower} to **identify** the frequent itemsets in $DB - db$ and **append** them to L'';
13. **end for**;
14. **return** L'';
15. **end procedure**;

From the above description, after eliminating the itemsets from L' and L''' that are infrequent or frequent in $DB - db$, it is obvious that the remaining itemsets in L' are all infrequent in *DB*, while the remaining itemsets in L''' are infrequent in DB and they do not occur in *db*.

4. Experiments

We have discussed the problem of pattern maintenance when deleting data from a large database and proposed an algorithm named **DUA** to deal with this problem. We have conducted experiments on a DELL Workstation PWS650 with 2G main memory and 2.6G CPU. The operating system is WINDOWS 2000.

4.1. Experiments for determining S' for mining db

As discussed in Section 3.3, we employ a traditional algorithm *Apriori* (Agrawal & Srikant, 1994) with a lower minimal support threshold S' to get some infrequent itemsets in *db*. In order to choose an appropriate threshold experimentally, we take into account the theoretical analysis for obtaining a lower threshold (Toivonen, 1996) and have conducted some experiments on synthetic databases.

We first generated a database T10.I4.D100K as *DB* and also a *db* of 10% transactions randomly deleted from *DB*. We set the minimal support S_0 to 1.5% and 1% respectively, and use a threshold of S' from $0.3 \times S_0$ to $0.9 \times S_0$ to mine *db*. As mentioned in Section 3.2, itemsets in *DB*, whose supports are greater than the lower bound S_{lower} and less than S_0, are most likely to become frequent in $DB - db$. We denote the infrequent itemsets in *db* as *IIS*. Table 2 shows the execution time, total itemsets in *db*, *IIS* and *UIIS* when using a different minimal support S' to mine *db*.

In another case, *db* may mainly consist of transactions, most of which contain some identical items. For example, the user deleted all the transactions that contain either item "5" or item "8" from the database. Thus, all the transactions in the database *db* either contain the specified item "5" or item "8". In Table 3, we also use another database T10.I4.D100K as *DB*, and construct *db* by deleting all transactions that contain item "120" from *DB*. The minimal support is S_0 = 1.5% and 1% respectively.

As explained before, all the *UIIS* in *DB* must be examined in order to find out whether they will become frequent in $DB - db$. So a lower threshold S' is used to find out as many *UIIS* as possible in *db*. From the above tables, we can see that the computational costs are less with few *UIIS* found when S' is set to $0.9 \times S_0$, $0.8 \times S_0$ and $0.7 \times S_0$ respectively. More *UIIS* can be found when S' is below $0.6 \times S_0$, but more computational costs are required. The appropriate tradeoff point for S' may be in the interval $[0.6 \times S_0, 0.7 \times S_0]$. We use $0.62 \times S_0$ as the minimal support threshold S' for mining *db*.

Table 2
Using different S' for mining *db* (randomly deleted).

S'	S_0 (%)	Execute itemsets	Total itemsets	IIS	UIIS
$0.9 \times S_0$	1.5	195.406	266	26	7
	1	434.265	387	36	13
$0.8 \times S_0$	1.5	283.891	294	54	13
	1	525.282	442	91	24
$0.7 \times S_0$	1.5	341.906	339	99	13
	1	617.453	482	131	28
$0.6 \times S_0$	1.5	429.875	387	147	13
	1	745.625	538	187	28
$0.5 \times S_0$	1.5	578.937	465	225	13
	1	887.015	628	277	28
$0.4 \times S_0$	1.5	746.719	538	298	13
	1	1090.937	1065	714	28
$0.3 \times S_0$	1.5	980.485	727	487	13
	1	1369.718	2658	2307	28

Table 3
Using different S' for mining *db* (specified deleted).

S'	S_0 (%)	Execute ime(s)	Total itemsets	IIS	UIIS
$0.9 \times S_0$	1.5	24.985	1631	52	0
	1	53.047	1955	106	0
$0.8 \times S_0$	1.5	32.469	1733	154	0
	1	77.453	3025	1176	0
$0.7 \times S_0$	1.5	43.937	1849	270	0
	1	91.156	1651	1802	1
$0.6 \times S_0$	1.5	53.047	1955	376	1
	1	115.25	5397	3818	2
$0.5 \times S_0$	1.5	75.188	3025	1446	2
	1	166.546	8867	7018	2
$0.4 \times S_0$	1.5	115.25	5397	3818	2
	1	379.578	18103	16254	2
$0.3 \times S_0$	1.5	380.594	18103	16254	2
	1	872.64	37301	35452	2

4.2. Experiments for algorithm DUA

We have conducted two sets of experiments to study the performance of DUA. The first set of experiments is done when *db* is generated by randomly deleting transactions from *DB*. The second set of experiments is done when *db* is generated by deleting transactions that contain specified items from *DB*.

4.2.1. Experiments for db formed by random deleting

We construct *db* by randomly deleting some transactions from *DB* (T10.I4.D100K), and then use **DUA** on *db* to study its performance against the Apriori algorithm. We define the accuracy of **DUA** as the ratio of the number of frequent itemsets found by **DUA** against the number of frequent itemsets found by Apriori in $DB - db$.

We let $|db| = 1\%|DB|$, $5\%|DB|$, $10\%|DB|$, $20\%|DB|$ and $30\%|DB|$ respectively. The following figures present the performance ratio against Apriori and the accuracy of **DUA**.

Figs. 2.1 and 2.2 reveal that **DUA** performs 6 to 8 times faster than Apriori for several databases of 100 K transactions, and the accuracy of **DUA** is 99.2% to 100%. A trend also can be seen that with the decrease of the minimal support from 1.5% to 0.5%, the accuracy is dropping. The reason is that there are many itemsets with lower supports in a database in general. When the minimal support threshold S_0 is set very low, many itemsets whose supports are slightly below S_0, namely *UIIS*, may become frequent in $DB - db$. While only a fraction of all the *UIIS* can be found in *db* and the sampling database *db′* when using **DUA**. This means that there are some frequent itemsets in $DB - db$ which cannot be found using **DUA**. So the accuracy of **DUA** drops when a lower minimal support threshold is given.

With the increase of the size of *db*, the average time ratio against Apriori slows down. In Figs. 3.1 and 3.2, when $|db| = 30\%|DB|$, **DUA** is only 1.2 times faster than Apriori. It is clear that Apriori should be applied to mining $DB - db$ instead of using **DUA**, when the amount of the data deleted from *DB* is greater than 30% of the total data in *DB*, i.e., $|db| > 30\%|DB|$.

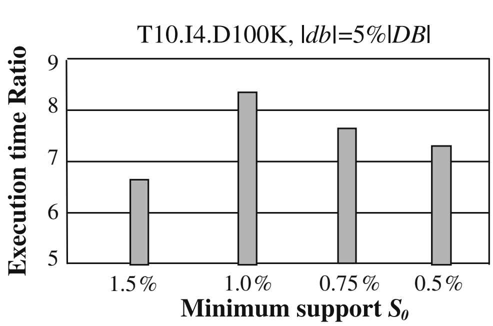

Fig. 2.1. Performance ratio versus Apriori.

4.2.2. experiments for db formed by specified deletion

In the following experiments, we construct *db* by deleting transactions that contain specified item "100" from *DB*. The above figure presents the performance and accuracy of **DUA** on *db* that is formed by deleting transactions containing the specified item

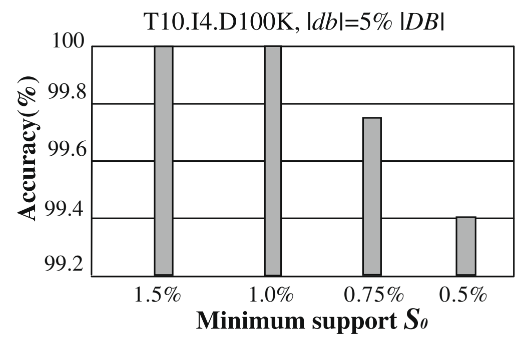

Fig. 2.2. Accuracy of **DUA**.

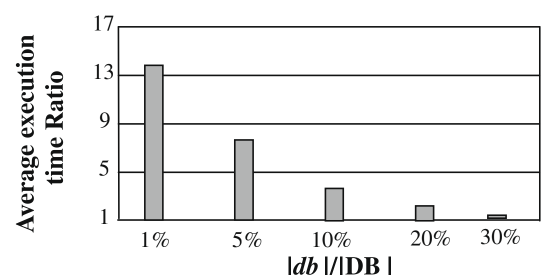

Fig. 3.1. Execution time ratio against decremental size.

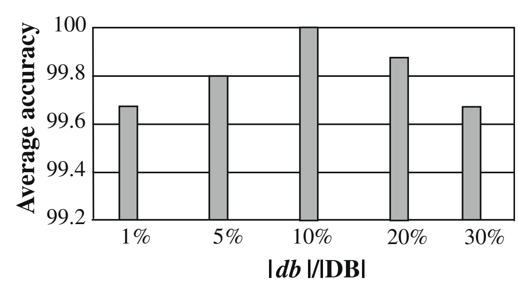

Fig. 3.2. Average accuracy with different decrement size.

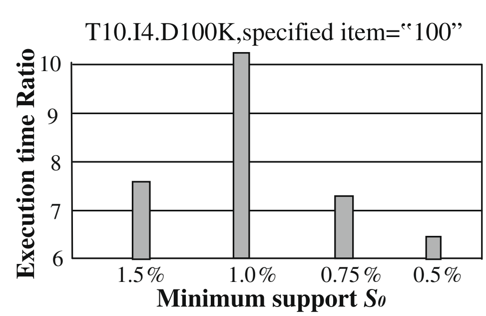

Fig. 4.1. Performance ratio against Apriori.

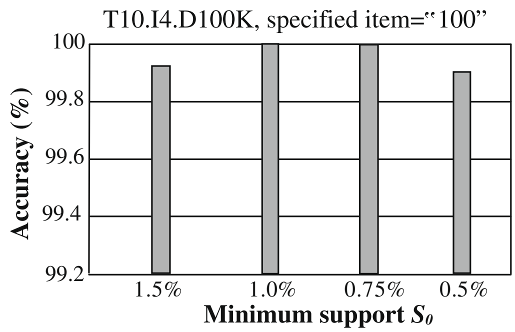

Fig. 4.2. Accuracy of DUA.

"100", which has moderate support in *DB*. In this case, DUA is still 6 to 10 times faster than re-running Apriori on $DB - db$ (Figs. 4.1 and 4.2).

5. Conclusion and future work

Pattern maintenance is a challenging task in data-mining and machine learning. In this paper, we have studied the problems of pattern maintenance when data is subtracted from a large database, and have proposed an efficient decremental algorithm to deal with the problem. Our method checks the itemsets found in the subtracted dataset and itemsets found in the original database in order to determine which itemsets are frequent in the remaining database and which are not frequent any more. Experiments have shown that our **DUA** algorithm is faster than using Apriori on the remaining database with a high accuracy.

In practice, the operations of data insertion and deletion are interleaving, which make the problem of pattern maintenance in large databases more complicated. Also, a theoretical analysis of the error bounds for **DUA** is very important. These are the main aspects of our future work.

Acknowledgements

This work was supported in part by the Australian Research Council (ARC) under grant DP0985456, the Nature Science Foundation (NSF) of China under grant 90718020, the China 973 Program under grant 2008CB317108, the Research Program of China Ministry of Personnel for Overseas-Return High-level Talents, the MOE Project of Key Research Institute of Humanities and Social Sciences at Universities (07JJD720044), and the Guangxi NSF (Key) grants.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *VLDB94*, 487–499.
Cheung, D., Han, J., Ng, V., & Wong, C. (1996). Maintenance of discovered association rules in large databases: An incremental updating technique. *ICDE'96*, 106–114.
Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD00*, 1–12.
Parthasarathy, S., Zaki, M., Ogihara, M., & Dwarkadas, S. (1999). Incremental and interactive sequence mining. *CIKM'99*, 251–258.
Thomas, S., Bodagala, S., Alsabti, K., & Ranka, S. (1997). An efficient algorithm for the incremental updation of association rules in large databases. *KDD'97*, 263–266.
Toivonen, H. (1996). Sampling large databases for association rules. *VLDB96*, 134–145.
Utgoff, P. (1994). An improved algorithm for incremental induction of decision trees. *ICML'94*, 318–325.
Zhang, S. C., Zhang, C., & Yan, X. (2003). Post-mining: Maintenance of association rules by weighting. *Information Systems, 28*(7), 691–707.
Zhang, S. C., Zhang, J. L., & Zhang, C. Q. (2007). EDUA: An efficient algorithm for dynamic database mining. *Information Sciences, 177*(13), 2756–2767.