NBP: Negative Border with Partitioning Algorithm for Incremental Mining of Association Rules

Yasser El.Sonbaty
Arab Academy for science and Technology
College of Computing and Information Technology
P.O.box. 1029 Abu Kir, Alexandria
Egypt
Yasser@aast.edu

Rasha F. Kashef
Arab Academy for science and Technology
College of Computing and Information Technology
P.O.box. 1029 Abu Kir, Alexandria
Egypt
rashak@aast.edu

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 [1-3][5][10][12]. 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 3. Recently, many interesting researches have been published in association rules mining including mining of quantitative association rules, multi-level association rules and parallel and distributed mining of association rules [1-4][6][9][10].

As the database grows and more transactions are submitted, the previously discovered rules have to be maintained, thus discarding the rules that become statistically insignificant and including new valid ones that satisfy the statistical and the semantic constraints. In these situations, conventional algorithms must re-process the entire updated databases to find final association rules for each newly added set of transactions. This strategy has proven to be inefficient in terms of time and space complexities, since it does not benefit from the previously discovered information. Many algorithms were reported in the literature [11] [13 -15] [17] [18] for handling the problem of incremental mining of association rules. A brief discussion about these algorithms is shown in section 2. 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 for the whole database is also created. When adding a new set of transactions to the database, the *NBP* applies the Update Large Itemsets (*ULI*)-like algorithm [13] 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 reviews related work. Section 3 gives a description of the problem while section 4 presents the proposed algorithm. Section 5 describes performance analysis of the proposed algorithm in comparison with some related algorithms. Finally conclusions are discussed in Section 6.

2. RELATED WORK

The *Apriori* algorithm [2] 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. The idea rests on the fact that any subset of large itemset must be large as well. Hence, C_k can be generated by joining L_{k-1} and deleting candidates that contain any subset that is not large. This would result in a significantly smaller number of generated candidate itemsets. After *Apriori*, the *Direct Hashing and Pruning (DHP) algorithm* [4] is the next used algorithm for the efficient mining of association rules. It employs a hash technique to reduce the size of the candidate itemsets and the database. *DHP* has significant speed improvements due to the reduced size of the candidate itemsets generated. However, it causes additional overheads due to the need to do hashing and to maintain a hash table.

Unlike the above discussed algorithms, the *Continuous Association Rule Mining Algorithm (CARMA)* [7] allows the user to change the support threshold and continuously displays the resulting association rules with support and confidence bounds during the first scan or phase. During the second phase, it determines the precise support of each item set and extracts out all the large itemsets. Incremental mining is brought to another new level when the *adaptive algorithm* [11] is introduced. This algorithm is not only incremental but also adaptive in nature. By inferring the nature of the incremental database, it can avoid unnecessary database scans. The cost of maintaining association rules can be considered once we know the type of incremental database being mined. The *Fast Update algorithm (FUP)* is an incremental algorithm which makes use of past mining results to speed up the mining process [15], moreover, at finding the new large itemsets, the pool of candidate itemsets can be pruned substantially. Also *FUP* uses some optimization techniques for reducing the database size during the update process. Basically, the framework of *FUP* is similar to that of *Apriori* [2] and *DHP* [4]. It contains number

of iterations. The iteration starts at the 1-itemsets, and at each iteration, all the large itemsets of the same size are found. Moreover, at each iteration the candidate sets are generated based on the large itemsets found at the previous iteration. Another version of *FUP* that deals with the case of transaction deletion is *FUP2* [16], it is based on the same technique on *FUP* with addition to some optimization methods. *Update Large Itemsets algorithm (ULI)* [13] uses negative borders to decide when to scan the whole database and it can be used in conjunction with any level-wise algorithm like *Apriori* or *DHP*. The intuition behind the concept of negative border is that for a given set L_k, the negative border contains the closest itemsets that could be large too. First we compute the large itemsets of the increment database. The algorithm requires a full scan of the whole database only if the negative border of the large itemsets expands, that is, if an itemset outside the negative border gets added to either the large itemsets or its negative border. Even in such cases, it requires only one I / O Pass over the whole data set. Recently *Fast Online Dynamic Association Rule Mining (FODARM) algorithm* [18] is introduced for 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. It allows *FODARM* to generate large 1-itemsets and 2-itemsets quickly without scanning the database. In addition, the *SOTrieIT* structure can be easily and quickly updated when transactions are added or removed. Another algorithm for online generation of profile association rules is introduced in [14]. The concept of profile association rules discusses the problem of relating consumer buying behavior to profile information. A New approach to online generation of association rules [17] 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 .

3. PROBLEM DESCRIPTION

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

3.1 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 [1].

3.2 Updating association rules

After some update activities, a new set of transactions is 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 itemsets 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^+.

4. THE PROPOSED ALGORITHM

In this section, we develop an efficient algorithm for updating the association rules when a new set of transactions is added to the database. The main objective of the proposed algorithm is to minimize the number of scans needed to update the association rules. This is to be done by partitioning the database into n partitions, keeping a local summary for each partition. These local summaries include large and negative border itemsets for each partition. In addition, a global summary is kept for the whole database; this global summary contains the large and negative border for the whole database. When updating the database in terms of adding a new set of transactions to the database, the proposed algorithm *NBP* applies the Update Large Itemsets *(ULI)*-like algorithm [13] that uses these summaries instead of scanning the whole database, thus reducing the number of database scans to a fraction of one scan. The list of symbols that used in the proposed 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

4.1 Algorithm description

The new algorithm can be described in two main steps preprocessing step and updating steps. These steps are described as follow:

Preprocessing step

In this step we divide the original transactional database into n partitions each partition with size q. For simplicity we assume q as a multiple of $|\Delta^+|$ (for generality q can be of any size). 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 evaluation of negative border itemsets for all the partitions and for the whole database is done by calling the function *Negtiveborder_gen (L)* that is described in Fig. 2. The *Negtiveborder_gen* takes a set of large itemsets as input parameters and generates a set of negative border itemsets. The Pseudo code of preprocessing step is described in Figures 1, 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)

Fig. 1. A high-level description of preprocessing step

Function Negtiveborder_gen (L)

Split L into L_1 , L_2,.., L_r , 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[2]

$L \cup NBd(L) = U_{i=2,\ldots,r+1} C_k \cup I_1$, *where* I_1 *is the set of 1-itemset*

Fig. 2. The of Negativeborder_gen function.

Updating Step

After the preprocessing step, we have a set of n partitions with their corresponding large, negative border itemsets and the large, negative border itemsets for the whole database. For updating the database D a new set of transactions Δ^+ is added to the database. With the assumption that partition size q is multiple of the increment database size $|\Delta^+|$, Δ^+ is added either to the last partition or to a new partition according to space availability. If Δ^+ is added to a new partition, the large and negative border itemsets are evaluated by using *Apriori* as a level wise algorithm (*Apriori* [2] generates large itemsets, and we can get the negative border itemsets by applying the function *Negtiveborder_gen(L)* to the resulting large itemset obtained from *Apriori* algorithm). If Δ^+ is appended to the last partition then we have to update the large itemsets L_{pn} and negative border itemsets $NBd(L_{pn})$ of the partition p_n using the Negative Border with Partitioning function $NBP(p_n, \Delta^+, p_n)$. *NBP* function takes the original database, the increment database and set of partitions to search through as input parameters and it updates the large and negative border itemsets of the original database as shown in Fig. 3.

After updating the last partition, the next step is to update large L^D and negative border itemsets $NBd(L^D)$ of the whole database D to obtain the updated large and negative border itemset of the updated database D' . First we compute the large and negative border itemset of the increment database Δ^+, simultaneously we evaluate the support count for all itemsets x belonging to both large and negative border itemsets of the original database in Δ^+. If x passes the minimum support count in D', then x is added to the updated large itemset otherwise x is added to the updated negative border itemset. If some itemsets found in Δ^+ passed the minimum support count of the updated database while they are not found in either large or negative border itemsets of the database D then they can join the updated large itemset, otherwise they are added to the negative border itemset of the updated database. The proposed algorithm can be described in terms of the NBP (D, Δ^+, Partitions) function as described in Fig. 3.

Function NBP (D, Δ^+, Partitions)

$L^{D'} = \Phi$, $NBd(L^{D'}) = \Phi$, *Large_to_Large* = Φ and *Negative_to_Large* = Φ // initialization

Compute $L^{\Delta+}$, $NBd(L^{\Delta+})$ *//using Apriori [2]* and *Negtiveborder_gen* [13]

If $|p_n| < q$ then

Add Δ^+ to p_n and update the partition p_n

else n++, Add Δ^+ 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*

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*

else add x to $NBd(L^{D'})$

For each itemset $x \in L_1^{\Delta+} \cup NBd(L_1^{\Delta+})$, $x \notin L^D$ and $x \notin NBd(L^D)$ do

if $(\sigma_x + \delta_x \geq s * (|D| + |\Delta^+|)$ then

add x to $L^{D'}$

else 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)

Fig. 3 NBP algorithm using function *NBP* ( )

The updating process of L^D could also potentially change $NBd(L^D)$. Therefore some itemsets may be missed in both the updated large and negative border itemsets. We define two sets *Large_to_Large* (set of itemsets that still large after updating the database D) and *Negative_to_Large* (set of itemsets that moved from the original negative border itemset to the set of updated large itemset). We need to get all the possible candidates that can be generated from the set of itemsets that cross the negative border to become large itemsets, so we join *Negative_to_Large* with *Large_to_Large* to get a new set called *Self_Join_Set*, and this is to be accomplished by calling the function join (L_{k-1}) which joins a set of large itemsets with length (k-1) with itself to get C_k a set of candidate itemsets with length k. The function *join* (L_{k-1}) is described in Fig. 4.

function join (L_{k-1})

For each $X, Y \subset L_{k-1}$ do

if $X.item_1 = Y.item_1,\ldots,X.item_{k-2} = Y.item_{k-2}$, $X.item_{k-1} < Y.item_{k-1}$ then $Z = X.item_1, X.item_2,\ldots, X.item_{k-1}, Y.item_{k-1}$ Insert Z into C_k

For all itemsets $c \in C_k$ do *//pruning step*

For all (k-1) subsets x of c do

if $(x \notin L_{k-1})$ then delete c from C_k

Return C_k

Fig. 4. High Level Description of the join function

The updating of the support count of each itemset belongs to the *Self_Join_Set* set is done by calling the Update Large Negative Border function *ULNBd*. For each itemset $t \in$ *Self_Join_Set*, check all partitions p_i, $i = 1, 2, \ldots, n$. If t is found at the large itemset L_{pi} *or* negative border itemset $NBd(L_{pi})$ of the partition p_i, 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 [2]) to get the support count of a set of itemsets within this partition. If the support count of $t \geq$ minimum support count of D', then add t to updated large itemset $L^{D'}$; otherwise add t to updated negative border itemset $NBd(L^{D'})$. The pseudo code of the function *ULNBd ()* is given in Fig. 5.

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'

$Self\_Join\_Set_t = \Phi$

// initialize *Self_Join_Set* of length t to be empty

For $k = 1, 2, \ldots, \ell$ 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 each itemset $t \in$ *Self_Join_Set* do

For $i=1, 2, \ldots, n$ do

// search all partitions for updating the support count of all //elements found in *Self_Join_Set*

$\sigma_t' = 0$ // initialize support count of itemset t

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 $i=1, 2, \ldots, n$ 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 [2])

For each itemset $t \in$ *Self_Join_Set* do

if $\sigma_t' \geq s*(|D| + |\Delta^+|)$ then

add t to $L^{D'}$

else add t to $NBd(L^{D'})$

Fig. 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.

5. 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.

5.1 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 [2], modified in [4] and used in many algorithms like [13] and [15]. Table 5 gives a list of the parameters used in the data generation method.

Table 5: Parameters for data generation

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

In the following we use the notation Tx.Iy.Di+d , modified from the one used in [2] , to denote an experiment using databases with the following sizes $|D| = i$ thousands, $|\Delta^+|=k$ thousands, $|T|= x$, and $|I|=y$. In the Experiments we set N=1000 and $|\pounds|$=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.

5.2 Experimental results

In each experiment, we run the proposed algorithm *NBP* on the generated synthetic test data. We compare the execution time of the new incremental algorithm *NBP* with respect to running *Apriori* on the whole data set and some other relevant incremental algorithms. The proposed algorithm is tested using the settings T10.I4.D100+d. The support threshold is varying between 0.5% and 3.0%.

5.2.1 Changing partition and increment database sizes

For simplicity we assumed that the partition size q is a multiple of the size of the increment database $|\Delta^+|$. We run the algorithm for q = 1, 2, 5, 10 multiples of $|\Delta^+|$. Figures 7, 8 and 9, show the experimental results when applying the new algorithm *NBP* on the same test data with d is 1%, 2% and 5% respectively. The support threshold is varying from 0.5% to 3.0% in each experiment. It can be concluded from Figures. 7, 8 , and 9 that when applying the *NBP* algorithm on the test data it achieves an average speed up ranging from 6 to 67, from 4 to 37 and from 2 to 14 respectively in comparison with *Apriori* algorithm.

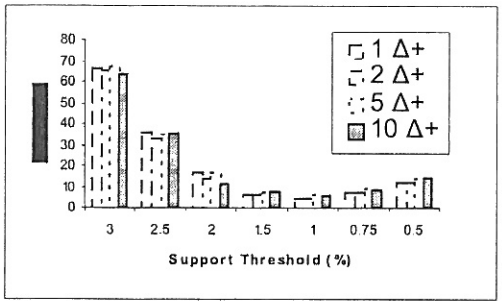

Fig. 7. Performance Ratio of *NBP* at $|\Delta^+|$= 1%

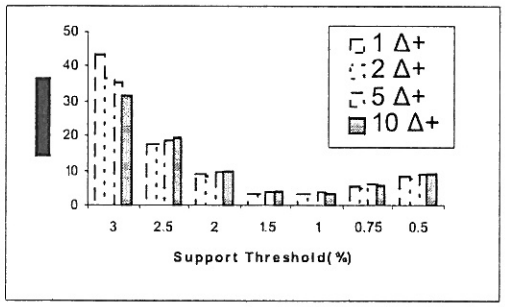

Fig. 8. Performance Ratio of *NBP* at $|\Delta^+|$= 2%

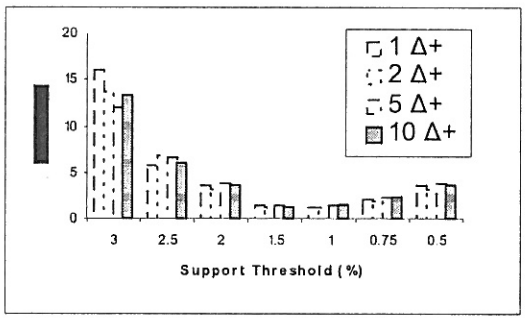

Fig. 9. Performance Ratio of *NBP* at $|\Delta^+|$= 5%

From Figures 7, 8 and 9, 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 large and negative border itemsets of the partitions 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 is also noticed 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.

5.2.2 *Comparison with FUP*

FUP [15] 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 both the proposed algorithm *NBP* and *FUP* algorithm on the previous test data used in section 5.2. For support threshold varying between 1.0% and 3.0%, and $|\Delta^+|$ = 1 % , Fig. 10 shows that the proposed *NBP* algorithm has an average speed up ranging from 4 to 12 against *FUP* algorithm.

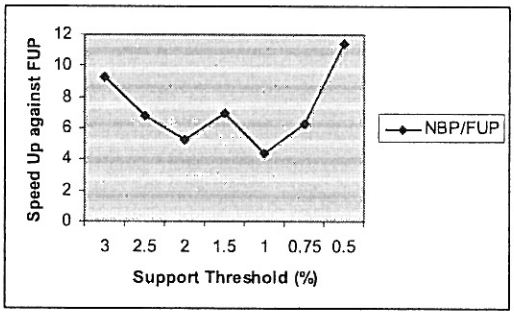

Fig. 10. Speed up of *NBP* against *FUP*

From Fig. 10, it is shown that at high support threshold, the proposed *NBP* algorithm may need a fraction of a scan or zero scan, while *FUP* needs *k* scans over the database (*k* is small at high support threshold), and that explains the better performance of *NBP* algorithm in comparison with *FUP* at high support threshold At low support threshold, *NBP* needs 1 scan (worst case) but *FUP* needs *k* scans (k is large at low support threshold). That is why the speed up of *NBP* is higher at low support threshold than at high support threshold in comparison with *FUP* algorithm. It is noticeable from Fig.10 that at average value of support threshold, the speed up of *NBP* in comparison with *FUP* may fluctuate; this is because the *NBP* reaches the case in which the algorithm generates a large number of local summaries at smaller support threshold or at sometimes there is a possibility to scan more partitions for discover the count of the new generated items.

5.2.3 *Comparison with ULI*

It is costly to run *ULI* [13] 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. In this experiment we run both the proposed algorithm *NBP* and *ULI* on the previous test data. It is concluded from Fig. 11 that for support threshold varying between 0.5%and 3.0%, and $|\Delta^+|$ = 1 % The *NBP* algorithm has an average speed up ranging from 0.8 to 20 against the *ULI* algorithm.

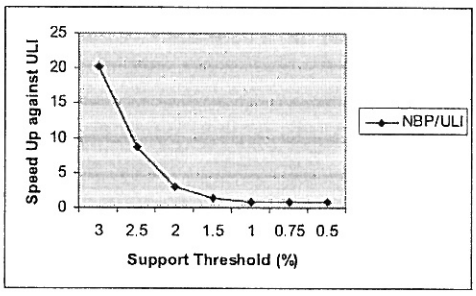

Fig. 11. Speed up of *NBP* against *ULI*

From Fig.11, at high support threshold *NBP* may need a fraction of a scan or zero scan, while *ULI* needs one scan over

the database. At low support threshold both *NBP* and *ULI* need almost one scan that is because *NBP* at worst case performs like *ULI* and may need a one scan over the database to discover the count of new generated itemsets. *NBP* performs like *ULI* at the worst case for low support threshold.

6. 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. 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 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.
2. 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.
3. *NBP* achieves high speed up from 6 to 67 for support threshold varying from 0.5% to 3.0% against the *Apriori* algorithm.
4. *NBP* shows better performance than the algorithms of *FUP* and *ULI.*

7. REFERENCES

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

[2] R. Agrawal, and R. Srikant. "Fast Algorithms for Mining Association Rules". *Proc.. Very Large Data bases .Int Conf,* 1994.

[3] R. Agrawal, and J. C. Shafer, "Parallel Mining of Association Rules", *IEEE Trans on Knowledge and Data Engineering*, 1996.

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

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

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

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

[8] J. Han, J. Pei, and Y. Yin. "Mining Frequent Patterns without Candidate Generation". *Proc. ACM SIGMOD. Int Conf. on management of Data,*2000.

[9] J. Han and Y.Fu. "Discovery of Multiple-Level Association Rules from Large Databases". *Proc. Very Large data Bases. Int conf,*1995.

[10] R. Srikant and R. Agrawal. "Mining Quantitative Association Rules in Large Relational Tables". *Proc. ACM SIGMOD .Int Conf on management of Data*, 1996.

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

[12] H. Mannila, H. Toivonen, and A. I. Verkamo, "Efficient Algorithms for Discovering Association Rules". *Proc. AAAI Workshop on Knowledge Discovery in databases (KDD-94)* ,1994.

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

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

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

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

[17] C. C. Aggarwal, and P. S. Yu," A New Approach for Online Generation of Association Rules", *IEEE transactions on Knowledge and Data Engineering, 2001*

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