NEW FAST ALGORITHM FOR INCREMENTAL MINING OF ASSOCIATION RULES

Yasser El Sonbati, Rasha Kashef
Arab Academy for science & Technology, College of Comuputing and Information Technology
P.0.Box. 1029. Abu Kir , AlexandriaEgypt

Keywords: Data mining, Association rules, Incremental mining.

Abstract: Mining association rules is a well-studied problem, and several algorithms were presented for finding large itemsets. In this paper we present a new algorithm for incremental discovery of large itemsets in an increasing set of transactions. The proposed algorithm is based on partitioning the database and keeping a summary of local large itemsets for each partition based on the concept of negative border technique. A global summary for the whole database is also created to facilitate the fast updating of overall large itemsets. When adding a new set of transactions to the database, the algorithm uses these summaries instead of scanning the whole database, thus reducing the number of database scans. The results of applying the new algorithm showed that the new technique is quite efficient, and in many respects superior to other incremental algorithms like *Fast Update Algorithm* (*FUP)* and *Update Large Itemsets* (*ULI)*.

1 INTRODUCTION

Data mining is the process of discovering potentially valuable patterns, associations, trends, sequences and dependencies in data [Agrawal et al., 1993][Agrawal et al., 1994][Agrawal and Yu 1998]. Mining association rules is one of the vital data mining problems. An association rule is a relation between items in a set of transactions. This rule must have a statistical significance (support) with respect to the whole database and its structure must have a semantic prospective (confidence), as will be stated in more details later in section 2. *Apriori* algorithm [Agrawal et al., 1994] is the first successful algorithm for mining association rules. It introduces a method to generate candidate itemsets C_k in pass k using only large itemsets L_{k-1} in the previous pass. *Direct Hashing and Pruning (DHP) algorithm* [Park et al., 1997] is the next algorithm for efficient mining of association rules. It employs a hash technique to reduce the size of the candidate itemsets and the database The *Continuous Association Rule Mining Algorithm (CARMA)* [Hidber 1998] allows the user to change the support threshold and continuously displays the resulting association rules with support and confidence bounds.

After some Update activities, new transactions are added to the database. When new transactions are added to the database, insignificant rules will be discarded. Similarly new valid ones that satisfy the statistical and the semantic constraints will be included. The A*daptive algorithm* [Sarda and Srinivas 1998] is not only incremental but also adaptive in nature. By inferring the nature of the incremental database, it can avoid unnecessary database scans. The *Fast Update algorithm (FUP)* is an incremental algorithm which makes use of past mining results to speed up the mining process [Cheung et al., 1996]. *Update Large Itemsets algorithm (ULI)* [Thomas et al., 1997] uses negative borders to decide when to scan the whole database. Recently *Fast Online Dynamic Association Rule Mining (FODARM) algorithm* [Woon et al., 2002] is introduced for incremental mining in electronic commerce. It uses a novel tree structure known as a *Support-Ordered Trie Itemset (SOTrieIT)* structure to hold pre-processed transactional data. Another algorithm for Online Generation of Profile Association Rules is introduced in [Aggarwal et al., 2002]. A New approach to Online Generation of Association Rules [Aggarwal and Yu 2001] introduces the concept of storing the preprocessed data in such a way that online processing may be done by applying a graph theoretic search algorithm whose complexity is proportional to the size of the output .

The algorithm presented in this paper *NBP: Negative Border with Partitioning* is based on partitioning the database, keeping a summary for each partition. This summary includes the locally large itemsets, their negative border and any other previously counted itemset in the partition. Another global summary including the large and negative border itemsets is also created for the whole database. When adding a new set of transactions to the database, the *NBP* applies the *Update Large Itemsets* (*ULI)*-like algorithm [Thomas et al., 1997] that uses these summaries instead of scanning the whole database, thus reducing the number of database scans to less than one scan.

The rest of the paper is organized as follows. The next section gives a description of the association rules mining (*ARM*) problem while section 3 presents the negative border with partitioning algorithm. Section 4 describes performance analysis of the proposed algorithm in comparison with some related algorithms. Finally conclusions are discussed in Section 5.

2 PROBLEM DESCRIPTION

The problem of association rules mining is described in the following two subsections.

2.1 Mining of association rules

The problem of mining association rules is described as follow: let the universal itemset, $I = \{i_1, i_2,.., i_m \}$ be a set of literals called *items* , D be a database of transactions, where each transaction T contains a set of items such that $T \subseteq I$. An *itemset* is a set of items and k-itemset is an itemset that contains exactly k items. For a given itemset $X \subseteq I$ and a given transaction T, T contains X if and only if $X \subseteq T$.

The *support count* σ_x of an itemset X is defined as the number of transactions in D containing X. An item set is large, with respect to a support threshold of s%, if $\sigma_x \geq |D| \times s$, where $|D|$ is the number of transactions in the database D. An association Rule is an implication of the form *"$X \Rightarrow Y$"* where $X, Y \subseteq I$ and $X \cap Y = \varnothing$. The association rule $X \Rightarrow Y$ holds in the database with confidence c% if no less than c% of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has support s% in D if $\sigma_{x \cup y} = |D| \times s\%$. For a given pair of confidence and support thresholds, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. This problem can be reduced to the problem of finding all large itemsets for the same support threshold [Agrawal et al., 1993].

2.2 Update of association rules

After some Update activities, new transactions are added to the original database D. When new transactions are added to the database, an old large itemset could potentially become small in the updated database. Similarly, an old small itemset could potentially become large in the new database.

Let Δ^+ be the set of newly added transactions (increment database), D' be the updated database where $D' = (D \cup \Delta^+)$, σ_x' be the new support count of an itemset X in the updated database D', $L^{D'}$ be the set of large itemset in D', C_k is the set of candidate k-itemsets in D and δ_x be the support count of an itemset X in the increment database Δ^+.

3 THE PROPOSED ALGORITHM

In this section, we develop an efficient algorithm for updating the association rules when new transactions are added to the database. The proposed algorithm uses negative borders [Thomas et al., 1997]. The intuition behind the concept of negative border is that, for a given set of large itemsets L, the negative border *NBd(L)* contains the closest itemsets that could be large too.

The list of symbols that used in our algorithm is shown in Table 1.

Table 1: Symbols of the proposed algorithm *NBP*.

Symbol	Definition
q	Partition size
$\|p_i\|$	Cardinality of partition p_i
L_{pi}	Large itemset of partition p_i
$NBd(L_{pi})$	Negative border itemset of partition p_i
$NBd(L^D)$	Negative border itemset of original database D
$NBd(L^{D'})$	Negative border itemset of updated database D'
n	Total Number of partitions

3.1 Algorithm Description

The new algorithm can be described in two main steps.

Preprocessing step

In this step we divide the original transactional database into number of partitions with size q. For simplicity we assume q as a multiple of $|\Delta^+|$ (for generality q can be of any size). Total number of partitions is assumed to be n. In the preprocessing step we evaluate for each partition p_i; i =1, 2... n the large itemset L_{pi} with its corresponding negative border itemset $NBd(L_{pi})$. Each itemset in L_{pi} or $NBd(L_{pi})$ is stored with its corresponding support count in the partition p_i. Also we compute the large itemset L^D and negative border itemset $NBd(L^D)$ for the whole database D.

The size of large and negative border itemsets for all partitions may be big enough not to be fitted into the computer memory due to memory limitations, so we suggest adding another support threshold *ls* "*local support threshold*" as a fraction of the global support threshold *s* which limits the size of negative border itemsets for all partitions. Itemsets with support count less than $ls \times s \times |pi|$ are discarded from the negative border itemset of the partition p_i. The large itemsets for all partitions is kept unchanged as they contain significant information of most frequent itemsets.
The Pseudo code of preprocessing step is described in both Figures 1and 2.

```
Function preprocessing (D, n,q)
Divide the original database into n partitions, each partition with size q.
for i = 1 to n do
        $L_{pi}$ = large-itemset for partition $p_i$
        $NBd(L_{pi})$=Negtiveborder_gen ($L_{pi}$)
$L^D$ = large-itemset for the whole database
$NBd(L^D)$ = Negtiveborder_gen ($L^D$)
```

Figure 1: A high-level description of preprocessing step

```
Function Negtiveborder_gen (L)
Split L into $L_1, L_2,.., L_r$ where r is the size of the largest itemset in L
For all k=1, 2, ….,r do
    Compute $C_{k+1}$ using apriori-gen($L_k$)
//Apriori[Agrawal et al., 1994]
$L \cup NBd(L) = U_{i=2,....,r+1} C_k \cup I_1$ , where $I_1$ is the set of 1-itemset.
```

Figure 2: The of Negativeborder_gen function.

Updating Step

In this step we have the incremental database Δ^+, set of n partitions with their corresponding large and negative border itemsets. With the assumption that partition size q is multiple of the incremental database size, Δ^+ is added either to the last partition or to a new partition. The next step is to update the large itemsets L_{pn} and negative border itemsets $NBd(L_{pn})$ of the partition p_n using $NBP(p_n, \Delta^+, p_n)$ function. If Δ^+ is added to a new partition evaluate L_{pn} and $NBd(L_{pn})$ using *Apriori* as a level wise algorithm (*Apriori* [Agrawal et al., 1994] generates only candidate itemsets, we get the negative border itemsets by applying the function *Negtiveborder_gen(L)* to the resulting large itemset from *Apriori* algorithm .

After updating the partitions, the next step is to update L^D and $NBd(L^D)$ of the whole database to obtain the updated large itemset $L^{D'}$ and updated negative border itemset $NBd(L^{D'})$. First we compute the large itemset $L^{\Delta+}$ and negative border itemset $NBd(L^{\Delta+})$ of Δ^+, simultaneously we count the support for all itemsets $t \in L^D \cup NBd(L^D)$ in Δ^+. If an itemset $t \in L^D$ or $NBd(L^D)$ has minimum support in D', then t is added to $L^{D'}$ otherwise it is added to $NBd(L^{D'})$. For each itemset $x \in L^{\Delta+} \cup NBd(L^{\Delta+})$, $x \notin L^D$ and $x \notin NBd(L^D)$ add x into $NBd(L^{D'})$.The change in L^D could potentially change $NBd(L^D)$ also.

Therefore some itemsets may be missed in both $L^{D'}$ and $NBd(L^{D'})$. We define two sets *Large_to_Large* (set of itemsets that moved from L^D to $L^{D'}$) and *Negative_to_Large* (set of itemsets that moved from $NBd(L^D)$ to $L^{D'}$). Join *Negative_to_Large* with *Large_to_Large* to get new set *Self_Join_Set*, using the function join (L_{k-1}) which join a set of large itemsets with length (k-1) with itself to get C_k a set of candidate itemsets with length k. the *join* (L_{k-1}) function is described in Figure 3.

```
function join ($L_{k-1}$)
$C_k = \Phi$
For each $X, Y \subset L_{k-1}$ do
        if $X.item_1 = Y.item_1, .., X.item_{k-2} = Y.item_{k-2}$, $X.item_{k-1} < Y.item_{k-1}$ then
                $Z = X.item_1, X.item_2, .., X.item_{k-1}, Y.item_{k-1}$
                Insert $Z$ into $C_k$
//pruning step
For all itemsets $c \in C_k$ do
        For all ($k$-1) subsets $s$ of $c$ do
                if ($s \notin L_{k-1}$) then
                        delete $c$ from $C_k$
Return $C_k$
```

Figure 3: High Level Description of the join function

For each itemset $t \in$ *Self_Join_Set*, check all partitions p_i, i = 1, 2, …, n. If t is found at the large itemset L_{pi} *or* negative border itemset $NBd(L_{pi})$, then update the support count of t. If t is not found in either L_{pi} or $NBd(L_{pi})$ then scan partition p_i to get the support count of t. Scanning a partition is done once for all itemsets need to be scanned in this partition. This means, we only need maximum of one scan for the whole database (all partitions) at worst case. In general, the proposed algorithm needs a fraction of a scan to update the large and negative border itemsets for the updated database. We use the hash tree structure (Apriori [Agrawal et al., 1994]) to get the support count of a set of itemsets within this partition. If the support count of $t \geq$ the support threshold of D', then add t to $L^{D'}$; otherwise add t to $NBd(L^{D'})$.

The description of the *NBP* (*D*, Δ^+, *Partitions*) function is described Figure 4.

```
Function NBP ( D, $\Delta^+$, Partitions)
$L^{D'} = \Phi$ , $NBd(L^{D'}) = \Phi$, $Large\_to\_Large = \Phi$ and $Negative\_to\_Large = \Phi$ // initialization
Compute $L^{\Delta+}$, $NBd(L^{\Delta+})$
If $|p_n| < q$ then
    Add $\Delta^+$ to $p_n$ and update the partition $p_n$
else $n$++, Add $\Delta^+$ to the new partition ,Compute $L_{pn}$, $NBd(L_{pn})$
For each itemset $x \in L^D$
    if ($\sigma_x + \delta_x \geq s * (|D| + |\Delta^+|)$) then
//s :minimum support threshold
            add $x$ to both $L^{D'}$ and Large_to_Large sets
    else add $x$ to $NBd(L^{D'})$
For each itemset $x \in NBd(L^D)$
    if ($\sigma_x + \delta_x \geq s * (|D| + |\Delta^+|)$) then
        add $x$ to both $L^{D'}$ and Negative_to_Large sets
    else add $x$ to $NBd(L^{D'})$
For each itemset $x \in L^{\Delta+} \cup NBd(L^{\Delta+})$ , $x \notin L^D$ and $x \notin NBd(L^D)$ do
    add $x$ to $NBd(L^{D'})$
if $L^D \neq L^{D'}$ then
    ULNBd($L^{D'}$, $NBd(L^{D'})$, Large_to_Large, Negative_to_Large, Partitions)
```

Figure 4: Negative Border with Partitioning algorithm using function *NBP* ( )

The pseudo code of the function *ULNBd ()* is given in Figure5.

```
ULNBd ($L^{D'}$,$NBd(L^{D'})$,Large_to_Large, Negative_to_Large, Partitions)
// generate all possible candidates "Self_Join_Set"
//for the set of large items in the updated database $D'$
$Self\_Join\_Set_1 = \Phi$
// initialize Self_Join_Set of length1 to be empty
For $k$ = 1, 2, ….,ℓ do
 //ℓ:size of the largest itemset in Negative_to_Large
    $LL_k$= set of itemsets with length $k$ from Large_to_Large
    $NL_k$ = set of itemsets with length $k$ from Negative_to_Large
    $Self\_Join\_Set_{k+1}$ = join($LL_k \cup NL_k \cup Self\_Join\_Set_k$)
For $i$=1, 2, ..., $n$ do     // $n$: number of partitions
    $p_i$ _itemsets= $\Phi$ // $p_i$ _itemsets: set of items to be scanned within a partition $p_i$
For each itemset $t \in$ Self_Join_Set do
    $\sigma_t' = 0$        // initialize support count of itemset $t$
    For $i$=1, 2, ..., $n$ do
    // search all partitions for the support count of all elements found in Self_Join_Set
        if $t \in L_{pi}$ then
            $\sigma_t' = \sigma_t'$ + support count of $t$ in $L_{pi}$
        else if $t \in NBd(L_{pi})$ then
            $\sigma_t' = \sigma_t'$ + support count of $t$ in $NBd(L_{pi})$
            else  add $t$ to $p_i$ _itemsets
For $i$=1, 2, ..., $n$ do
    if $p_i$ _itemsets $\neq \Phi$ then
        Scan $p_i$ to get support count of each itemset
         $x \in p_i$ _itemsets
```

```
//scanning using hash tree structure // (Apriori [Agrawal et al.,
        1994])
For each itemset $t \in$ Self_Join_Set do
//Update support count of t after scanning all
//partitions
    if $\sigma_t' \geq$ minsup*(|D| +| $\Delta^+$|)  then
        add t to $L^{D'}$
    else   add t to $NBd(L^{D'})$
```

Figure 5: Update Large and Negative Border of D' using *ULNBd* () function

The number of scans over the whole database needed for *NBP* algorithm is varying from 0 to 1. The zero scan is obtained when the information needed after adding the increment database is found in either the global summary of the whole database or the local summary in each partition. The one scan is occurred at the worst case when the algorithm needs to scan all partitions (whole database) to get the count of some itemsets. In general, the algorithm needs a fraction of a scan to reach the final results

4 PERFORMANCE ANALYSIS

In this section, the proposed algorithm is tested using several test data to show its efficiency in handling the problem of incremental mining of association rules.

4.1 Generation of synthetic data

In this experiment, we used synthetic data as the input database to the algorithms. The data are generated using the same technique as introduced in [Agrawal et al., 1994], modified in [Park et al., 1997] and used in many algorithms like [Thomas et al., 1997] and [Cheung et al., 1996]. Table 2 gives a list of the parameters used in the data generation method.

Table 2: Parameters for data generation

$\mid D \mid$	Number of transactions in original database
$\mid D' \mid$	Number of transactions in the updated database
$\mid \Delta^+ \mid$	Number of added transactions
$\mid T \mid$	Mean size of transactions
$\mid I \mid$	Mean size of potentially large itemsets
$\mid £ \mid$	Number of potentially large itemsets
N	Number of items

We use the notation T*x*.I*y*.D*i*+*d* , and modified from the one used in [Agrawal et al., 1994] , to denote an experiment using databases with the following sizes |D| = *i* thousands, Δ^+=*k* thousands, |*T*|= *x*, and |*I*|=*y* . In the Experiments we set N=1000 and |£|=2000.The increment database is generated as follow: we generate 100 thousand transactions, of which (100-*d*) thousands is used for the initial computation and *d* thousands is used as the increment, where *d* is the fractional size (in percentage) of the increment.

4.2 Experimental Results

In each experiment, we run the proposed algorithm *NBP* on the previous test data. We compare the execution time of the incremental algorithm *NBP* with respect to running *Apriori* on the whole data set. The proposed algorithm is tested using the settings T10.I4.D100+*d*. The support threshold is varying between 0.5% and 3.0%. For simplicity we assumed that the partition size *q* is a multiple of the size of the increment database | Δ^+| .We run the algorithm for *q* = 1, 2, 5, 10 multiples of | Δ^+| and *d* = 1% as a fraction from the whole database size. Figure 6 shows the speed up of the incremental algorithm over *Apriori* with support threshold is varying between 0.5% and 3.0%. It can be shown that when applying the *NBP* algorithm on the test data it achieves an average speed up ranging from 6 to 67 in comparison with *Apriori* algorithm.

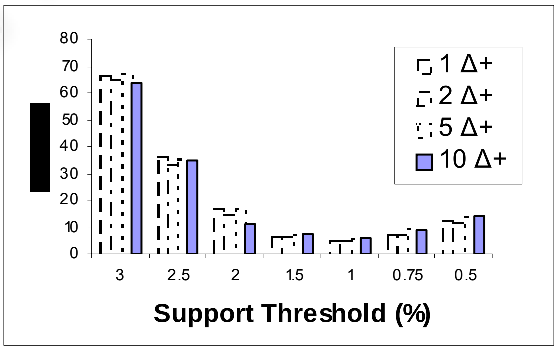

Figure 6: Performance Ratio of *NBP* at Δ^+ = 1%

Figures 7 and 8, show the experimental results when applying the new algorithm *NBP* on the same test data but with *d* is 2% and 5% respectively. The support threshold is varying between 0.5%and 3.0% in both experiments. It can be concluded from Figure 7 that the proposed algorithm has an average speed up ranging from 4 to 37 in comparison with *Apriori* algorithm. From Figure 8 the *NBP* algorithm achieves an average speed up ranging from 2 to 14.

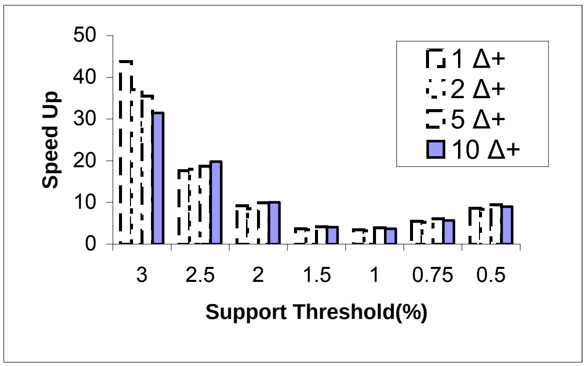

Figure 7: Performance Ratio of *NBP* at Δ^{+} = 2%

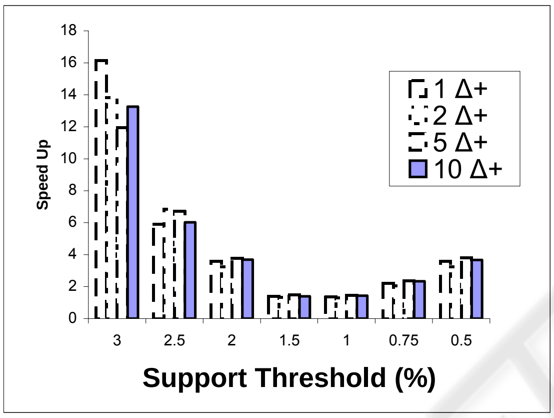

Figure 8: Performance Ratio of *NBP* at Δ^{+} = 5%

From Figures 6, 7 and 8, it is noticed that the proposed algorithm shows better performance for high support than low support. At high support thresholds, the possibility to get new large itemsets from the original negative border is low so the searching time within the partitions' large itemsets and negative border itemsets is small. At low support thresholds, there is a high probability of getting more new large itemsets immigrating from the set of negative border to the set of large itemsets. This increases the possibility to scan most partitions causing the increase of execution time. Also, the speed up of the proposed algorithm is higher for smaller increment sizes since the new algorithm needs to process less data. It can be shown that the *NBP* algorithm achieves better performance when the partition size is five times of the increment database Δ^{+} and the size of increment database is 1% of the whole database.

4.3 Comparisons with *FUP*

FUP may require *O (k)* scans over the whole database where *k* is the size of maximal large itemsets, while the new NBP algorithm needs a fraction of a scan to update the results. In this experiment, we run the proposed algorithm *NBP* on the previous test data. We compare the execution time of the incremental algorithm *NBP* with respect to running *FUP* on the same data set. For support threshold varying between 1.0% and 3.0%, and $|\Delta^{+}|$ = 1 % Figure 9 shows that the proposed *NBP* algorithm has an average speed up ranging from 6 to 67 while *FUP* algorithm achieves a speed up from 2 to 7 against *Apriori* algorithm.

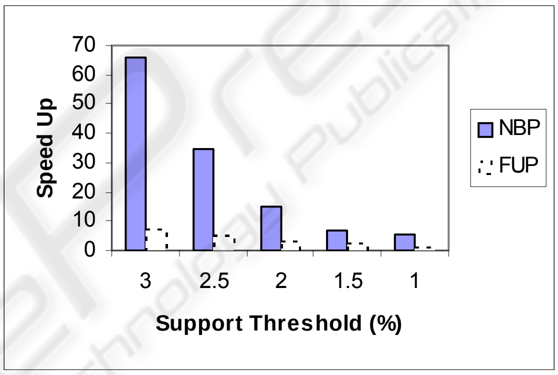

Figure 9: Speed up of *NBP* against *FUP*

4.4 Comparisons with *ULI*

It is costly to run *ULI* at high support thresholds where the number of large itemsets is less and at low support threshold the probability of the negative border expanding is higher so *ULI* may have to scan the whole database. We run the proposed algorithm *NBP* on the previous test data and compare the execution time of the incremental algorithm *NBP* with respect to running *ULI* on the same data set.It is concluded from Figure 10 that for support threshold varying between 0.5%and 3.0%, and $|\Delta^{+}|$ = 1 % The *NBP* algorithm has an average speed up ranging from 6 to 67 while *ULI* algorithm achieves a speed up from 5 to 20 against *Apriori* algorithm.

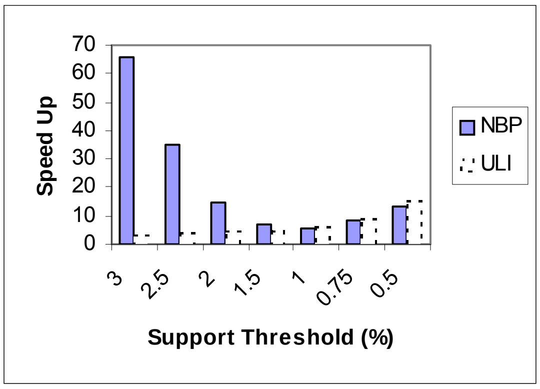

Figure 10: Speed up of *NBP* against *ULI*

5 CONCLUSIONS

In this paper a new algorithm *NBP: Negative Border with Partitioning* is presented for incremental mining of association rules. The proposed algorithm is based on partitioning the database, keeping a summary for each partition. Another global summary including the large and negative border itemsets is also created for the whole database. When adding a new set of transactions to the database, the *NBP* applies a *ULI*-like algorithm that uses these summaries instead of scanning the whole database, thus reducing the number of database scans to less than one scan. From algorithm discussion and experimental results, the following points can be concluded:

1. The new algorithm *NBP,* can efficiently handle the problem of incremental mining of association rules. *NBP* shows better performance than the algorithms of *FUP* and *ULI.*
2. The number of scans over the whole database needed for *NBP* algorithm is varying from 0 to 1.
3. *NBP* achieves high speed up from 6 to 67 for support threshold varying from 0.5 to 3.0 against the *Apriori* algorithm.

REFERENCES

Agrawal, R. ,Imielinski, T. and Swami, A., 1993. Mining Association Rules between Sets of Items in Large Databases. *Proc. ACM SIGMOD. Int Conf,* 1993.

Agrawal, R. and Srikant, R..Fast Algorithms for Mining Association Rules .*Proc.*(*VLDB).Int Conf,* 1994.

Cheung, D.W. Lee, S.D. and Kao, B. A General Incremental Technique for Maintaining Discovered Association Rules. *Proc. Database systems for Advanced Applications, Int Conf,* 1998.

Park, J.S. Chen, M.S. and Yu, P.S.. Using a Hash Based Method with Transaction Trimming for Mining Association Rules. *IEEE Trans on Knowledge and Data Engineering,* 1997.

Agrawal, C.C. and Yu, P.S. Mining Large Itemsets for Association Rules, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 1998.

Sarasere, A. Omiecinsky, E. and Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Databases. *Very Large Databases (VLDB). Int Conf.* 1995.

Hidber,C.Online Association Rule Mining.*Proc.ACM SIGMOD Int Conf. Management of Data*, 1998.

Han, J. ,Pei, J. and Yin, Y. Mining frequent patterns without candidate generation. *Proc. ACM SIGMOD. Int Conf. on management of Data,*2000.

Woon , Ng, Y. W. and Das, A. , Fast Online Association Rule Mining , *IEEE transactions on Knowledge and Data Engineering* ,2002.

Sarda, N.L. and Srinivas, N. V.An Adaptive Algorithm for Incremental Mining of Association Rules. *Proc .Database and Experts systems. Int Conf* , 1998.

Thomas, S., Bodagala, S. Alsabti, K. and Ranka, S.. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. *Proc. Knowledge Discovery and Data Mining (KDD 97). Int conf,* 1997.

Aggarwal, C.C., Sun, Z. and Yu, P.S., Fast Algorithms for Online Generation of Profile Association Rules, *IEEE transactions on knowledge and Data Engineering*, September 2002.

Cheung, D.W. Han, J. Ng, V.T. and Wong, C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proc. Data Engineering. Int Conf,* 1996.

Aggarwal, C. and Yu, P. A new Approach for Online Generation of Association Rules, *IEEE transactions on Knowledge and Data Engineering, 2001*