Available online at www.sciencedirect.com

ScienceDirect

Information Sciences 177 (2007) 2756–2767

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

EDUA: An efficient algorithm for dynamic database mining ☆

Shichao Zhang [a,*], Jilian Zhang [a], Chengqi Zhang [b]

[a] *Department of Computer Science, Guangxi Normal University, Guilin, China*
[b] *Faculty of IT, University of Technology Sydney, P.O. Box 123, Broadway NSW 2007, Australia*

Received 2 September 2005; received in revised form 29 January 2007; accepted 30 January 2007

Abstract

Maintaining frequent itemsets (patterns) is one of the most important issues faced by the data mining community. While many algorithms for pattern discovery have been developed, relatively little work has been reported on mining dynamic databases, a major area of application in this field. In this paper, a new algorithm, namely the Efficient Dynamic Database Updating Algorithm (EDUA), is designed for mining dynamic databases. It works well when data deletion is carried out in any subset of a database that is partitioned according to the arrival time of the data. A pruning technique is proposed for improving the efficiency of the EDUA algorithm. Extensive experiments are conducted to evaluate the proposed approach and it is demonstrated that the EDUA is efficient.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Dynamic database mining; Pattern maintenance; EDUA; Scan reduction strategy

1. Introduction

While traditional data mining has been developed to facilitate knowledge discovery from within static databases [1,2,5,6,8,10,13,15,16,19], some algorithms have recently been developed for mining dynamic databases [4,7,9,11,12,15,17,18]. One essential difference between dynamic database mining and traditional database mining is that recently added transactions can be more "interesting" than those data items accumulated some time ago. Tracking these dynamic changes will benefit the manager of a department store in decision marking. For example, some retail items such as suits, toys, and some foods are with smart market basket data. "jeans" and "white shirts" may be purchased frequently from a department store in a period of time, while "black trousers" and "blue T-shirts" may be purchased frequently in another period of time. This means that in general, people's buying behaviors are changing with time-some goods are very frequently purchased in a period

☆ This work is partially supported by Australian Large ARC Grants (DP0449535, DP0559536 and DP0667060), a China NSF Major Research Program (60496327), a China NSF Grant (60463003), an Overseas Outstanding Talent Research Program of the Chinese Academy of Sciences (06S3011S01), and an Overseas-Returning High-level Talent Research Program of China Human-Resource Ministry.
* Corresponding author.
E-mail addresses: zhangsc@mailbox.gxnu.edu.cn (S. Zhang), zhangjilian@yeah.net (J. Zhang), chengqi@it.uts.edu.au (C. Zhang).

0020-0255/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2007.01.034

of time, while other goods are purchased frequently in some period of time. These items are called *smart goods*, or *smart items*. Apparently, most of smart items may not be frequent itemsets in the whole market basket data set. Instead, they only frequently appear in a period of time, for example, the skis that are sold in winter. Consequently, identifying them is useful for making decisions concerning the latest buying behavior, referred to as trend patterns [17]. These situations generate an urgent need for efficiently dealing with the problem of mining dynamic databases.

In real-world applications, a business database is dynamic in the sense that (1) its contents are continually being updated over time, and (2) transactions are continuously being added. For example, the content and size of the transaction database of a supermarket changes rapidly over time, with the different branches of a store chain such as Wal-Mart handling 20 million transactions a day.

These database changes over time occur by means of the data *insertion, deletion,* and *modification*[1] operations that are frequently used in database management activities. Existing algorithms are designed for mining those dynamic databases in which transactions are continually being added; this is referred to as *incremental mining*. Generally, incremental mining algorithms use previously acquired mining results to facilitate frequent itemset maintenance in the changed database instead of mining the whole database from scratch. This incremental mining strategy can lead to considerable reductions in execution time when maintaining the changed database.

In this paper, we propose a new algorithm for maintaining association rules after deleting data from a database. Our *Efficient Dynamic Database Updating Algorithm* (EDUA) is different from traditional incremental data-mining algorithms. An optimization procedure for the EDUA is also proposed as a means of improving its performance.

The rest of the paper is organized as follows. In Section 2, we briefly recall some related work and present basic concepts. The new algorithm EDUA is designed in Section 3. In Section 4, an optimization technique that can be applied to the EDUA is described. Finally, evaluation experiments are detailed in Section 5 and some conclusions are provided in Section 6.

2. Preliminaries

This section briefly outlines existing work relating to the present theme and defines the basic concepts needed.

2.1. Related work

Over the past decade, much research has been reported relating to the issue of incremental data mining [4,7,9,11,12,14,17,18]. The focus of such research is the problem of maintaining frequent itemsets when new data are added to a database. The algorithm FUP, proposed in [9], is designed to tackle the incremental mining problem while data continues to be added to a database. The idea of the FUP is to store the counts for all the frequent itemsets found in the prior mining process and then to use these stored counts to check the newly-added data. The overall counts of these itemsets are obtained by scanning the original database. The FUP iteratively discovers the new frequent itemsets in an Apriori-like manner from the changing database. The FUP$_2$ presented in [7] is an extension of the FUP that can address the problem of incremental mining in situations involving data insertion as well as data deletion. Both the FUP and the FUP^2 must scan the database k times, where k is the length of the maximal frequent itemset.

Another incremental mining algorithm was proposed in [14]. It is based on the concept of *Negative Border*, which was first introduced by Toivenon [15]. The algorithm maintains the negative borders of the frequent itemsets of the original database in the main memory. When data is added to or deleted from the original database, it requires only one scan of the whole database to ascertain whether the negative borders of the set of the original frequent itemsets have changed due to database addition or deletion operations. Although it outperforms the Apriori algorithm, the size of negative borders may be very large. This makes it impossible to keep

[1] A *modification* comprises a *deletion* followed by an *insertion*.

them in the memory, especially when the database itself is huge or has many distinct items and contains many long transactions.

The SWF is an efficient algorithm for incremental mining [11]. It adopts the technique of sliding-window filtering and mainly consists of two steps. The first one comprises a preprocessing procedure in which the database is divided into several partitions according to the arriving time of the data. The candidate 2-itemsets (C_2)) from each part are then calculated. When C_2 is obtained, the idea of *scan reduction* [6,11] can be used to generate all the candidate itemsets based on C_2. Thus only two database scans are needed to verify the whole set of candidate itemsets. Note that $C_2$is kept in the memory for the purposes of successive incremental mining procedures. The second step is straightforward in that it involves first modifying the support counts of C_2 according to the deleted and added partitions, and then repeating the process of candidate itemsets generation in a manner that is similar to the preprocessing step. Finally, the whole of the candidate itemsets are verified against the changed database, and then the identified frequent itemsets are output.

An enhanced SWF algorithm is also proposed in [4] as a further alternative for incremental data mining. It utilizes the previously discovered frequent itemsets to improve the performance of SWF in the candidate generation step and the verification step. These two algorithms can address the incremental mining problem more efficiently than other traditional incremental mining algorithms. However, they can only deal with the situation where a whole partition is eliminated while data is being deleted from the original database. This is because the two attributes *start partition* and *count* of the candidate 2-itemsets cannot work well when data is deleted from within any partition of the database. In practice, such data deletion occurred within some partitions of a database is certainly more common than deleting a whole partition from the database.

Although the SWF is efficient, it is only suitable for situations in which a block of data is deleted from a database at a time. Our EDUA proposed in this paper is designed for frequent itemset maintenance when data is deleted from any partition of a database.

2.2. Basic concepts

Let $I = \{i_1, i_2, \ldots, i_N\}$ be a set of distinct literals called items, the size of *I* is N. Let *DB* be a set of transactions over *I*, where each transaction *T* is a set of items such that $T \subseteq I$. Each transaction has a unique identifier *TID* associated with it.

An itemset *X* is a set of items. A transaction *T* is said to contain *X* if and only if $X \subseteq T$. The number of items in *X* is the length of the itemset *X*. *X* is called a *k*-itemset if the length of *X* is *k*.

An association rule is an implication of the form $X \rightarrow Y$, where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \phi$. *X* is the *antecedent* of the rule and *Y* is the *consequent* of the rule. The support of a rule $X \rightarrow Y$, denoted as $supp((X \cup Y)$, is *s* in the database *DB* if *s*% of transactions in *DB* contain $X \cup Y$. The confidence of a rule $X \rightarrow Y$, denoted as $conf(X \rightarrow Y)$, is *c* if *c*% of transactions in *DB* that contain *X* also contain *Y*. The confidence of a rule can be calculated by $conf(X \rightarrow Y) = supp(X \cup Y)/supp(X)$.

The problem of association rule mining is to identify all of the association rules that have support and confidence greater than or equal to the predefined support and confidence thresholds, namely *minsupp* and *minconf*, as determined by the user. An itemset *X* is said to be *frequent* if $supp(X) \geqslant minsupp$, and is said to be *infrequent* if $supp(X) < minsupp$. The main task of association rule mining is to discover all the *frequent* itemsets and the generation of rules from these frequent itemsets is then straightforward.

In real-world applications, a dynamic database refers to situations where the contents of the database are changed frequently over time. This involves adding new data into and deleting old (or stale) data from the database.

3. The EDUA algorithm

The algorithm EDUA is similar to the SWF in adopting the technique of *scan reduction*. The SWF divides the database into several partitions, $P^1, P^2, \ldots, P^n$, according to the arrival time of the data, and then calculates the candidate 2-itemsets from each partition by using the technique of sliding-window filtering. Each of the 2-itemsets has two attributes associated with it, namely the *start partition* and the *count*. When the first several partitions (only the first partition is deleted in the SWF, for example) are deleted, the SWF modifies the *start partition* and *count* attributes accordingly. Space limitations do not allow us to describe the SWF algorithm in detail here, and interested readers may refer to [11]. We know that in practice, situations requiring the deletion of continuous blocks of data from databases are not very common. More often, the *delete* sentence is used in most DBMS to eliminate old or obsolete data that may be found in any partition of the database. Such data needs to satisfy the conditions specified in the *delete* sentence.

Compared with the SWF, our EDUA algorithm can efficiently deal with more general cases of data deletion. A description of the EDUA algorithm is provided in Section 3.1 and then an example is presented in Section 3.2.

3.1. The algorithm description

The algorithm EDUA consists of two procedures. One is the pre-mining procedure based on the *scan reduction* technique. This procedure calculates all the frequent itemsets of the original database (DB) and maintains all the 2-itemsets (C_2) of DB for the following procedure. The second is the dynamic maintaining procedure in

```
Pre-mining procedure of the EDUA
Input:  minsupp $s$, original database $DB$
Output: all 2-itemsets $C_2$, large itemsets $L_k$
1.  $C_k=\phi$ , $L_k=\phi$ ;   //assume $C_1$ is given
2.  for each transaction $t \in DB$
3.      begin for each 2-itemset $I \in t$
4.          {  $I$.count++;
5.             $C_2= C_2 \cup I$; }
6.      end.
7.  keep the $C_2$ in memory;
8.  calculate $L_2$ from $C_2$;
9.  $C_3= L_2 \times L_2$;
10. $k$=3;
11. while ($C_k \neq \phi$) begin //scan reduction
12.     $C_{k+1}= C_k \times C_k$;
13.     $k= k+1$;
14. end.
15. for each transaction $t \in DB$ //verify $I$
16.     begin for each itemset $I \in C_k$ //$k$>2
17.         if ($I \subset t$) $I.count$++;
18.     end.
19. for each itemset $I \in C_k$
20.     if ($I.count \geq s \times |DB|$)
21.         $L_k= L_k \cup I$;
22. return $L_k$
```

```
Dynamic maintaining procedure of the EDUA
Input: minsupp $s$, $C_2$ of $DB$, deleted dataset $db$
Output: $C_2'$, $L_k'$ of $DB$-$db$
1.  $C_2^{db}=\phi$ ;   //assume $C_1^{db}$ of $db$ is given
2.  for each transaction $t \in db$
        begin for each 2-itemset $I \in t$
3.          {  $I.count$++;
4.             $C_2^{db}= C_2^{db} \cup I$;   }
5.      end.
6.  for each 2-itemset $I \in C_2^{db}$ begin
7.      if  ($I \in C_2$)
8.          $C_2'.I.count= C_2.I.count - C_2^{db}.I.count$;
9.  end.
10. keep the $C_2'$ in memory;
11. calculate $L_2'$ from $C_2'$;
12. $C_3'= L_2' \times L_2'$;
13. $k$=3;
14. while ($C_k' \neq \phi$) begin //scan reduction
15.     $C_{k+1}'=C_k' \times C_k'$;   $k=k+1$;
16. end.
17. $L_k'=\phi$ ;
18. for each transaction $t \in (DB-db)$ //verify $I$
19.     for each itemset $I \in C_k'$      // $k$>2
20.         if ($I \subset t$) $I.count$++;
21. for each itemset $I \in C_k'$
22.     if ($I.count \geq s \times |DB-db|$)
23.         $L_k'=L_k' \cup I$;
```

Fig. 1. The EDUA algorithm.

which all the 2-itemsets (C_2') in *db* (deleted from *DB*) are calculated and then the support counts of the corresponding 2-itemsets in *DB* are subtracted. The process of generating new candidate itemsets from the changed $C_2$is the same as in the pre-mining procedure. Finally, the new candidate itemsets are checked against the database *DB-db* in order to discover the frequent itemsets of the changed database. The details of these two procedures of the algorithm EDUA are set out in Fig. 1.

Note that for ease of discussion, we only use the situation of data deletion as an example in the present paper. Nevertheless, the algorithm EDUA can readily be extended to deal with the problem of data addition and deletion simultaneously. By inserting codes between lines 5 and 6 in the dynamic maintaining procedure in which the support count of 2-itemsets of the added dataset db^+ is calculated, these support counts can be added to the support counts of the corresponding 2-itemsets in C_2. While in the checking process (lines 18–20), we only need to modify line 18 to read *for each transaction* $t \in (DB - db + db^+)$. This allows the EDUA algorithm to deal with the dynamic maintenance task involving both data deletion and addition. Of course, the code of line 22 in the dynamic maintaining procedure needs to be changed to read $if(I.count \geqslant s\times \mid DB - db + db^+ \mid)$.

From Fig. 1, it can be seen that in the first step, the EDUA scans the database once in order to get the support counts of 1-itemsets and all the 2-itemsets (denoted as C_2, which we keep in the memory for the successive dynamic maintaining procedure). The frequent 2-itemsets (denoted as L_2) from C_2 are then utilized to generate candidate 3-itemsets based on the *scan reduction* [6,11], and so on. Finally, by verifying the candidate itemsets against the database, we can get all the frequent itemsets. While in the dynamic maintaining procedure (to reiterate, here we only discuss the situation of data deletion), EDUA calculates support counts for all the 2-itemsets (denoted as C_2^{db}) from the deleted dataset *db*, then modifies the support counts of the 2-itemsets in C_2that also appear in the C_2^{db}. The following processes of candidate itemsets generation and verification are similar to those of the pre-mining procedure. By the time the verification is completed, the set of new frequent itemsets of database *DB-db* is obtained.

In Section 5 below, we will demonstrate that the EDUA is more efficient than the SWF and its extension. This is mainly because that the EDUA uses L_2 to generate C_3, which will have a smaller size than that generated from C_2in the SWF. Also, the EDUA uses a technique different from that of the enhanced SWF algorithm in order to diminish the number of candidate itemsets that need to be checked against the database *DB-db*, a characteristic that further improves the performance of the EDUA.

The optimization technique to be applied to the EDUA, *verification reduction*, will be presented in Section 4 below. Note that because the EDUA only keeps the C_2in the memory, it consumes less memory space than do the other incremental algorithms [14] which keep large numbers of itemsets as well as the *negative borders* in memory. It is to be recognized, though, that the size of C_2 kept by the EDUA is larger than that kept by the SWF. This drawback can be compensated for to some extent by the smaller size of candidate itemsets generated from L_2 in the EDUA when compared to the larger size of candidate itemsets generated from C_2 in the SWF. The EDUA will be illustrated using an example in the next subsection.

3.2. An example

In this subsection, we provide an example to demonstrate how the EDUA may be applied. Note again that for the purposes of expediency in the present discussion, we only consider the situation of data deletion. An example of a transaction database is set out in Fig. 2 and the symbols used in this paper are listed in Table 1. The database *DB* is divided into two databases, one is the deleted dataset *db*, and the other is the remainder dataset *DB-db*. Let the minimal support be $s = 30\%$.

In the pre-mining procedure of the EDUA, a full scan of *DB* is required to obtain the C_2 and all the 2-itemsets C_2 ={AB(4), AC(2), AD(1), AE(4), AF(3), BC(3), BD(2), BE(5), BF(5), CE(4), CF(4), DE(2), EF(4)} (note that the numbers in the parenthesis represent the support counts of the corresponding itemset). The itemsets in $C_2$for which the support count is greater than or equal to 3 ($10 \times 30\%$=3) are stored as L_2, so we have L_2 ={AB(4), AE(4), AF(3), BC(3), BE(5), BF(5), CE(4), CF(4), EF(4)}. The $C_2$was kept in memory to facilitate the successive dynamic mining steps. Then C_3 was generated from $L_2 \; * L_2$, and the following C_{k+1} were generated from C_k $(k = 3, 4, \ldots, n)$. For expediency, we do not list the process of candidate itemsets generation. When all the candidate itemsets are generated, another scan of *DB* is executed in order to verify all the

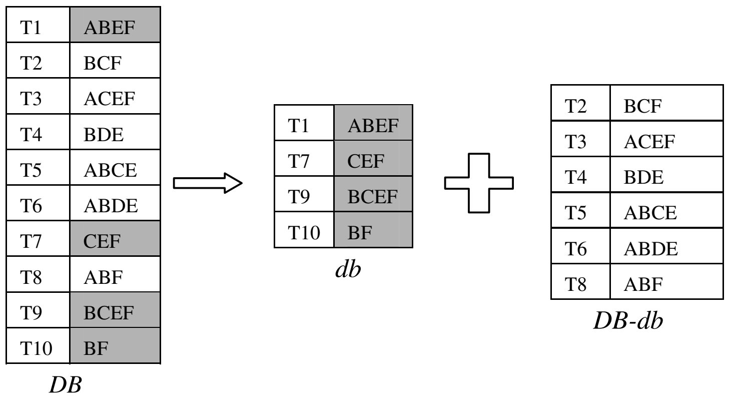

Fig. 2. Example databases.

Table 1
Some symbols used and their meanings

s	The minimal support threshold
DB	The original database
db	Database formed by data deleted from DB
$DB-db$	The database after deleting some useless data
C_k	Candidate k-itemsets of DB
L_k	Frequent k-itemsets of DB
C_k^{db}	Candidate k-itemsets of db
C'_k	Candidate k-itemsets of DB-db
L'_k	Frequent k-itemsets of DB-db

generated candidate itemsets. Thus, the EDUA can calculate all the frequent itemsets by just two scans of the database. In this example, we obtain L_3 ={ABE(3), CEF(3)} (with L_1, L_2 obtained already in the first scan of DB).

We suppose that the deleted dataset db consists of several transactions deleted from DB. In the dynamic mining procedure, the algorithm first calculates C_2^{db} from db. Here C_2^{db} ={AB(1), AE(1), AF(1), BC(1), BE(2), BF(3), CE(2), CF(2), EF(3)}. Then all the support counts of the 2-itemsets in C_2^{db} are subtracted from those of the same 2-itemsets in C_2. Thus, the modified C_2, i.e. C'_2 ={AB(3), AC(2), AD(1), AE(3), AF(2), BC(2), BD(2), BE(3), BF(2), CE(2), CF(2), DE(2), EF(1)}, contains all the 2-itemsets in DB-db. Then L'_2 ={AB(3), AC(2), AE(3), AF(2), BC(2), BD(2), BE(3), BF(2), CE(2), CF(2), DE(2)} is calculated from C'_2. Note that $C_2$is changed to C'_2, with C'_2 kept in the memory for the second dynamic maintenance task. Next, the processes of candidate itemsets generation and checking against the database are just the same as those in the pre-mining procedure. We verify the candidate itemsets C'_k ($k = 3, 4, \dots, n$) against the remainder database DB-db, and when the verification is completed, we have L'_3 ={ABE(2), ACE(2), BDE(2)} in this example.

The EDUA is an efficient means of mining dynamic databases that adopts the *scan reduction* technique while not using the mining results (frequent itemsets) of the prior mining task. In the next section, we will detail an optimization technique that aims at further improving the performance of the EDUA. It uses prior knowledge in a different manner from that in the enhanced SWF algorithm [4].

4. Optimization

As will see in Section 5, the EDUA algorithm can efficiently maintain frequent patterns in dynamic databases. Here we propose an optimization technique that can further improve the efficiency of the EDUA. The method uses heuristic knowledge of the prior mining results, which firstly uses those candidate itemsets C'_k ($k = 3, 4, \dots, n$) to scan the db. By calculating the scanning results relating to some candidate itemsets in C'_k, which are also contained in L_k, we can determine whether they are frequent in the database *DB*-*db*. They can then be removed from C'_k and need not be checked in *DB*-*db*. For the rest of the candidate itemsets in C'_k, it cannot be determined whether they are frequent or infrequent after scanning *db*, so they do need to be checked in *DB*-*db*.

4.1. Verification reduction using heuristic knowledge

In the dynamic maintaining process of the EDUA, when candidate itemsets are generated, the maintenance process involves checking all these itemsets against the database *DB*-*db*. We can see here that the maintenance process does not use the frequent itemsets of *DB*. In fact, the mined frequent itemsets can be used to reduce the number of candidate itemsets that need to be checked against *DB*-*db*.

First of all, we define a function $f(x,y)$ which denotes the support of an itemset *IS* in the database.

$$f(x,y) = \frac{x}{y} \tag{1}$$

where $0 \leqslant x \leqslant y$, $0 < y$, x denotes the number of transactions in *DB* that contain *IS*, and y denotes the total number of transactions in *DB*. It is readily seen that $\frac{\partial f(x,y)}{\partial x} = \frac{1}{y} > 0$, that is, $f(x,y)$ is an increasing function when y remains constant. Similarly, we have $\frac{\partial f(x,y)}{\partial y} = -\frac{x}{y^2} \leqslant 0$, with $f(x,y)$ as a non-increasing function when x is remains constant. These properties both hold in the situations of insertion and deletion in databases. Here, too, for reasons of limited space, we only discuss the situation of data deletion by using Eq. (1).

Let $f(x_0,y)$ be the minimal support, with $f(x,\mid DB \mid)$, $f(x_1,\mid db \mid)$ the supports of an itemset *IS* in *DB* and *db* respectively. Then the support of *IS* in *DB*-*db* is

$$f(x - x_1, |DB| - |db|) = \frac{|DB| \times f(x,|DB|) - |db| \times f(x_1,|db|)}{|DB| - |db|}$$

We let $flag = f(x - x_1, \mid DB \mid - \mid db \mid) - f(x_0,y)$, then if $flag \geqslant 0$, the itemset *IS* is frequent in *DB*-*db*; otherwise *IS* is infrequent. Then we have

$$\begin{aligned} flag &= \frac{|DB| \times f(x,|DB|) - |db| \times f(x_1,|db|) - (|DB| - |db|) \times f(x_0,y)}{|DB| - |db|} \\ &= \frac{(f(x,|DB|) - f(x_0,y)) - (f(x_1,|db|) - f(x_0,y)) \times (|db|/|DB|)}{1 - |db|/|DB|} \end{aligned} \tag{2}$$

We assume that $\mid db \mid < \mid DB \mid$ in the situation of data deletion. From Eq. (2), we can extract some useful heuristic knowledge as set out in Table 2.

Note that although we do not know the sign of *flag* for situation 1 directly, we can decide whether *IS* is frequent if we know the support of *IS* in *db*. The optimization procedure for the EDUA is based on the discussion above. Instead of scanning the database *DB*-*db*, we first scan *db* using all the candidate itemsets C'_k. Then we remove those candidate itemsets from C'_k that satisfy situations 1, 2, and 3. These need not be checked against *DB*-*db* because their support counts can be obtained after scanning *db* (situations 1 and 2) or they are simply removed due to infrequentness (situation 3). This step can eliminate many candidate itemsets from the original candidate itemsets C'_k that all need to be checked against *DB*-*db* before optimization. After scanning *db*, there are many remainder candidate itemsets for which it cannot be decided whether they are frequent or

Table 2
Heuristic knowledge about an item set

Situation	$f(x,\mid DB \mid) \geqslant f(x_0,y)$	$f(x_1,\mid db \mid) \geqslant f(x_0,y)$	Sign of flag
1	Yes	Yes	Unknown
2	Yes	No	>0
3	No	Yes	<0
4	No	No	Unknown

```
Verification reduction for dynamic procedure
1.for each transaction $t \in db$
2.   for each itemsets $I \in C_k'$ //$k$>2
3.       if ($I \subset t$) $I.count$++;
4.for each itemsets $I \in C_k'$
5.   if ($I \in L_k$)
6.       if ($L_k.I.count - C_2'.I.count \geq s \times |DB\text{-}db|$ )
7.       {   $L_k' = L_k' \cup I$;   //situation 1 and 2
8.           remove $I$ from $C_k'$ }
9.   else
10.      if ($I.count \geq s \times |db|$)     //situation 3
11.          remove $I$ from $C_k'$
12  endif
```

Fig. 3. The verification reduction procedure.

infrequent (situation 4). We then verify these remainder itemsets against *DB-db* in order to get their support counts. Generally, the size of *DB-db* is much larger than that of *db*, so that using a reduced size of candidate itemsets to scan *DB-db* can lead to the saving of time. Thus, the optimization technique discussed in this section can further improve the efficiency of the EDUA.

Consider now the dynamic maintaining procedure for the example in Section 3.2, specifically C_3' ={ABC, ABE, ABF, ACE, ACF, AEF, BCD, BCE, BCF, BDE, BDF, BEF, CEF} (note that because there are not any frequent 4-itemsets in this example, and in the interests of continued expediency, we only discuss the 3-itemsets). Instead of checking C_3' against *DB-db* as in the dynamic maintaining procedure without optimization, we first check it against *db*. After scanning *db*, C_3' ={ABC(0), ABE(1), ABF(1), ACE(0), ACF(0), AEF(1), BCD(0), BCE(1), BCF(1), BDE(0), BDF(0), BEF(2), CEF(2)}. Here {BEF(2), CEF(2)} are frequent because their support count=2 $> 4 \times 30\% = 1.2$. By considering the frequent itemsets obtained earlier L_3 ={ABE(3), CEF(3)}, we know that ABE(2) must be frequent in L_3' and that CEF(1) is infrequent in accordance with situations 2 and 1 in Table 2 respectively. So ABE(2) is stored to L_3' and removed from C_3' directly without further verification. The itemset CEF(1) is infrequent, so it is removed from C_3'. Considering the itemset BEF(2), while it is infrequent in *DB* (not in L_3), it is frequent in *db*. Consequently, it must ultimately be infrequent in *DB-db* and is removed from C_3' in accordance with situation 3. Next, the remaining elements in C_3' are verified against the database *DB-db*, which corresponds to situation 4.

The pseudo code of our optimization is presented in Fig. 3, and it can be inserted between the code of lines 17 and 18 of the dynamic maintaining procedure of the EDUA in Fig. 1.

5. Experimental studies

We conducted extensive experiments to evaluate the performance of the EDUA algorithm. The experimental platform used is a personal computer with a 2G CPU and 256MB of main memory. The operating system is Windows XP and the algorithm EDUA was coded in VC++. We generated several synthetic datasets using the same technique as in [9,13,14]. Space limitations allow us only to present the results of experiments on $|L|$= 2000 and $domain = 10K$, where $|L|$ and *domain* denote the number of frequent itemsets and the number of distinct items in the database respectively. We use d to denote the deleted data from the original database. For example, the notation T10-I4-D100K-*d*10K denotes that there are about 10,000 transactions deleted from a database of 100,000 transactions, with T as the average length of transactions and I as the average number of frequent itemsets.

5.1. Performance comparison

As indicated in the foregoing analyses, the EDUA adopts the *scan reduction* technique that is also used by the SWF and the enhanced SWF. We find from the experiments that the EDUA significantly outperforms the Apriori and the FUP algorithm, just as do the SWF and enhanced SWF (denoted as E-SWF in Fig. 4). The performance comparison is presented in Fig. 4 (in the interests of readily distinguishing the advantage of the EDUA over other algorithms, we do not show the performance of the Apriori algorithm in Fig. 4). It is apparent from these results that the EDUA is faster than the SWF. The reason for this is that the EDUA uses L_2 to generate C_k and the SWF utilizes $C_2$to generate C_k. While $C_2$is not significantly larger than L_2, the total number of candidate itemsets generated by the SWF is slightly larger than that generated by the EDUA. The main advantage of the EDUA over the SWF is that it can deal effectively with the situation where data are deleted from any part of the database, while the SWF can only function where the whole first partition of data is deleted from the database.

The performance of the EDUA with optimization is presented in Fig. 4 as EDUA-O. The EDUA-O is slightly faster than the EDUA in the databases T20I4D100K and T20I6D100K. When the minimal support threshold is low, there are many frequent itemsets discovered that need to be processed by the EDUA-O. This will diminish the speed gained while checking the reduced candidate itemsets against the database. By contrast, when the minimal support threshold is high, there are only a few candidate itemsets generated and several dozen itemsets reduced, but the EDUA-O needs to process the prior mining results while the EDUA only needs to check all the candidate itemsets against *DB*-*db*.

5.2. Comparison for candidate itemsets reduction

The EDUA, SWF, and enhanced SWF adopt the *scan reduction* technique to improve the performance of each. A sliding-window technique is also used by SWF in order to reduce the number of candidate 2-itemsets. There is, consequently, a significant reduction in the total number of candidate itemsets relative to those generated by Apriori-like algorithms [11]. The EDUA calculates all the 2-itemsets from the database, and then uses the frequent 2-itemsets L_2 from among these to generate candidate 3-itemsets. By comparison, the SWF algorithm uses $C_2$to generate candidate 3-itemsets. The total number of candidate itemsets generated by EDUA is less than that generated by SWF, as can be seen in Tables 3 and 4 (EDUA-Cand. and SWF-Cand. in these tables denote the candidate itemsets generated by EDUA and SWF, and EDUA/Freq. and SWF/Freq. denote the ratio of the number of candidate itemsets to the number of the frequent itemsets generated by EDUA and SWF respectively).

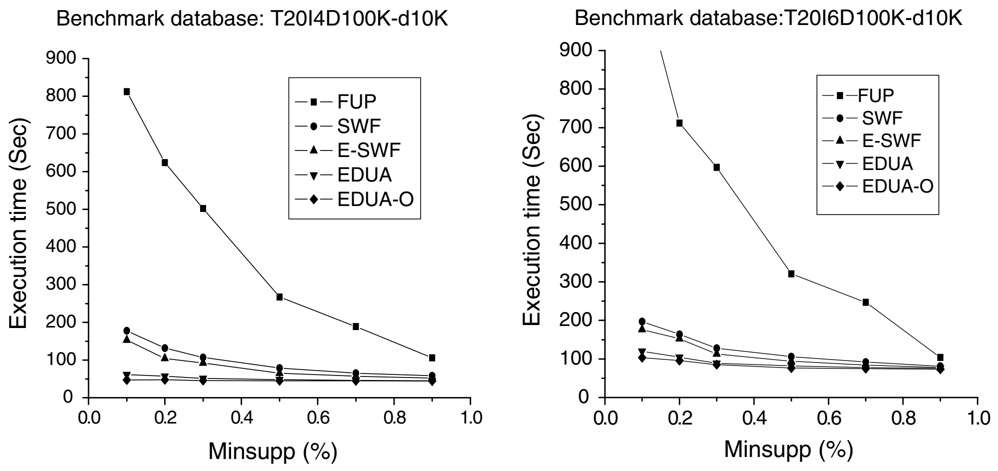

Fig. 4. Comparing the running time.

Table 3
Summary of generated candidate itemsets (T10I4D100K-d10k, *minsupp*=0.2%)

k-itemsets	Freq. itemsets	EDUA-Cand	SWF-Cand	EDUA/Freq.	SWF/Freq.
1	1830	1830	1830	1	1
2	3008	3008	3154	1	1.048
3	3255	3612	3776	1.109	1.160
4	2661	3010	3157	1.131	1.186
5	1495	1799	1926	1.203	1.288
6	529	770	883	1.455	1.669
7	121	224	241	1.851	1.991
8	17	39	46	2.294	2.705
9	1	3	4	3	4

Table 4
Summary of generated candidate itemsets (T20I6D100K-d10k, *minsupp*=0.1%)

k-itemsets	Freq. Itemsets	EDUA-Cand.	SWF-Cand.	EDUA/Freq.	SWF/Freq.
1	3909	3909	3909	1	1
2	19268	19268	19464	1	1.02
3	36317	37067	37865	1.02	1.043
4	50002	51418	52048	1.028	1.041
5	53544	55598	56642	1.038	1.058
6	46317	48573	49466	1.048	1.068
7	33219	34883	35543	1.05	1.07
8	19730	20589	21207	1.043	1.075
9	9501	9819	11082	1.033	1.166
10	3575	3656	4135	1.023	1.157
11	994	1007	1104	1.013	1.111
12	188	189	192	1.005	1.021
13	21	21	21	1	1
14	1	1	1	1	1

5.3. Memory requirements

The SWF uses the sliding-window technique as a means of significantly reducing the number of candidate 2-itemsets that are to be retained in the main memory for the later dynamic mining tasks. By contrast, the EDUA calculates all the 2-itemsets (C_2) of the database and keeps them in the memory. Although the number of C_2 itemsets kept in the memory by the EDUA is larger than that maintained by the SWF, it is still smaller than the number of candidate 2-itemsets generated by Apriori algorithms. Moreover, the size of C_2 retained by the EDUA is independent of the minimal support threshold, for it only relies on the number of distinct items, the average length of transactions, and the size of the database. Fortunately, many modern computers have large, multi-gigabyte main memories. Here, for example, only several megabytes are needed to store the counters for all the candidate 2-itemsets for the EDUA with respect to the databases shown in Table 5. The

Table 5
Memory requirements

Database	Number of candidate 2-itemsets		
	Apriori	SWF	EDUA
T10I4D100K	2689528	7482	2028837
T10I4D500K	5121054	25936	3718863
T20I4D100K	5295885	11058	4364699
T20I6D100K	7638186	26643	5392910

minimum support threshold is set at 0.1% for all the databases in this Table (note that the number of counters for 2-itemsets generated by the EDUA is independent of the minimum support threshold).

6. Conclusions and future work

In this paper, we have proposed and developed the EDUA as an efficient algorithm for mining dynamic databases. The approach aims at addressing the problem of data deletion from within any partition of the database, and this represents a key advantage relative to the SWF algorithm that can only deal with the situation where a continuous data block is deleted. The EDUA utilizes the scan reduction technique to reduce the I/O costs. Further efficiency gains stem from the verification reduction technique that is different from that adopted by the enhanced SWF algorithm. Extensive experiments have been conducted in order to demonstrate the performance of the EDUA algorithm.

The *scan reduction* technique [6] adopted by the SWF, the enhanced SWF, and our EDUA algorithms is based on the assumption that the number of candidate itemsets C_k is not much larger than L_k. In the experiments (see Tables 3 and 4), it is apparent that C_k is very close to L_k in the synthetic datasets. But we found that in real-world datasets, this assumption is not always valid. For example, we have found that for the real dataset BMS-POS (downloaded from http://fimi.cs.helsinki.fi/data/ of the ICDM workshop on FIMI 2003 and containing about 500 K transactions), the number of candidate itemsets of BMS-POS generated by using the SWF algorithm is 67886, while the number of frequent itemsets is only 1099 when minsupp is set to 1%. In this example, the number of candidate itemsets will increase to as many as 382925 when the minsupp is fixed to 0.8%, and among them only 1695 frequent itemsets are identified. This comes about mainly because the itemsets of the synthetic datasets follow a normal distribution, while the real datasets are usually skewed and follow the Zipf distribution [3]. Sometimes, the number of candidate itemsets C_k generated from C_{k-1} within real datasets is much larger than that of L_k. This can lead to great deficiency in algorithm performance. A more efficient technique is required in order to reduce the number of candidate itemsets for real datasets.

Acknowledgement

The authors would like to thank the anonymous reviewers for their constructive comments on the old versions of this paper.

References

[1] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large database, in: Proceedings of the ACM-SIGMOD Conference on management of data, May 1993, pp. 207–216.
[2] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th VLDB, Santiago, Chile, September 1994, pp. 487–499.
[3] Z. Bi, Christos Faloutsos, Flip Korn. The "DGX" distribution for mining massive, skewed data, in: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, California, USA, August, 2001, pp. 17–26.
[4] C. Chang, Shi-Hsan Yang, Enhancing SWF for incremental association mining by itemset maintenance, in: Proceedings of the 7th Pacific-Asia international conference on Knowledge Discovery and Data Mining, PAKDD, 2003, pp. 301–312.
[5] Guoqing Chen, Ming Ren, Peng Yan, Xunhua Guo, Enriching the ER model based on discovered association rules, Information Sciences 177 (7) (2007) 1558–1566.
[6] M. Chen, Jiawei Han, P.S. Yu, Data Mining: An overview from database perspective, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 866–883.
[7] D.W. Cheung, S.D. Lee, B. Kao. A general incremental technique for updating discovered association rules, in: Proceedings of the International Conference on Database Systems For Advanced Applications, DASFAA, April 1997, pp. 185–194.
[8] J. Han, Micheline Kamber, Data mining: Concepts and techniques, Higher Education Press, Beijing, 2001.
[9] D.W. Cheung, Jiawei Han, Vincent T. Ng, C.Y. Wong, Maintenance of discovered association rules in large databases: an incremental updating technique, in: Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, 1996, pp. 106–114.
[10] Yaochun Huang, et al. Mining maximal hyperclique pattern: A hybrid search strategy. Information Sciences, in press.
[11] C. Lee, Cheng-Ru Lin, Ming-Syan Chen, Sliding-window filtering: an efficient algorithm for incremental mining, in: Proceedings of international Conference on Information and Knowledge Management, CIKM01, November 2001, pp. 263–270.

[12] Ming-Yen Lin, Suh-Yin Lee, Interactive sequence discovery by incremental mining, Information Sciences 165 (3-4) (2004) 187–205.
[13] J.S. Park, M.S. Chen, P.S. Yu, An effective hash-based algorithm for mining association rules. in: Proceedings of the ACM-SIGMOD Conference on management of data, San Jose, California, May 1995, pp. 175–186.
[14] S. Thomas, Sreenath Bodagala, Khaled Alsabti, Sanjay Ranka, An efficient algorithm for the incremental updation of association rules in large databases, in: Proceedings of the 3th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, 1997, pp. 263–266.
[15] H. Toivonen, Sampling large databases for association rules, in: Proceedings of the 22th international conference on Very Large DataBase, September 1996, pp. 134–145.
[16] Jeffrey Xu Yu et al., Information Sciences 176 (14) (2006) 1986–2015.
[17] S. Zhang, Chengqi Zhang, Xiaowei Yan, Post-mining: maintenance of association rules by weighting, Information Systems 28 (7) (2003) 691–707.
[18] S. Zhang, Li Liu, Mining dynamic databases by weighting, Acta Cybernetica 16 (2003) 179–205.
[19] S. Zhang, J.L. Lu, C.Q. Zhang, A fuzzy logic based method to acquire user threshold of minimum-support for mining association rules, Information Sciences 164 (1–4) (2004).