

# A Review on Different Association Rule Mining Algorithms

Author: Shahana Sheikh<sup>1</sup>, Mr. Rupesh Patidar<sup>2</sup>

PG Scholar, CSE Department, Jawaharlal Institute of Technology, Borawan (M.P.)<sup>1</sup>,

Asst. Prof. of CSE Department, Jawaharlal Institute of Technology, Borawan (M.P.)<sup>2</sup>

E-mail: [shahanajit@gmail.com](mailto:shahanajit@gmail.com)<sup>1</sup>, [rupeshgovt@gmail.com](mailto:rupeshgovt@gmail.com)<sup>2</sup>

## ABSTRACT

The discovery of association rules has been known to be useful in selective marketing, decision analysis, and business management. An important application area of mining association rules is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased together. With the increasing use of the record-based databases whose data is being continuously added, recent important applications have called for the need of incremental mining. In dynamic transaction databases, new transactions are appended and obsolete transactions are discarded as time advances. In this paper, a review of different association rule mining algorithms like Apriori, FP-tree, FUP, FUP2, BORDER are present.

## Keywords

Data Mining, support, confidence, association rules.

## 1. INTRODUCTION

Data mining [1] is one of the fastest growing fields in the computer industry. Data mining is the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining, also known as knowledge discovery from database (KDD) [2] is an intelligent data analysis technique in the 1980's then gradually developed. Data mining is an approach to discover such useful information from very large and dynamic databases. This information includes association rules, characteristic rules, classification rules, generalized relations, discriminant rules, etc.

Association rule mining is an important topic in data mining field. Since it was introduced in 1993 by Agrawal et al, it has attracted a great deal of attention. Many efficient algorithms, extensions and applications have been reported. The classic application of association rule mining is market basket data analysis, which attempts to discover how items purchased by customers in a supermarket are associated. Association rules are of the form  $X \rightarrow Y$

, where  $X$  and  $Y$  are collection of items and intersection of  $X$  and  $Y$  are null.

The problem of association rule mining can be stated as follows: Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of items. Let  $T = \{t_1, t_2, \dots, t_n\}$  be a set of transactions, where each transaction  $t_i$  is a set of items such that  $t_i \subseteq I$ . An association rule is an implication of the form,  $X \rightarrow Y$ , where  $X \subset I$ ,  $Y \subset I$ , and  $X \cap Y = \emptyset$ .

### Support(S)-

Support(S) of an association rule is defined as the percentage/fraction of records that contain  $X \cup Y$  to the total number of records in the database. Suppose the support of an item is 0.1%, it means only 0.1 percent of the transaction contain purchasing of this item.

**Support (XY) = Support count of (XY) / Total number of transaction in D**

### Confidence(C)-

Confidence(C) of an association rule is defined as the percentage/fraction of the number of transactions that contain  $X \cup Y$  to the total number of records that contain  $X$ . Suppose the confidence of the association rule  $X \rightarrow Y$  is 80%, it means that 80% of the transactions that contain  $X$  also contain  $Y$  together.

**Confidence (X|Y) = Support (XY) / Support (X).**

The mining process of association rules can be divided into two steps-

(a) Frequent Itemset Generation

(b) Association Rule Generation

a. **Frequent Itemset Generation:** generate all sets of items that have support greater than a certain threshold, called minsupport.

b. **Association Rule Generation:** from the frequent itemsets, generate all association rules that have confidence greater than a certain threshold called min confidence [3].

Extracting association rules from the transactional databases, relational databases and

other information repositories which satisfies minimum support and minimum confidence is association rule discovery task. In real-world applications, new transactions are usually inserted into databases incrementally. In this case, the originally desired large itemset may become invalid, or new large itemset may appear in the resulting updated databases. The solution to batch mining algorithms such as Apriori and FP-tree is incremental mining algorithms. Incremental mining provides the solution to this problem. Incremental data mining tries to use the previous results as the basis to incrementally mine the database with new transactions. There are two performance issues on incremental data mining. The first one is the need to rescan the original database to enumerate the support count of the patterns when a database is updated. The second issue is how to deal with the threshold changes during the lifetime of the database.

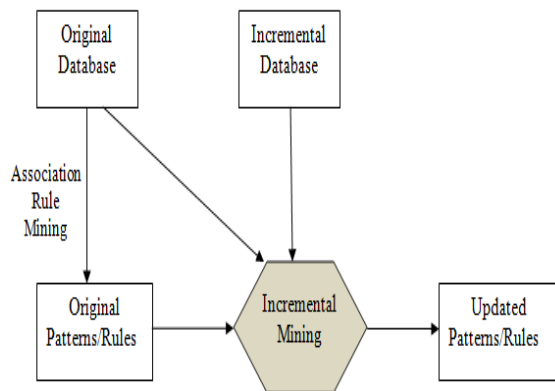


Figure 1: Process of Incremental Mining [4]

The strategies for mining frequent itemset, which is the essential part of discovering association rules, have been widely studied over the last decade such as the Apriori, and FPgrowth. Many researchers have proposed several algorithms to discover frequent pattern but most of them are static in nature. In 1994 Agrawal and Srikant proposed Apriori algorithm. It makes use of downward closure property. Han, Pei et al. proposed a data structure called FP-tree (frequent pattern tree). Fp-Growth approach is based on divide and conquers strategy for producing the frequent item sets. FP-growth is mainly used for mining frequent item sets without candidate generation. Cheung et al. thus proposed the FUP algorithm in 1996 to effectively handle new transactions for maintaining association rules. In 1997 Cheung et al. proposed a new algorithm. It is an extension of FUP algorithm. The Border algorithm is introduced in 1997 by Mannila and Toivonen., the algorithm makes one pass over the increment database, and at most one Pass over the whole database. In real world data is growing continuously. The static algorithms are not able to work efficiently whenever any change happens to the original database. Existing support as well as association rule may invalidate. If support is static (once it set to very high or low) important information loss may occur. Support should be flexible, based on real time event to avoid loss of important information.

## 2. Literature Review

Mining frequent item sets is an important problem in data mining and is also the first step of deriving association rules [5]. Association rule mining was first presented at 1993 by R. Agrawal, T. Imielinski, and A. Swami [5a]. After that many algorithms have been proposed and developed- Apriori [6], DHP [7], and FP growth [8]. The Apriori algorithm [9] employs a bottom-up breadth-first approach to get the large item set. As it was presented to hold the relational data this algorithm cannot be applied directly to mine complex data. Apriori Association rule mining technique uses a two step process. The first step is to identify all the frequent itemsets based on the support count value of the itemsets. It uses the downward closure property of itemsets to remove the infrequent itemsets. The second step is the generation of association rules from the frequent itemsets using the support and confidence [5].

Han j et al developed an algorithm for mining frequent patterns without candidate generation. It uses divide-and-conquer approach. First it calculates the frequent items and characterizes the frequent items in a tree called frequent-pattern tree. This tree can also exploit as a compressed database. The association rule mining is done on the compressed database with the help of this FP tree. This denotes that the dataset needs to be inspecting once. Also this algorithm does not need the candidate item set generation.

Another recognized algorithm is Border algorithm. The Borders algorithm is based on the notion of border sets, introduced in Mannila and Toivonen in 1997. This algorithm finds the frequent itemsets from the dynamic dataset, using the frequent itemsets already discovered from the old dataset.

For evolving databases there are a few incremental mining algorithms that work for exact data have been developed. Just For example in [10] the Fast Update algorithm (FUP) was proposed to efficiently maintain frequent item set & for a database to which new tuples are inserted. The proposed incremental mining framework is inspired by FUP. In [11] the FUP2 algorithm was developed to handle both addition and deletion of tuples.

## 3. Different Algorithms to Find Association Rules

### 3.1 Apriori Algorithm

Apriori is an algorithm [9] for frequent item set mining and association rule learning over transactional databases. Apriori was proposed by Agrawal and Srikant in 1994. The algorithm finds the frequent set L in the database D. It makes use of the downward closure property. The algorithm is a bottom search, moving upward level-wise in the lattice. However, before reading the database at every level, it prunes many of the sets which are unlikely to be frequent sets, thus saving any extra efforts.

**Candidate Generation-** Given the set of all frequent (k-1) item-sets. We want to generate superset of the

set of all frequent k-item-sets. The intuition behind the Apriori candidate generation procedure is that if an item-set X has minimum support, so do all subsets of X.

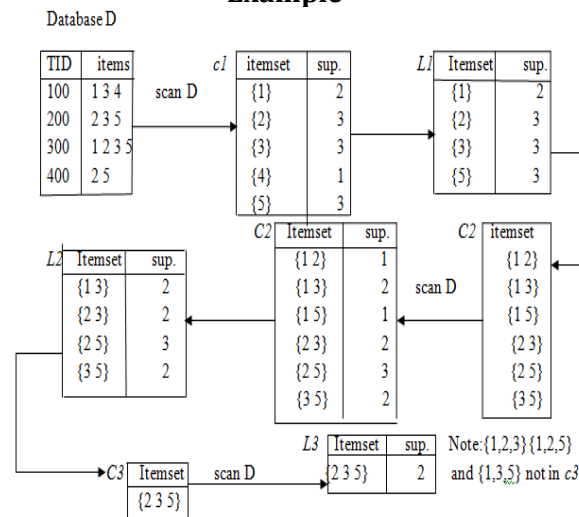
**Pruning:**The pruning step eliminates the extensions of (k-1) item-sets which are not found to be frequent, from being considered for counting support. For each transaction t, the algorithm checks which candidates are contained in t and after the last transaction are processed; those with support less than the minimum support are discarded.

**Algorithm**

```

L1 = {frequent 1-itemsets}; k = 2;
while (Lk-1 ≠ ∅) do
    Ck = candidate itemsets from Lk-1
    forall transactions t ∈ DBASE do
        forall candidate itemsets c ∈ Ck do
            count[c] = count[c] + 1
    Lk = All c ∈ Ck with minimum support
    k++
    
```

**Example**



It is no doubt that Apriori algorithm successfully finds the frequent elements from the database. But as the dimensionality of the database increase with the number of items then: more search space is needed and I/O cost will increase. Number of database scan is increased thus candidate generation will increase, results in increase in computational cost.

**3.2 FP-tree**

Han, Pei et al .proposed a data structure called FP-tree (frequent pattern tree). FP-Growth approach is based on divide and conquers strategy for producing the frequent item sets. FP-growth is mainly used for mining frequent item sets without candidate generation. The FP-Growth Algorithm is an alternative way to find frequent itemsets without using candidate generations, thus improving performance. In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one

associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity. In large databases, it's not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.

Major steps in FP-growth-

**Step1:** Build a compact data structure called FP tree. FP-tree is built using 2 passes over the dataset.

**Step2 :** It extract frequent item sets from FP-tree directly.

**FP-tree Example**

**STEP1**

Step 1: Scan DB for the first time to generate L<sub>1</sub>

TID	Items bought
100	{f, a, c, d, g, i, m, p}
200	{a, b, c, f, l, m, o}
300	{b, f, h, j, o}
400	{b, c, k, s, p}
500	{a, f, c, e, l, p, m, n}

Item	frequency
f	4
c	4
a	3
b	3
m	3
p	3

By-Product of First Scan of Database

Step 2: scan the DB for the second time, order frequent items in each transaction.

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

**STEP2**

Step 1: scan the DB for the second time, order frequent items in each transaction.

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Method: The FP-tree is constructed as follows.

1. Scan the transaction database *DB* once. Collect *F*, the set of frequent items, and the support of each frequent item. Sort *F* in support-descending order as *FList*, the list of frequent items.

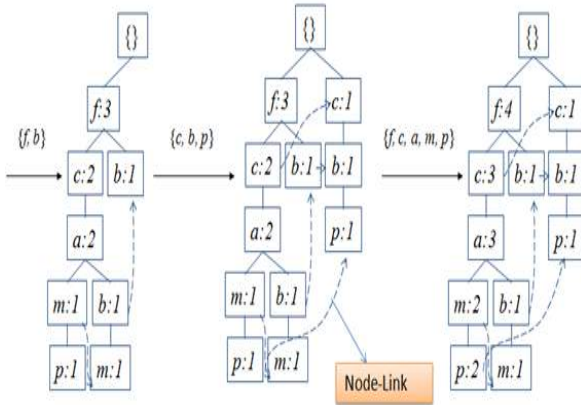
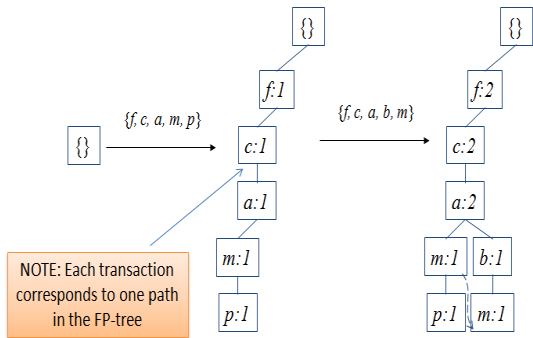
2. Create the root of an FP-tree, *T*, and label it as "null". For each transaction *Trans* in *DB* do the following.

Select the frequent items in *Trans* and sort them according to the order of *FList*. Let the sorted frequent-item list in *Trans* be  $[p | P]$ , where *p* is the first element and *P* is the remaining list. Call *insert tree*( $[p | P], T$ ).

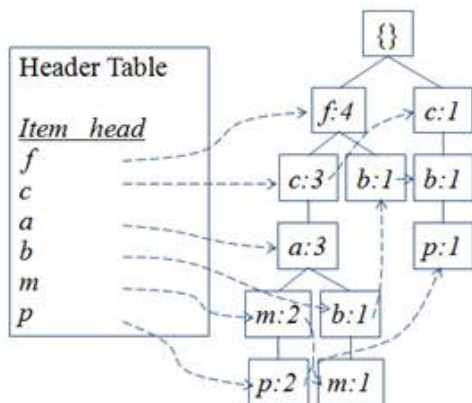
The function *insert tree*( $[p | P], T$ ) is performed as follows. If *T* has a child *N* such that *N.item-name* = *p.item-name*, then increment *N*'s count by 1; else create a new node *N*, with its count initialized to 1, its parent link linked to *T*, and its node-link linked to the nodes with the same *item-name* via the node-link structure. If *P* is nonempty, call *insert tree*(*P*, *N*) recursively.

FP-growth is faster than Apriori because—No candidate generation, no candidate test, Use compact data structure, Eliminate repeated database scan, Basic operation is counting and FP-tree building (no pattern matching). The disadvantage of FP-tree is that it may not fit in main memory. The most significant advantage of the FP-tree is that it Scan the DB only twice and twice only.

Step 2: construct FP-tree



Final FP-tree



**Algorithm - (FP-tree construction)**

**Input:** A transaction database *DB* and a minimum support threshold  $\xi$ .

**Output:** FP-tree, the frequent-pattern tree of *DB*.

**3.3 FUP**

FUP (fast update) [10] is the first algorithm of incremental association rule mining. Cheung et al. thus proposed the FUP algorithm (Cheung et al., 1996) to effectively handle new transactions for maintaining association rules. Considering an original database and some new inserted transactions, the following four cases (illustrated in Fig.2) may arise:

**Case 1:** An itemset is frequent both in an original database and in newly inserted transactions.

**Case 2:** An itemset is frequent in an original database but not frequent in newly inserted transactions.

**Case 3:** An itemset is not frequent in an original database but frequent in newly inserted transactions.

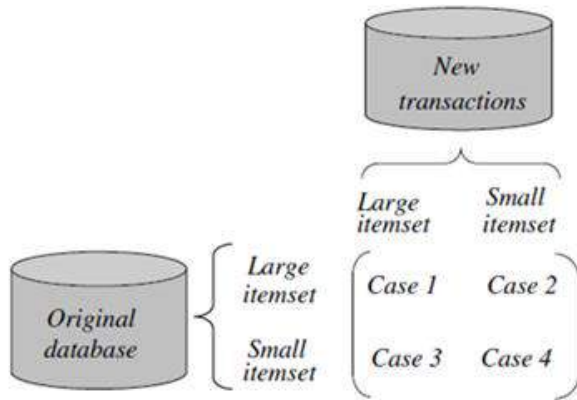


Figure2: four cases when new transactions are inserted into existing databases

**Case 4:** An itemset is not frequent both in an original database and in newly inserted transactions.

Since itemsets in Case 1 are large in both the original database and the new transactions, they will still be large after the weighted average of the counts. Similarly, itemsets in Case 4 will still be small after the new transactions are inserted. Thus Cases 1 and 4 will not affect the final large itemsets. Case 2 may remove existing large itemsets, and Case 3 may add new large itemsets. The FUP algorithm thus maintains large itemsets by considering the above four cases. In FUP, large itemsets with their counts in preceding runs are recorded for later use in maintenance. As new transactions are added, FUP first scans them to generate candidate 1-itemsets (only for these transactions), and then compares these itemsets with the previous ones. FUP partitions candidate 1-itemsets into two parts according to whether they are large for the original database. If a candidate 1-itemset from the newly inserted transactions is also among the large 1-itemsets from the original database, its new total count for the entire updated database can easily be calculated from its current count and previous count since all previous large itemsets with their counts are kept by FUP. Whether an original large itemset is still large after new transactions are inserted is determined from its updated support. By contrast, if a candidate 1-itemset from the newly inserted transactions does not exist among the large 1-itemsets in the original database, one of two possibilities arises. If this candidate 1-itemset is not large for the new transactions, then it cannot be large for the entire updated database, which means no action is necessary. If this candidate 1-itemset is large for the new transactions but not among the original large 1-itemsets, the original database must be re-scanned to determine whether the itemset is actually large for the entire updated database. Using the processing tactics mentioned above, FUP is thus able to find all large 1-itemsets for the entire updated database. After that, candidate 2-itemsets from the newly inserted transactions are formed and the same procedure is used to find all large 2-itemsets. This procedure is repeated until all

large itemsets have been found. A summary of the four cases and their FUP results is given in Table 1.

FUP is thus able to handle Cases 1, 2 and 4 more efficiently than conventional batch mining algorithms.

**Table 1**

Four cases and their FUP results

Cases: Original - New	Results
Case 1: Large - Large	Always large
Case 2: Large - Small	Determined from existing information
Case 3: Small - Large	Determined by rescanning original Database
Case 4: Small - Small	Always Small

### 3.4 FUP2

Cheung et al. [11] proposed a new algorithm FUP2 in 1997. It is an extension of FUP algorithm. The FUP updates the association rules in a database when new transactions are added to the database whereas FUP2 updates the existing association rules when transactions are added to and deleted from the database. It works with incremented database as well as decremented database. So i.e. it will handle deletion of transaction from old database also. FUP2 is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion. A very feature is that the old frequent  $k$  itemsets  $L_k$  from the previous mining result is used for dividing the candidate set  $C_k$  into two parts:  $P_k = C_k \cap L_k$  and  $Q_k = C_k - P_k$ . In other words,  $P_k$  and  $Q_k$  are the sets of candidate itemsets that are previously frequent and infrequent with respect to  $D$ . For the candidate itemsets in  $Q_k$ , their supports are unknown since they were infrequent in the original database  $D$ , posing some difficulties in generating new frequent itemsets. It is noted that if a candidate itemset in  $Q_k$  is frequent in  $\Delta^-$ , it must be infrequent in  $D^-$ . This itemset is further identified to be infrequent in the updated database  $D'$  if it is also infrequent in  $\Delta^+$ . This technique helps on effectively reducing the number of candidate itemsets to be further checked against the unchanged portion  $D^-$  which is usually much larger than  $\Delta^-$  or  $\Delta^+$ . For a general case that transactions are added and deleted, algorithm FUP2 can work smoothly with both the deleted portion  $\Delta^-$  and the added portion  $\Delta^+$  of the whole dataset. It gives poor result if it used with temporal database.

### 3.5 Border Algorithm

The Borders algorithm is [12] based on the notion of border sets, introduced in Mannila and Toivonen (1997). A set  $X$  is a border set if all its proper subsets are frequent sets (i.e., sets with at least minimum support), but it itself is not a frequent set.

The algorithm works as follows-

Initially, the  $L_{old}$  and  $B_{old}$  are known along with their respective support counts. The algorithm

starts by counting the supports of the itemsets of  $L_{old} \cup B_{old}$  in  $T_{new}$ . This requires one pass over the increment portion of the database. During this pass, the algorithm collects two categories of itemsets: F and B.

Set F contains the itemset of  $L_{old}$  which becomes a frequent set in  $T_{whole}$ . we should take care of the threshold count while counting the support. For example, consider that  $\sigma=5\%$ ,

$|T_{old}|=1000$ . Clearly, any itemset that is supported by 50 or more transactions in  $T_{old}$  is its frequent set. Let  $|T_{new}|=400$  and hence,  $|T_{whole}|=1400$ . The itemset X is frequent in  $T_{whole}$ , if it is supported by 70 or more transactions. Thus, while determining frequent sets in  $T_{old}$ , our threshold of absolute count is 50, whereas the same in  $T_{whole}$  is 70.

Set B contains all the border sets. If there is no promoted border, then F contains all the frequent sets of  $T_{whole}$ . But if there is at least one border set that becomes a promoted border, then the algorithm generates candidate sets which are super sets of the promoted border sets. It makes one pass over the database to count the support of these candidate sets. Thus, the algorithm makes one pass over the increment database, and at most one Pass over the whole database.

#### Algorithm

Read T-new and increment the support count of X for

$$\text{all } X \in L_{old} \cup B_{old}$$

$$F := \{X \mid X \in L_{old} \text{ and } s(X)^{T_{whole}} \geq \sigma\}$$

$$B := \{X \mid X \in B_{old} \text{ and } s(X)^{T_{whole}} \geq \sigma\}$$

Let m be size of the largest element in B.

Candidate-generation

for all itemsets  $l_1 \in B_{k-1} \cup C_{k-1}$  do begin

for all itemsets  $l_2 \in B_{k-1} \cup F_{k-1} \cup C_{k-1}$  do begin

if  $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \dots \wedge l_1[k-1]$

$< l_2[k-1]$  then

$$c = l_1[1], l_1[2], \dots, l_1[k-1], l_2[k-1]$$

$$C_k = C_k \cup \{c\}$$

end do

prune  $C_k$  for all k:

all subsets of k-1 size should be present in  $B_{k-1} \cup$

$$F_{k-1} \cup C_{k-1}$$

k:=k+1

$$\text{candidate } C := \cup C_k$$

read  $T_{whole}$  and count the support values of each item in C.

$$\text{new\_frequent\_sets} := \{X \mid X \in C \text{ and } s(X)^{T_{whole}} \geq \sigma\}$$

$$L_{whole} := F \cup \text{new\_frequent\_sets}$$

$$B_{whole} := (B_{old} \setminus B) \cup \{X \in C \text{ and } s(X)^{T_{whole}} < \sigma \text{ and}$$

all its subsets are in  $L_{whole}\}$

#### 4. Conclusion

In this paper, a review of different association rule mining algorithms like Apriori, FP-tree, FUP, FUP2, BORDER are present. Each algorithm has its own benefits and drawbacks. FP-growth is faster than Apriori because-No candidate generation, no candidate test, Use compact data structure, Eliminate repeated database scan. FUP algorithm works only for incremented database not for decremented database. To overcome the problem of FUP a new extension FUP2 has been developed. FUP2 not only works for incremented database but it will also work for decremented database.

#### 5. References

- [1] Chen, M. S., Han, J., & Yu, P. S. – “Data mining: An overview from a database perspective”, IEEE Transactions on Knowledge and Data Engineering, 8(6), 866-883, 1996
- [2] Tan Pang-Ning; Steinbach.M.; Kumar.V., “Introduction to Data Mining”. China Machine Press. 2010
- [3] Raymond Chi-Wing Wong, Ada Wai-Chee Fu, “Association Rule Mining and its Application to MPIS”, 2003.
- [4] V. Pudi, “Data Mining: Concepts and Techniques”, Oxford University Press, Jan-2009
- [5] R. Agrawal, T. Mielinski, A. Swami. (1993), “Mining association rule between sets of items in large databases”, in proceedings of the ACM SIGMOD international Conference on Management of data. pp: 207-216, 1993.
- [6] J. S. Park, M.-S. Chen, and P. S. Yu, “An effective Hash-Based Algorithm for Mining Association Rules”, Proceedings of the ACM SIGMOD, San Jose, CA, May 1995, pp. 175-186.
- [7] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, “Dynamic Itemset Counting and Implication Rules for Market Basket Data”, Proceedings of the ACM SIGMOD, Tucson, AZ, May 1997, pp. 255-264.
- [8] J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, Proceedings of the ACM SIGMOD, Dallas, TX, May 2000, pp. 1-12

[9] R. Agrawal and R. Shrikant, "Fast Algorithm for Mining Association Rules", Proceedings Of VLDB conference, pp 487 – 449, Santiago, Chile, 1994..

[10] D. Cheung, J. Han, V. Ng, and C. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," Proc. 12th Int'l Conf. Data Eng. (ICDE), 1996.

[11] D. Cheung, S.D. Lee, and B. Kao, "A General Incremental Technique for Maintaining Discovered Association Rules," Proc. Fifth Int'l Conf. Database Systems for Advanced Applications (DASFAA), 1997.

[12] Feldman R, Aumann Y and Lipshtat O(1999), "Borders : An Efficient Algorithm for Association Generation in Dynamic Databases", Journal of Intelligent Information System, Pages 61-73.

Ijournals