An Efficient Algorithm for Incremental Mining of Association Rules

Chin-Chen Chang
Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan, R.O.C
Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, R.O.C
ccc@cs.ccu.edu.tw

Yu-Chiang Li Jung-San Lee
Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, R.O.C
{lyc, ljs91}@cs.ccu.edu.tw

Abstract

Incremental algorithms can manipulate the results of earlier mining to derive the final mining output in various businesses. This study proposes a new algorithm, called the New Fast UPdate algorithm (NFUP) for efficiently incrementally mining association rules from large transaction database. NFUP is a backward method that only requires scanning incremental database. Rather than rescanning the original database for some new generated frequent itemsets in the incremental database, we accumulate the occurrence counts of newly generated frequent itemsets and delete infrequent itemsets obviously. Thus, NFUP need not rescan the original database and to discover newly generated frequent itemsets. NFUP has good scalability in our simulation.

1. Introduction

Recent developments in information science have resulted in the rapid accumulation of enormous amounts of data. Consequently, efficiently managing very large databases and quickly retrieving useful information are very important. Data mining or knowledge discovery techniques are normally used to discover useful information in data warehouses. They have come to represent an important field research and have been applied extensively to several areas [7], including financial analysis, market research, industrial retail and decision support.

Mining association rules is the core task of numerous data mining techniques. As the amount of data increases, designing an efficient mining algorithm becomes increasingly urgent; accordingly, two of the main issues concerning data mining are therefore studied extensively herein. One is the design of algorithms for mining rules or patterns. The other is the design of algorithms to update and maintain rules, called incremental mining.

The most celebrated algorithm of the first type is the Apriori [3, 4] algorithm. The Apriori algorithm solves two sub-problems (1) to find all frequent itemsets, and (2) to use these frequent itemsets to generate association rules. The first sub-problem importantly governs the overall performance of the mining process. After frequent itemsets have been recognized, the corresponding association rules may be derived easily [3, 4]. Other efficient approaches have also been presented, including for example those in [1, 2, 6, 10, 11, 16, 18].

Existing mining methods cannot be applied to mine a *publication-like* database efficiently. A publication database is also a transaction database where each item involves an individual exhibition period [12]. Consider a publication database in Fig. 1, items *A*, *B* and *C* are exhibited from 1996 to 2004. However, item *F* is exhibited from 2002 to 2004. Traditional mining techniques ignore the exhibition period of each item and use a unique minimum support threshold. This measure is unfair for new products. Therefore, the PPM algorithm has been proposed to solve this problem [12].

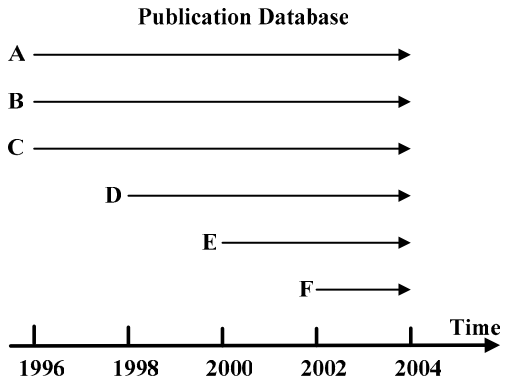

Figure 1. Each item has an individual exhibition period

In the real world where large amounts of data grow steadily, some old association rules can become useless, and new databases may give rise to some implicitly valid patterns or rules. Hence, updating rules or patterns is also important. The FUP algorithm is well known in related to

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

this issue [8]. A simple method for solving the updating problem is to reapply the mining algorithm to the entire database, but this approach is time-consuming. The FUP algorithm reuses information from old frequent itemsets to improve its performance. Several other approaches to incremental mining have been proposed [5, 9, 11, 13, 14, 15, 17, 19, 20].

Although many mining techniques for discovering frequent itemsets and associations have been presented, the process of updating frequent itemsets remains troublesome for incremental databases. The mining of incremental databases is more complicated than the mining of static transaction databases, and may lead to some severe problems, such as the combination of frequent itemsets occurrence counts in the original database with the new transaction database, or the rescanning of the original database to check whether the itemsets remain frequent while new transactions are added.

Earlier incremental algorithms have focused on reducing the number of scanning on original database while it is updated. However, they require the original database to be rescanned at least once in many situations. This work presents a novel algorithm NFUP for incremental mining, which is based on FUP. This algorithm can discover latest rules and does not need to rescan the original database. This work focuses on the generation of frequent itemsets in incremental publication-like database.

The rest of this paper is organized as follows. Section 2 introduces some work on association rules. Then, Section 3 describes the proposed method (NFUP). The experimental results of NFUP will be presented in Section 4. Conclusions are finally drawn in Section 5.

2. Related Work

In 1993, Agrawal *et al*. first defined the mining of association rules in databases [3]. They considered the example following example; 60% of transactions in which bread is purchased are also transactions in which milk is purchased. The formal statement is as follows [3]. Let $I = \{i_1, i_2, \ldots, i_m\}$ represent the set of literals, called *items*. The symbol T represents an arbitrary transaction, which is a set of items (*itemset*) such that $T \subseteq I$. Each transaction has a unique identifier, *TID*. Let *DB* be a database of transactions. Assume X is an itemset; a transaction T contains X if and only if $X \subseteq T$. An association rule applies in the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$ (For example, $I = \{ABCDE\}$, $X = \{AC\}$, $Y = \{BE\}$). An association rule $X \Rightarrow Y$ has two properties, *support* and *confidence*. When s% of transactions in *DB* contain $X \cup Y$, the support of the rule $X \Rightarrow Y$ is s%. If some of the transactions in *DB* contain X and, and c% also contain Y, then the confidence in the rule $X \Rightarrow Y$ is c%. In general, the confidence is expressed in the form confidence($X \Rightarrow Y$) = support($X \cup Y$)/support(X). Given the user-assigned minimum support (*minSup*) and minimum confidence (*minConf*) thresholds for the transaction database *DB*, the mining of association rules is to find all rules whose support and the confidence, in which are greater than the two respective minimum thresholds. An itemset is called a *frequent itemset* when its support is no less than the *minSup* threshold; otherwise, it is an *infrequent itemset*.

2.1 Apriori Algorithm

The Apriori algorithm concentrates primarily on the discovery of frequent itemsets according to a user-defined *minSup* [4]. The algorithm relies on the fact that an itemset could be frequent only when each of its subset is frequent; otherwise, the itemset is infrequent. In the first pass, the Apriori algorithm constructs and counts all 1-itemsets. (A k-itemset is an itemset that includes k items.) After it has found all frequent 1-itemsets, the algorithm joins the frequent 1-itemsets with each other to form candidate 2-itemsets. Apriori scans the transaction database and counts the candidate 2-itemsets to determine which of the 2-itemsets are frequent. The other passes are made accordingly. Frequent (k - 1)-itemsets are joined to form k-itemsets whose first k-1 items are identical. If $k \geq 3$, Apriori prunes some of the k-itemsets; of these, (k – 1)-itemsets have at least one infrequent subset. All remaining k-itemsets constitute candidate k-itemsets. The process is reiterated until no more candidates can be generated.

Example 2.1 Consider the database presented in Table 1 with a minimum support requirement is 50%. The database includes 11 transactions. Accordingly, the supports of the frequent itemsets are at least six. The first column "TID" includes the unique identifier of each transaction, and the "Items" column lists the set of items of each transaction. Let C_k be the set of candidate k-itemsets and F_k be the set of frequent k-itemsets. In the first pass, the database is scanned to count C_1. If the support count of a candidate exceeds or equals six, then the candidate is added to F_1. The outcome is shown in Figure 2. Then, $F_1 \infty F_1$ forms C_2 (Apriori-gen function is used to generate C_2 [4]). After the database has been scanned for a second time, Apriori examines which itemset of C_2 exceeds the predetermined threshold. Moreover, C_3 is generated from F_2 as follows. Figure 2 presents two frequent 2-itemsets with identical first item, such as $\{BC\}$ and $\{BE\}$. Then, Apriori tests whether the 2-itemset $\{CE\}$ is frequent. Since $\{CE\}$ is a frequent itemset, all the subsets of $\{BCE\}$ are frequent. Thus, $\{BCE\}$ is a candidate 3-itemset, or $\{BCE\}$ must be pruned. Apriori stops to look for frequent itemsets when no candidate 4-itemset can be joined from F_3. Apriori scans the database k times when candidate k-itemsets are generated.

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

Table 1. An example of a transaction database

TID	Items
001	A C D E
002	A C D
003	B C E
004	A B C E
005	A B E
006	B C E
007	A B E
008	B C D E
009	A B C D
010	C E F
011	A B C F

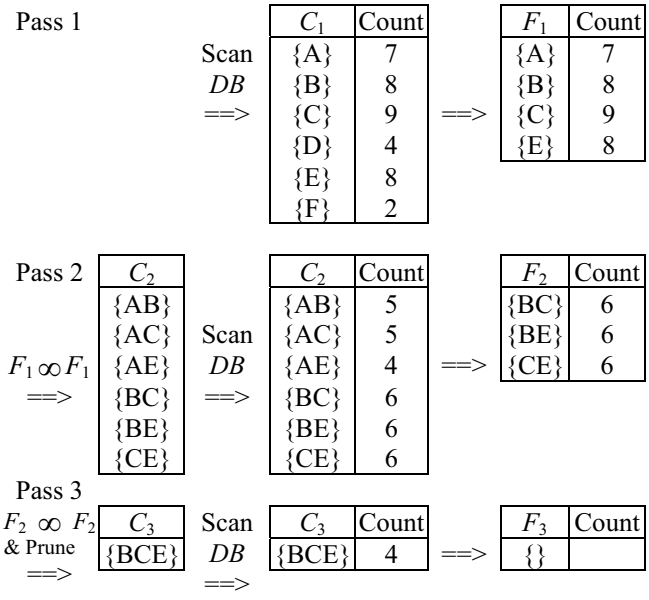

Figure 2. Application of Apriori algorithm

2.2 Fast Update Algorithm (FUP)

After a database has been updated, some existing rules are no longer important and new rules may be introduced. In 1996, Cheung *et al*. proposed the FUP algorithm to efficiently generate associations in the updated database [8]. The FUP algorithm relies on Apriori and considers only these newly added transactions. Let *db* be a set of new transactions and DB^+ be the updated database (including all transactions of *DB* and *db*). An itemset *X* is either frequent or infrequent in *DB* or *db*. Therefore, *X* has four possibilities, as shown in Table 2. In the first pass, FUP scans *db* to obtain the occurrence count of each 1-itemset. Since the occurrence counts of F_k in *DB* are known in advance, the total occurrence count of arbitrary *X* is easily calculated if *X* is in Case 2. If *X* is unfortunately in Case 3, *DB* must be rescanned. Similarly, the next pass scans *db* to count the candidate 2-itemsets of *db*. If necessary, *DB* is rescanned. The process is reiterated until all frequent itemsets have been found. In the worst case, FUP does not reduce the number of the original database must be scanned.

Table 2. Four scenarios associated with an itemset in DB^+

DB \ *db*	Frequent itemset	Infrequent itemset
Frequent itemset	Case 1: Frequent	Case 2:
Infrequent itemset	Case 3:	Case 4: Infrequent

2.3 Other Incremental Algorithms

In 1997, Cheung *et al*. described the FUP_2 algorithm, which is a more general incremental technique than FUP. FUP_2 is efficient not only on developing of a database but also on trimming data [9]. Tomas *et al*. proposed another method to accelerate the incremental mining by maintaining the *negative border* [19]. In 1999, Ayan *et al*. proposed the UWEP (Update With Early Pruning) method that employs a dynamic look-ahead strategy to update existing frequent itemsets for detecting and removing itemsets that are infrequent in the updated database [5]. In 2001, Lee *et al*. proposed the SWF approach [13]. SWF partitions databases into several partitions, and applies a filtering threshold in each partition to generate candidate itemsets. In 2002, Veloso *et al*. described the ZigZag algorithm which uses *tidlist* and computes maximal frequent itemsets in the updated database to avoid the generation of many unnecessary candidates [20].

3. New Fast Update Method (NFUP)

The key idea behind of previous incremental mining techniques is to reduce the number of times that databases need to be scanned. Although those techniques may avoid some unnecessary scanning, they do rescan the original database. The original database is normally much larger than the incremental database. Therefore, scanning the original database is time-consuming. This study proposes a new fast update algorithm (NFUP) for incremental mining of association rules. NFUP does not require the rescanning of the original database to detect new frequent itemsets or delete invalidate itemsets.

3.1 NFUP Algorithm

FUP rescans the original database when at least one candidate is in Case 3. In many situations, new information is more important than old information, such as in publication database, stock transactions, grocery markets,

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

or web-log records. Consequently, a frequent itemset in the incremental database is also important even if it is infrequent in the updated database.

To mine new interesting rules in updated database, NFUP partitions the incremental database logically according to unit time interval (month, quarter or year, for example). For each item, assume that the ending time of exhibition period is identical. NFUP progressively accumulates the occurrence count of each candidate according to the partitioning characteristics. The latest information is at the last partition of incremental database. Therefore, NFUP scans each partition backward, namely, the last partition is scanned first and the first partition is scanned last. As in the preceding section, the original transaction database is denoted as *DB*, where *db* indicates the incremental portion, and DB^+ signifies the updated database. The frequent set of itemsets of *DB* is known in advance. The new transaction database *db* includes *n* unit time intervals. Logically, *db* can be divided into *n* portions and each portion is called a partition ($db = P_1 \cup P_2 \cup, ..., \cup P_n$ where P_n denotes the partition *n*). Let $db^{m, n}$ represent the continuous time interval from partition P_m to partition P_n, where $n \geq m \geq 1$ and $n \in N$. Namely, $db^{m, n} = P_m \cup P_{m+1} \cup, ..., \cup P_{n-1} \cup P_n$. The NFUP algorithm is an Apriori-like algorithm. The final set of frequent itemsets consists of the three following types.

(1) α set: frequent itemsets in DB^+,
(2) β set: frequent itemsets in $db^{m, n}$ ($m \leq n$), but infrequent in $db^{m-1, n}$, and
(3) *r* set: frequent itemsets in $db^{m, m}$, but infrequent in $db^{m+1, n}$.

For $db^{n, n}$ (P_n), the process starts at 1-itemsets. Each candidate or frequent itemset has three attributes.

(1) *X*.count: includes the occurrence count in current partition,
(2) *X*.start: includes the partition number of the corresponding starting partition when *X* becomes an element of frequent set, and
(3) *X*.type: denotes one of the three types α, β, and *r*. In the beginning, set α, set β, and set *r* are empty.

After P_n has been scanned, all frequent 1-itemsets are added into the α set. Each frequent 1-itemset is joined to form 2-itemset candidates. In P_n, the process is performed like that of Apriori. NFUP is applied to the next partition P_{n-1} whenever no more candidate *k*-itemsets can be generated in P_n. The occurrence count of each candidate in P_{n-1} is known after P_{n-1} is scanned. In each partition, NFUP determines which candidate *k*-itemset will become an element of α, β or *r* set and identifies from which partition the *k*-itemset becomes frequent. After P_1 is scanned, the occurrence count is accumulated with that of *DB*. Figure 3 graphically depicts NFUP process for P_m, where F^α is the α set, F^β is the β set, and F^r is the *r* set.

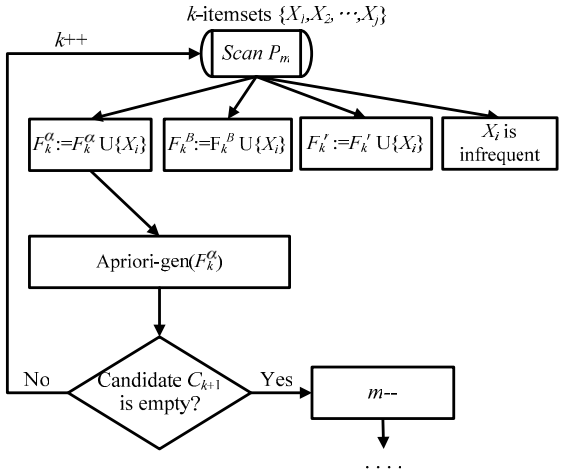

Figure 3. Process of NFUP for P*m*

The pseudo-code of NFUP algorithm is presented as follows.

Input: (1) DB^+: updated database that contain *DB* (original database) with size |*DB*| and the incremental database $db^{1,n}$ with size $|db^{1,n}| = |\sum_{m=1}^{n} P_m|$, (2) F_k: set of all frequent *k*-itemsets in *DB*, where $k = 1, 2, \ldots, h$, and (3) *s*: *minSup*.

Output: (1) F^α (α set): frequent itemsets in DB^+, (2) F^β (β set): frequent itemsets in $db^{m, n}$ ($m \leq n$), but infrequent in $db^{m-1, n}$ or infrequent in DB^+ (if $m = 1$), and (3) F^r (*r* set): frequent itemsets in $db^{m, m}$, but infrequent in $db^{m+1, m+1}$.

Procedure:

// $C^m{}_k$ is the candidate *k*-itemsets in partition *m*
// $F^\alpha{}_k$ is the set with *k* items in F^α; $F^\beta{}_k$ is the set with *k* items in F^β; $F^r{}_k$ is the set with *k* items in F^r;
// F^{DB} is the set of frequent itemsets in *DB*

```
01 F^α := φ ; F^β := φ ; F^r := φ ;
02 for m := n to 1 {
03    k:=1;
04    SubProcedure();
05    for k := 2 to h {
06          C^m_k := Apriori-gen(F^α_{k-1});
07          SubProcedure();
08    }
09 }
10 forall X ∈ F^α ∪ F^DB {
11    if X ∈ F^α && X ∈ F^DB {
12          F(X).start = 0 ;
13          F(X).count += F^DB(X).count; }
14    elseif X ∈ F^α && X ∉ F^DB      {
15          F(X).type = β;     }
16    elseif X ∈ F^DB && X ∉ F^α ∪ F^β ∪ F^r {
17          F(X).type = r;     }
18 }
19 return F^α, F^β, and F^r;
```


Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

In Line 1, α, β and r sets are initialized to empty. In Lines 4 and 7, SubProcedure() is applied to scan a portion of *db* and to compute the support number of each candidate. At the same time, it determines which candidate should be added to a proper frequent set or be ignored. The Apriori-gen function takes frequent (k-1)-itemsets to generate k-itemset candidates. The prune step is included in the Apriori-gen function. Some of the k-itemsets are deleted; of these, at least one (k-1) sub-itemset is not in F^{α}_{k-1}. From Line 10 to Line 18, the final support value of each frequent itemset of α set needs to be added its occurrence count in *DB*. Otherwise, they are the elements of β set. The pseudo-code of SubProcedure() is presented as follows.

```
SubProcedure():        // the k-th pass in each partition
01  if C^m_k ⊆ φ { break; }
02  if m == n {
03      forall X∈ C^m_k { F(X).count = 0; } }
04  forall X∈ C^m_k { X.count :=0; }
05  forall T∈ P_m {  // scan P_m, T:transaction
06      forall X∈ (subset of T) {
07          if X∈ C^m_k { X.count++; }        }
08  }
09  forall X∈ C^m_k {
10      if X∈ F^α_k {
11          if X.count ≥ s*|P^m| {
12              F(X).start = m ; F(X).count += X.count; }
13          else {
14              if F.count + F(X).count ≥ s*|db^{m,n}| {
15                  F(X).count += X.count; }
16              else {
17                  F^α_k := F^α_k - {X}; F^β_k := F^β_k ∪ {X};
18                  F(X).count += X.count;
19                  F(X).type := β; } }        }
20      elseif X∈ C^m_k − F^α_k − F^β_k − F^r_k{
21          if X.count ≥ s*|P^m| {
22              F(X).count := X.count;
23              if m == n { F(X).type := α ;}
24              else { F^r_k := F^r_k ∪ {X}; F(X).type := r ;}
25          }   }
26  }
```

Example 3.1 Consider the transaction database presented in Table 3 with a minimum support requirement is 50%. The original database contains five transactions and that six incremental transactions are divided into two portions $\{P_1, P_2\}$. *DB* corresponds to the time granularity from 1999 to 2001. P_1 and P_2 correspond to the time granularities 2002 and 2003, respectively. All frequent itemsets of *DB* are known in advance, and presented in Table 4. Initially, the three types of frequent set are null. Without loss of generality, suppose *DB* is the partition zero (P_0). All the frequent k-itemsets in P_m are also list in Table 4. P_2 is first to be scanned. In the beginning, these frequent k-itemsets in P_2 belong to the α set. Table 5 lists the three types of frequent set. The column "Start" states the identity of the start partition. After P_1 is scanned, the α set consists of five entries. In α set, the value of start partition of each frequent k-itemset is modified to be one. Four k-itemsets are removed to the β set because their total occurrence counts are less than the threshold (6 * 50% = 3). Furthermore, the three itemsets $\{E\}$, $\{BE\}$, and $\{CE\}$ are presented in r set. Although $\{BE\}$ is frequent in $db^{1,2}$, it is infrequent in P_2, as marked by "*". Finally, NFUP adds the occurrence counts of frequent itemsets in *DB* to the corresponding frequent itemsets of the α set. In Table 5, the final α set remains three entries $\{A\}$, $\{B\}$, and $\{C\}$. The β set and r set increase two and one entries, respectively.

Table 3. Transaction database

Date	Partition	TID	Transaction
1999 | 2001	*DB*	001	A C D E
		002	A C D
		003	B C E
		004	A B C E
		005	A B E
2002	P_1	006	B C E
		007	A B E
		008	B C D E
2003	P_2	009	A B C D
		010	C E F
		011	A B C F

Table 4. Frequent itemsets in each partition

P_2		P_1		*DB*	
Itemset	Count	Itemset	Count	Itemset	Count
{A}	2	{B}	3	{A}	4
{B}	2	{C}	2	{B}	3
{C}	3	{E}	3	{C}	4
{F}	2	{BC}	2	{E}	4
{AB}	2	{BE}	3	{AC}	3
{AC}	2	{CE}	2	{AE}	3
{BC}	2	{BCE}	2	{BE}	3
{CF}	2			{CE}	3
{ABC}	2				

As shown in Table 5 from 1999 to 2003, the three publications or products ($\{A\}$, $\{B\}$ and $\{C\}$) are very popular because their start partitions are zero. $\{AB\}$ and $\{BC\}$ are the two elements of the β set and P_1 is their start partition. Thus, from 2002 to 2003, the two combinations of products are interesting. $\{AE\}$ is in the r set and the start partition is zero. Hence, $\{AE\}$ is very popular from 1999 to 2001. However, $\{AE\}$ is no longer interesting from 2002 to 2003. The itemset $\{E\}$ is frequent in traditional incremental rules mining such as FUP, while $\{E\}$ is in the r set of the mining result of NFUP. Although the occurrence

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

count of {*E*} is eight in DB^+, {*E*} is no longer frequent in the year 2003.

Table 5. Frequent itemsets generated incrementally by NFUP

After P_2 is scanned		
α set	Start	Count
{A}	2	2
{B}	2	2
{C}	2	3
{F}	2	2
{AB}	2	2
{AC}	2	2
{BC}	2	2
{CF}	2	2
{ABC}	2	2

After P_1 is scanned		
α set	Start	Count
{A}	1	3
{B}	1	5
{C}	1	5
{AB}	1	3
{BC}	1	4
β set	Start	Count
{F}	2	2
{AC}	2	2
{CF}	2	2
{ABC}	2	2
r set	Start	Count
{E}	1	3
*{BE}	1	3
{CE}	1	2

Final frequent sets		
α set	Start	Count
{A}	0	7
{B}	0	8
{C}	0	9
β set	Start	Count
{F}	2	2
{AB}	1	3
{AC}	2	2
{BC}	1	4
{CF}	2	2
{ABC}	2	2
r set	Start	Count
{E}	1	3
{AE}	0	3
{BE}	1	3
{CE}	1	2

NFUP needs not rescan the original database since the α set is frequent in the updated database DB^+, and the β set provides important rules in $db^{m,n}$. Determining all frequent itemsets in DB^+ as Apriori or FUP, it requires the rescanning of the original database only once to check the β set and *r* set. However, this rescanning phase is unnecessary because all frequent itemsets in DB^+ are the subset of $\alpha \cup \beta \cup r$. Furthermore, all the itemsets contained in β set or *r* set are interesting. To determine all frequent itemsets in DB^+ as Apriori or FUP, the important rule could be deleted by pruning some frequent itemsets from the β set or *r* set.

The set ($\alpha \cup \beta \cup r$) is the superset of the mining result of Apriori in DB^+. The reason is that if *DB* is divided into *n* partitions, then the frequent itemset *X* must appear as a frequent itemset in least one of the *n* partitions [18].

4. Experimental Results

All the experiments were performed on a 1.5GHz Pentium IV PC with 640 MB of main memory, running under Windows 2000 professional. The algorithm was coded in Visual C++ 6.0. The synthetic datasets employed in our experiments are generated by using the same technique introduced in [4]. Table 6 is the parameters of the synthetic data generation program.

We generate the transaction database DB^+ with size $|DB^+|$, where the first $|DB|$ is the original database and the next $|db|$ ($|DB^+|$ - $|DB|$) is the incremental portion. The result is shown in Figure 4, where $|D|$ = 100,000, $|d|$ = 100,000, P = 1 to 4, $|T|$ = 10, $|I|$ = 4, $|L|$=2000, and N = 1000. The notation T*x*.I*y*.D*z*.d*m*.P*n* denotes an updated database DB^+, where $|T| = x$, $|I| = y$, $|D| = z$, $|d| = m$, and $P = n$. Figure 4 shows the running time for P = 1, 2 and 4. The *minSup* threshold is decreased from 1.2% to 0.2%. NFUP has the best performance when the partition number of *db* is one.

Table 6. Parameters

$\|D\|$	Number of transactions in *DB*
$\|d\|$	Number of transactions in *db*
P	Partition number of the incremental *db*
$\|T\|$	Mean size of the transactions
$\|I\|$	Mean size of the maximal potentially frequent itemsets
$\|L\|$	Number of maximal potentially frequent itemsets
N	Number of items

To test the scalability with the number of transactions of *db*, the number of partitions of *db* is set to 2. Figure 5 presents the results. Consider the two minimum support thresholds 0.2% and 0.4%, the running time slightly increases with the growth of *db*'s size. Thus, NFUP shows good scalability.

Figure 6 shows the scalability with the number of transactions of *DB*, where the incremental database is also divided into two partitions. The curves of 0.2% and 0.4% minimum support thresholds are very flat. Therefore, the running time of NFUP is irrelevant to the number of transactions of *DB*.

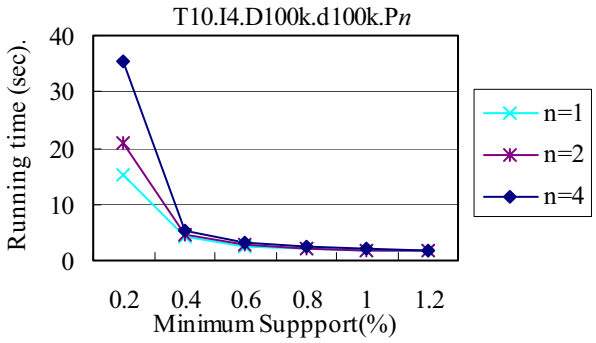

Figure 4. Running time of NFUP for n = 1, 2 and 4

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

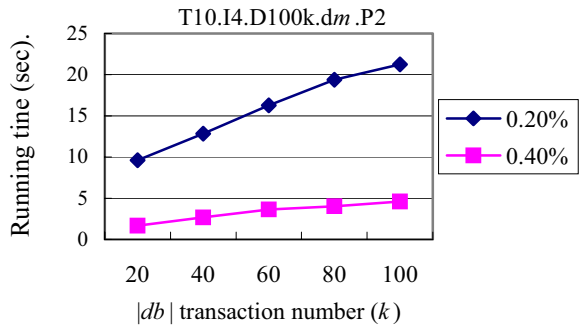

Figure 5. Scalability with the transaction number of db

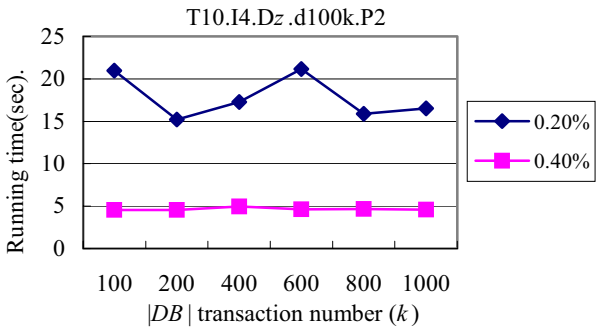

Figure 6. Scalability with the transaction number of DB

5. Conclusions

In the real world, databases are periodically and continually updated. Therefore, mining must be repeated. Valid patterns and rules must to be efficiently generated. Incremental mining must usually involve the original database and the new added transactions. Scanning the original database is very expensive, so the proposed method outperforms others by avoiding the rescanning of the original database.

This investigation has presented a new method, NFUP, for incremental mining. NFUP does not require the rescanning of the original database and can determine new frequent itemsets at the latest time intervals. The proposed method uses information available from a following partition to avoid the rescanning of the original database; it requires only the incremental database to be scanned. In reality, the transaction number of the incremental database is very small in contrast to the original database. The running time of NFUP rises almost in direct proportion with the transaction number of the incremental database. Accordingly, NFUP is suited frequently updated databases. In the future, the authors will consider the extension of the NFUP algorithm to sequence rules.

References

[1] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, "A tree projection algorithm for generation of frequent itemsets," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 3, pp. 350-361, 2000.

[2] C. C. Aggarwal and P. S. Yu, "Mining associations with the collective strength approach," *IEEE Trans. Knowledge and Data Engineering*, Vol. 13, No. 6, pp. 863-873, 2001.

[3] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases" In *Proc.* 1993 *ACM SIGMOD Intl. Conf. on Management of Data*, Washington, D.C., pp. 207-216, May 1993.

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," In *Proc. 20th Intl. Conf. on Very Large Data Bases*, Santiago, Chile, pp. 487-499, Sep. 1994.

[5] N. F. Ayan, A. U. Tansel, and E. Arkun, "An efficient algorithm to update large itemsets with early pruning," In *Proc. 5th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, San Diego, CA, pp. 287-291, Aug. 1999.

[6] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," In *Proc. 1997 ACM SIGMOD Intl. Conf. on Management of Data*, Tucson, AZ, pp. 255-264, May 1997.

[7] M. S. Chen, J. Han, and P. S. Yu, "Data mining: An overview from a database perspective," *IEEE Trans. Knowledge Data Engineering*, Vol. 8 No. 6, pp. 866-883, 1996.

[8] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, "Maintenance of discovered association rules in large databases: an incremental updating technique," In *Proc. 12th Intl. Conf. on Data Engineering*, New Orleans, LA, pp. 106-114, Feb. 1996.

[9] D. W. Cheung, S. D. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," In *Proc. 5th Intl. Conf. on Database Systems for Advanced Applications*, Melbourne, Australia, pp. 185-194, Apr. 1997.

[10] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," In *Proc.* 2000 *ACM-SIGMOD Intl. Conf. on Management of Data*, Dallas, TX, pp. 1-12, May 2000.

[11] T. P. Hong, C. Y. Wang, and Y. H. Tao, "A new incremental data mining algorithm using pre-large itemsets," *Intelligent Data Analysis*, Vol. 5, No. 2, pp. 111-129, 2001.

[12] C. H. Lee, M. S. Chen, and C. R. Lin, "Progressive partition miner: An efficient algorithm for mining general temporal association rules," *IEEE Trans. Knowledge Data Engineering*, Vol. 15, No. 4, pp. 1004-1017, 2003.

[13] C. H. Lee, C. R. Lin, and M. S. Chen, "Sliding-window filtering: An efficient algorithm for incremental mining," In *Proc. 10th Intl Conf. on Information and Knowledge Management*, Atlanta, GA, pp. 263-270, Nov. 2001.

[14] K. K. Ng and W. Lam, "Updating of association rules dynamically," *In Proc. Intl. Symposium on Database Applications in Non-Traditional Environments*, Kyoto, Japan, pp. 84-91, Nov. 1999.

[15] T. Y. Ng, M. L. Wong, and P. Bao, "Incremental mining of association patterns on compressed data," In *Proc. Joint 9th IFSA World Congress and 20th NAFIPS Intl. Conf.,* Vancouver, Canada, pp. 441-446, Jul. 2001.

[16] J. S. Park, M. S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," In *Proc.*

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**

1995 ACM-SIGMOD Intl. Conf. on Management of Data, San Jose, CA, pp. 175-186, May 1995.

[17] N. L. Sarda and N. V. Srinivas, "An adaptive algorithm for incremental mining of association rules," In *Proc. 9th Intl. Workshop on Database and Expert Systems Applications*, Vienna, Austria, pp. 240-245, Aug. 1998.

[18] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases," In *Proc. 21st Conf. on Very Large Data Bases*, Zurich, Switzerland, pp. 432-444, Sep. 1995.

[19] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, "An efficient algorithm for the incremental updating of association rules in large database," In *Proc. 3rd Intl. Conf. on Data Mining and Knowledge Discovery*, Newport Beach, CA, pp. 263-266, Aug. 1997.

[20] A. A. Veloso, W. Meira Jr., M. B. de Carvalho, B. Pôssas, S. Parthasarathy, and M. J. Zaki, "Mining frequent itemsets in evolving databases," In *Proc. 2nd SIAM Intl. Conf. on Data Mining*, Arlington, VA, Apr. 2002.

Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)
1097-8585/05 $20.00 © 2005 **IEEE**