Research Journal of Information Technology 3(2): 79-90, 2011
ISSN:

Frequent Pattern Mining for Multiple Minimum Supports with Support Tuning and Tree Maintenance on Incremental Database

[1]F.A. Hoque, [2]M. Debnath, [2]N. Easmin and [3]K. Rashed
[1]Department of CSE, University of Information Technology Sciences (UITS), Dhaka, Bangladesh
[2]Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000
[3]Department of Computer Science and Engineering, Chittagong University of Technology (CUET), Bangladesh

Abstract: Mining frequent patterns in transactional databases is an important part of the association rule mining. Frequent pattern mining algorithms with single *minsup* leads to rare item problem. Instead of setting single *minsup* for all items, we have used multiple minimum supports to discover frequent patterns. In this research, we have used multiple item support tree (MIS-Tree for short) to mine frequent patterns and proposed algorithms that provide (1) a complete facility of multiple support tuning (MS Tuning), and (2) maintenance of MIS-tree with incremental update of database. In a recent study on the same problem, MIS-tree and CFP-growth algorithm has been developed to find all frequent item sets as well as to maintain MS tuning with some restrictions. In this study, we have modified the maintenance method by adding the benefit of flexible MS tuning without any restriction. Again, since database is subject to practice, an incremental updating technique has been proposed for maintenance of the MIS-tree after the database is updated. This maintenance ensures that every time an incremental database is added to the original database, the tree can be kept in correct status without costly rescanning of the aggregated database. Experiments on both synthetic and real data sets demonstrate the effectiveness of our proposed approaches.

Key words: Association rules, database, data mining, frequent pattern, minimum supports, support tuning

INTRODUCTION

Database mining has recently attracted tremendous amount of attention in the database research because of its wide applicability in many areas, including decision support, market strategy, financial forecast and bioinformatics. Many approaches have been proposed to find out useful and invaluable information from large amount of databases. One of the most important approaches is mining association rules, which was first introduced by Agrawal *et al*. (1993). Association rule mining finds interesting association relationships among a large set of items and it can be stated as follows:

Let, J= $\{i_1, i_2, \ldots, i_m\}$ be a set of items and D be a set of database transactions, where each transactions T is a set of items so that $T \subseteq J$. An association rule is an implication of the form $A \rightarrow B$, where $A \subset J$, $B \subset J$ and $A \cap B = \varnothing$. The rule $A \rightarrow B$ holds in the transaction set T with confidence c, if c% of transactions in T that support A also support B. The rule has support s in T if s% of the transactions in T contains $A \cup B$. The problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence(called *minconf*). The key element that makes association-rule mining practical is *minsup*. It is used to prune the search space and to limit the number of rules generated. There are different methods for mining the simplest form of association rules with single *minsup*. Of these, Apriori algorithm, Agrawal and Srikant (1994) performs a breadth-first search in the search space by generating candidate *k*+1- itemsets from frequent *k*-itemsets. The frequency of an itemset is computed by counting its occurrence in each transaction. FP-growth, Han *et al*. (2004) is a well-known algorithm that uses the FP-tree data structure storing crucial, quantitative information about frequent patterns and employs a divide-and-conquer approach to decompose the mining problem into a set of smaller problems. Tree projection, Agarwal *et al*. (2001) and Transaction Mapping, Mingjun and Sanguthevar (2006) are other algorithms to find frequent patterns with single *minsup*. However, using only a single *minsup* implicitly assumes that all items in the data are of the same nature or have similar frequencies in the data.

Corresponding Author: F. A. Hoque, Department of CSE, University of Information Technology Sciences (UITS), Dhaka, Bangladesh

This is, however, seldom the case in real life applications. In many applications, some items appear very frequently in the data, while others rarely appear. If the frequencies of items vary a great deal, we will encounter two problems:

- If *minsup* is set too high, those rules that involve rare items will not be found
- To find rules that involve both frequent and rare items, *minsup* has to be set very low

This may cause combinatorial explosion because those frequent items will be associated with one another in all possible ways and many of them are meaningless. This dilemma is called the rare item problem, Bing *et al*. (1999). To solve the problem, Bing *et al*. (1999) have extended the existing association rule model to allow the user to specify multiple minimum supports to reflect different natures or frequencies of items. Specifically, the user can specify a different Minimum Item Supports (MIS) for each item. Thus, different rules may need to satisfy different minimum supports depending on what items are in the rules. This new model enables us to achieve our objective of producing rare item rules without causing frequent items to generate too many meaningless rules. In the new model, the minimum support of a rule is expressed in terms of Minimum Item Supports (MIS) of the items that appear in the rule. That is, each item in the database can have a minimum item support specified by the user. By providing different MIS values for different items, the user effectively expresses different support requirements for different rules. Let MIS (i) denote the MIS value of item i. The minimum support of a rule R is the lowest MIS value among the items in the rule. That is, a rule R, a_1, a_2, …, a_k, a_{k+1}, …, a_r where a_j belongs to I, satisfies its minimum support if the rule's actual support in the data is greater than or equal to: min (MIS (a_1), MIS (a_2), …, MIS (a_r)) (Bing *et al*., 1999).

Example 1: Consider the following items in a database: Bread, shoes, clothes. The user-specified MIS values are as follows:

MIS (Bread) = 2%, MIS (shoes) = 0.1%, MIS (clothes) = 0.2%

The following rule doesn't satisfy its minimum support:

Clothes → Bread [sup = 0.15%, conf = 70%].

Because min (MIS (Bread), MIS (Clothes)) = 0.2%. The following rule satisfies its minimum support:

Clothes → shoes [sup = 0.15%, conf = 70%]
Because min (MIS (Clothes), MIS (shoes)) = 0.1%

Different approaches had been used to mine association rules with multiple minimum supports. Of these, the MSapriori algorithm, Bing *et al*. (1999) extends the Apriori algorithm, Agrawal and Srikant (1994), so that it can find frequent patterns with multiple MS thresholds. After the application of the MSapriori algorithm, Bing *et al*. (1999), all frequent itemsets are found but the supports of some subsets may be still unknown. Thus, if we intend to generate association rules, we need a post-processing phase to find the supports of all subsets of frequent itemsets. This procedure is time-consuming because we need to scan the database again and compute the supports of all subsets of frequent itemsets. To solve the both problems, a novel multiple item support tree (MIS-tree for short) structure is proposed by Ya-Han and Yen-Liang (2006) which extends the FP-tree structure and an efficient MIS-tree-based mining method, the CFP-growth algorithm is developed for mining the complete set of frequent patterns with multiple minimum supports, Ya-Han and Yen-Liang (2006). In case of multiple minimum supports, users cannot find applicable support value at once and always tune its support value constantly. To do this, every time when users change the items' minsup, they must rescan database and then execute the mining algorithm once again. It is very time-consuming and costly. Thus, it is attractive to consider the possibility of designing a maintenance algorithm for tuning minimum supports (MS for short). The problem will become even more serious for frequent pattern mining with multiple MS, because previously users only need to tune a single MS threshold but now they need to tune many MS thresholds. Thus, it is even more demanding to have a maintenance algorithm for MS tuning. But maintenance algorithm for tuning MS proposed by Ya-Han and Yen-Liang (2006) includes following three restrictions:

- MIN value will remain unchanged
- MS of an item is not allowed to be greater than MIN when it is smaller than MIN
- MS of an item is not allowed to be smaller than MIN when it is greater than MIN

These restrictions are added so that all the items in MIS tree must be kept the same after the tuning process. With this restriction users do not need to access the database again when minimum supports of each item are changed, because all the data needed to find frequent patterns are kept in the MIS tree. Again this algorithm will extensively perform better when the database is static or will not be updated very frequently. In case of incremental update of database some strong rules may become weak and some weak rules may become strong. That means an infrequent item may be frequent or a frequent item may be infrequent. But following the

restrictions, the MIS-tree algorithm, Ya-Han and Yen-Liang (2006) will not be able to recover an infrequent data item without rescanning the whole database and it is not obviously wise decision to rescan the whole database each time of updating the database. To overcome these limitations, we modify the MIS-tree maintenance method, Ya-Han and Yen-Liang (2006).

In this study, we develop new algorithms for complete and flexible minimum support tuning as well as the maintenance of the MIS-tree for incremental database. This will allow us to prove the effectiveness of our methods compared to existing method through experimental results. The experimental results on both synthetic and real data sets show the superiority of the proposed methods in terms of runtime required to maintain the tree after support tuning and incremental update.

METHODOLOGY

To develop our method, we modified MIS-tree maintenance algorithm for support tuning to mine frequent patterns with multiple minimum support proposed by Ya-Han and Yen-Liang (2006). We have also included MIS-tree maintenance method for incremental update. MIS-tree algorithm, Ya-Han and Yen-Liang (2006) has extended FP-tree algorithm, Han *et al*. (2004) to mine frequent residue pattern. The MIS-tree algorithm, Ya-Han and Yen-Liang (2006) can be summarized as follows:

Let DB = $\{T_1, T_2, \ldots, T_n\}$ be a transaction database, where T_j ($j \in [1. . .n]$) is a transaction containing a set of items in I. The support of an itemset A is the percentage of transactions containing A in DB. If itemset A's support is no less than MIS (A), then pattern A is a frequent pattern. Let MIN denote the smallest MIS value of all items

$$MIN = \min [MIS (a_1), MIS (a_2),..., MIS (a_m)]$$

Let the set of MIN_frequent items, F denote the set of those items with supports no less than MIN. i.e. F = {items | support (item) $\geq$ MIN}

At first, we have to build a MIS Tree from a Transaction Database DB that is given in Table 1 and the MIS value of each item is given in Table 2.

Definition 1(*MIS-tree*): A Multiple Item Support tree, Ya-Han and Yen-Liang (2006) is defined as follows:

- It consists of one root labeled as "null", a set of item prefix sub trees as the children of the root, and a MIN_frequent item header table which contains all items in F.

Table 1: Transaction database DB (sorted according to count in descending order

Tid	Items bought	Items (ordered)
100	d, c, a, f	a, c, d, f
200	g, c, a, f, e	a, c, e ,f, g
300	b, a, c, f, h	a, b, c, f, h
400	g, b, f	b, f, g
500	b, c	b, c

Table 2: MIS value of each item

Item	a	b	c	d	e	f	g	h
MIS	4	4	4	3	3	2	2	2

Table 3: deleted_node_info table

Item	Count	Prefix path	Suffix path
d	1	{a,c}	{f}
e	1	{a,c}	{f,g}
h	1	{a,b,c,f}	Null

Table 4: Item_count table

Item (X)	Support count	MIS value
a	3	4(40%)
b	3	4 (40%)
c	4	4(40%)
d	1	3 (30%)
e	1	3 (30%)
f	4	2 (20%)
g	2	2 (20%)
h	1	2 (20%)

- Each node in the item prefix sub tree consists of three fields: *item-name*, *count* and *nodelink*, Where item-name registers which item this node presents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the MIS-tree carrying the same item-name, or null if there is none.
- Each entry in the MIN_frequent item header table consists of three fields: item-name, item's minsup MIS(a_i) and head of node-link which points to the first node in the MIS-tree carrying the item-name.
- All the items in the table are sorted in decreasing order in terms of their MIS values.

According to Definition 1, the MIS-tree constructed from given DB is shown in Fig. 1.From the tree, we will get the count of each item as (a:3, b:3, c:4, d:1, e:1, f:4, g:2, h:1) and store it in a table name item_count. We only need to retain those items with supports no less than MIN=2. So, we remove the nodes d, e, h and store them in deleted_node_info table. The tables are shown in Table 3 and 4. After deleting the nodes, the Complete MIS-tree is given in Fig. 2.

The CFP-growth algorithm, Ya-Han and Yen-Liang (2006) is proposed for mining the complete set of frequent patterns for the MIS-tree in Fig. 2. Now we have to build a conditional pattern base and conditional MIS-tree for each item. Whether an item is frequent in the item's conditional MIS-tree is checked by following the node-link of each item, summing up the counts along the link

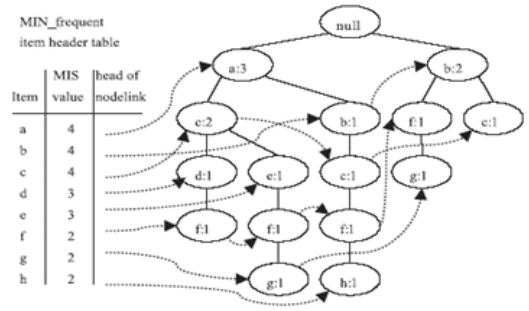


Fig. 1: The incompact MIS-tree, Ya-Han and Yen Liang (2006)

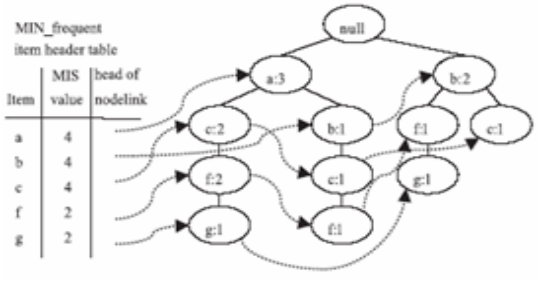


Fig. 2: The Complete MIS-tree, Ya-Han and Yen-Liang (2006)

Table 5 : All conditional patterns

Item	MIS	Conditional pattern base
g	2	{(a:1,c:1,f:1),(b:1,f:1)}
f	2	{(a:1,c:1),(a:1,b:1,c:1),(b:1)}
c	4	{(a:2),(a:1,b:1),(b:1)}
b	4	{(a:1)}
a	4	∅

Table 6: All conditional frequent patterns

Item	Conditional pattern	Frequent pattern
g	ag,bg,cg,fg,acg,afg,cfg,bfg,acfg	fg
f	af,bf,cf,abf,acf,bcf,abcf	af,bf,cf,acf
c	ac,bc,abc	∅
b	ab	∅

and seeing whether it exceeds the MIS value of that item. The CFP-growth method, Ya-Han and Yen-Liang (2006) for that item will not be terminated until all that item's conditional pattern bases contain no items. Repeatedly doing this for all items in the header table, we can get the whole conditional pattern base in Table 5 and all conditional patterns in Table 6.

Tuning MIS values without restrictions: In this study, we modified the MIS-tree maintenance method, Ya-Han and Yen-Liang (2006) by tuning MIS values violating the restrictions that were used in existing maintenance method, Ya-Han and Yen-Liang (2006). Note that in the

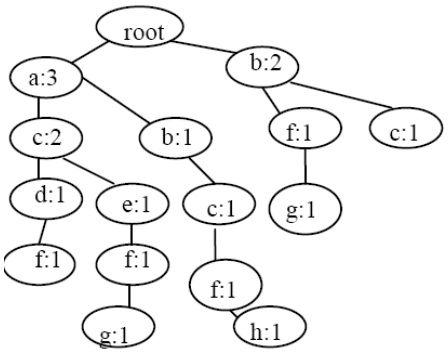


Fig 3: MIS-tree after inserting node d, e, h

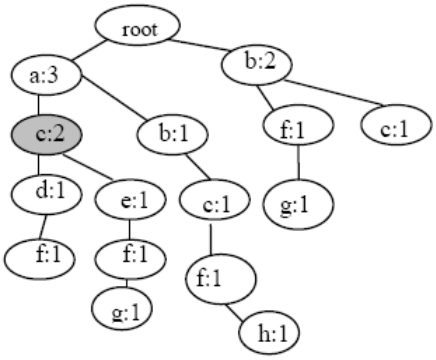
Table 7: The old and new MIS value of each item

Item	Order (old)	Support	MIS (old)	MIS (New)	Order (New)
a	1	3	4	5	2
b	2	3	4	3	4
c	3	4	4	6	1
d	4	1	3	2	5
e	5	1	3	1	7
f	6	4	2	4	3
g	7	2	2	2	6
h	9	1	2	1	8

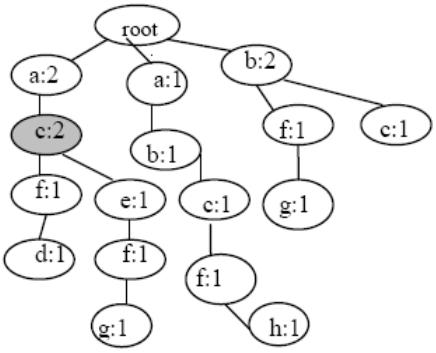
support tuning process of MIS tree, YaHan and Yen-Liang (2006), it was restricted that all the data items should be same in the MIS tree after tuning. Because it was not possible to recover a deleted data in the MIS tree without rescanning the database again. But in our proposed algorithm, we do not need to add any kind of restriction. Because we can recover deleted node if needed from the MIS tree after support tuning by traversing the path according to the prefix and suffix value of data item that is stored in the deleted_node_info table. The maintenance process can be stated as follows:

- After the user tunes the item's supports the order of items will be changed since the items are sorted in the MIS-tree according to their MIS value. At the same time, we have to determine whether an item should be recovered which is already deleted from the MIS-tree. For example, the new MIS value and order of each item is given in Table 7.

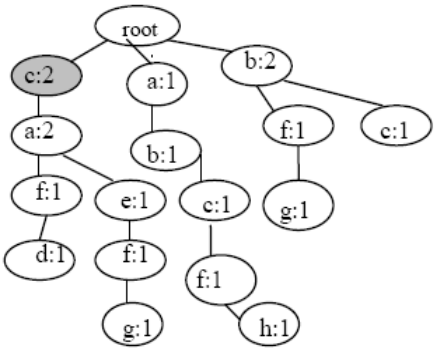
In the previous example, MIN value was 2 that mean 40% of all transactions. From the new MIS values, we find that MIN is 20% of all transactions or 1. According to Table 7, the MIN_frequent item set, F = {(c: 4), (a: 3), (f: 4), (b: 3), (d: 1), (g: 2), (e: 1), (h: 1)}. SO, after support tuning, data item d, e and h must be recovered and inserted into the tree by using the information in the deleted_node_info table. For example, prefix of d is {(a,



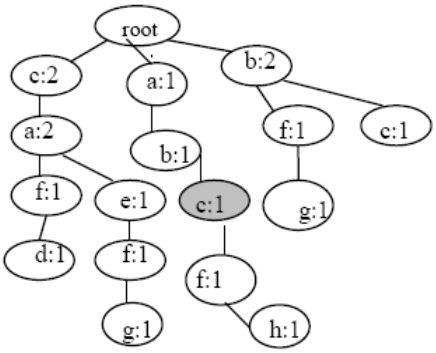
(a) Move up c



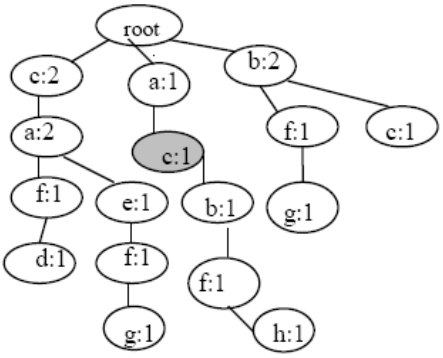
(b) Move up c



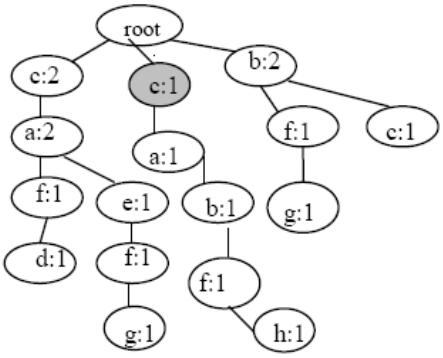
(c) Move up c



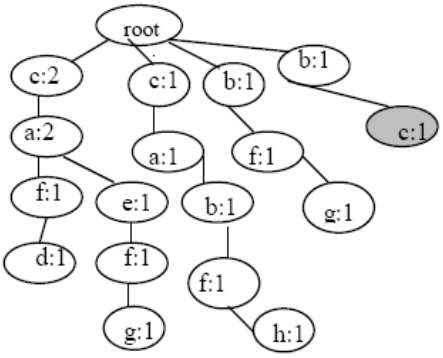
(d) Move up c



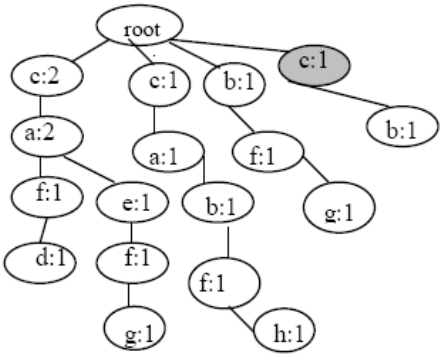
(e) Move up c



(f) Move up c



(g) Move up c



(h) Move up c

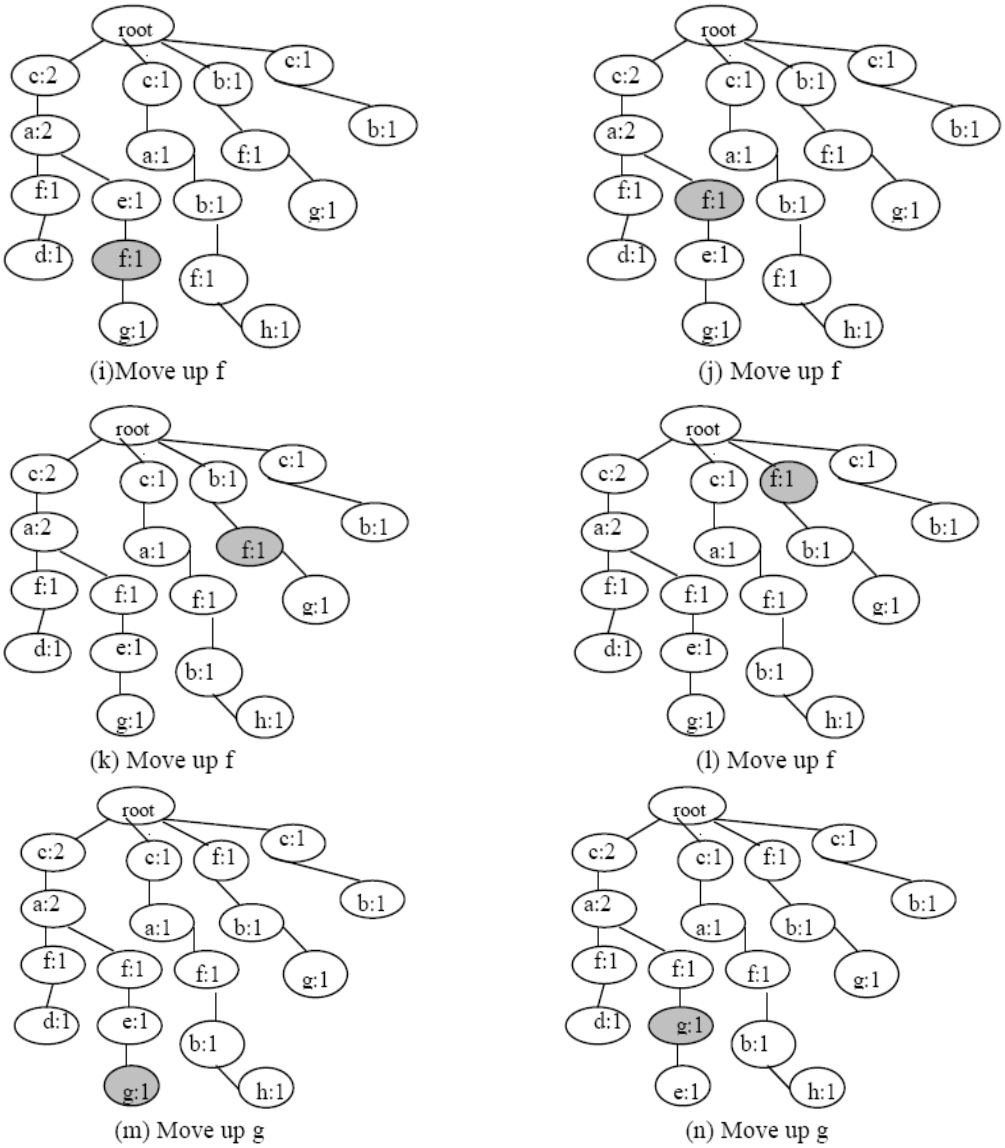
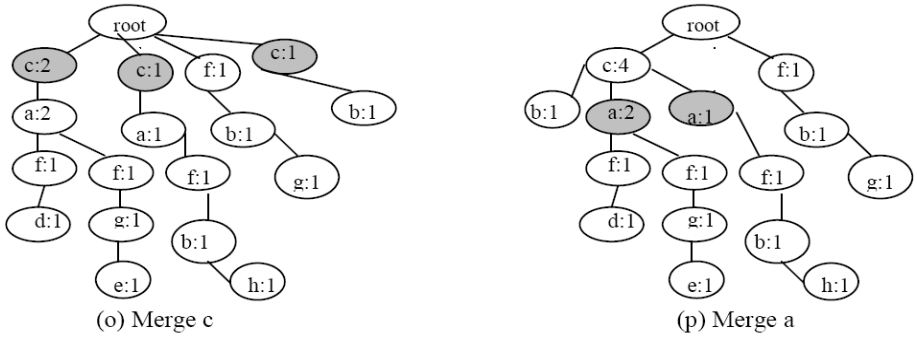


Fig. 4: Move_up method in MS tuning process



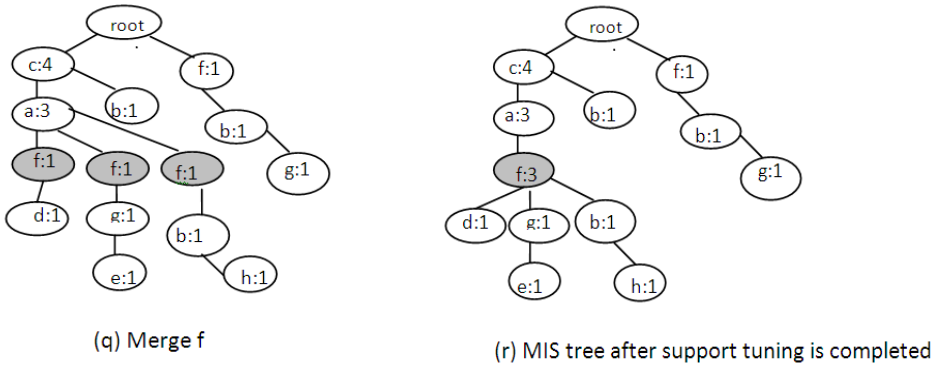


Fig. 5: MIS_merge method in MIS tuning process

c)} and suffix is {f}. We can insert d in the MIS tree by traversing the path {(a, c)} and obviously d will be the previous node of {f} in that path. The MIS-tree after inserting node d, e, h is shown in Fig. 3. The entries of the recovered node items have to be deleted from the deleted_node_info table.

- Note that after the user tunes the items' supports, we will find the new order of data items. So, we need to determine which items should be moved up so that items in the MIS-tree can match the new item order.

- Old order: a (4), b (4), c (4), d (3), e (3), f (2), g (2), h(2)
- New order: c (6), a (5), f (4), b (3), d (2), g (2), e (1), h (1)

If we find an item whose new order is smaller than its preceding items, then these items should be moved up by using Move_up method, Ya-Han and Yen-Liang (2006). Continuously doing this we can find all items that should be moved up. The steps can be described as follows:

- An item may occur several times in the tree, where all of them are linked together through its node-link, so we can visit all the nodes carrying the same item-name.
- Let the node we are currently visiting be node i, and let node f be the parent node of node i and node gf be the grandparent of node i. If the new order of node f is smaller than that of node i, then the work is over. On the contrary, node i should be moved up above node f.
- Here, if f.support = i.support, then we can directly swap these two nodes without any other modifications.
- If f.support > i.support, then we split node f into two nodes, say node f_1 and node f_2, where f_1.support = i.support and f_2.support = f.support - i.support. As for node f_1, we make i as its only child node and then we swap node f_1 and node i; as for node f_2, we make all child nodes of node f except node i as its child nodes; as for node gf, we make f_1 and f_2 as his children. This ascending process will run repeatedly until the new order of parent node is smaller than the currently visited node or until the parent node is the root. After moving up these nodes, the nodes in MIS-tree may contain child nodes carrying the same item name. For the compactness, we use MIS_merge method, Ya-Han and Yen-Liang (2006) to merge those nodes. The move-up operations and MIS_merge method is shown in Fig. 4 and 5.

change_deleted_node_info Algorithm:
Input: deleted_node_info and (updated) minimum support threshold of each item MIS(a_i)
Output: deleted_node_info (updated according to new order).
Method:
for each item Y in the deleted_node_info table do the following:
 Concatenate the prefix path, item Y, suffix path and sort the items except NULL according to new MIS value in descending order.
 if there are items before Y, then
  for each item X before Y do the following:
  store X in the prefix path of Y
 else store NULL in the prefix path of Y
 if there are items after Y, then
  for each item Z after Y do the following:
  store Z in the suffix path of Y
 else store NULL in the suffix path of Y

Fig. 6: change_deleted_node_info algorithm

After arranging the MIS tree according to new order, the deleted_node_info table has to be changed. We have given the algorithm change_deleted_node_info in Fig. 6. The whole approach can be summarized in Fig. 7.

Maintenance of MIS-tree after incremental update of database: In this study, we also developed an incremental

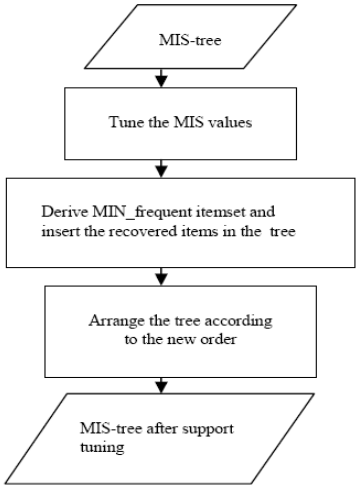


Fig. 7: Flow chart of the tree maintenance method after suuport tuning

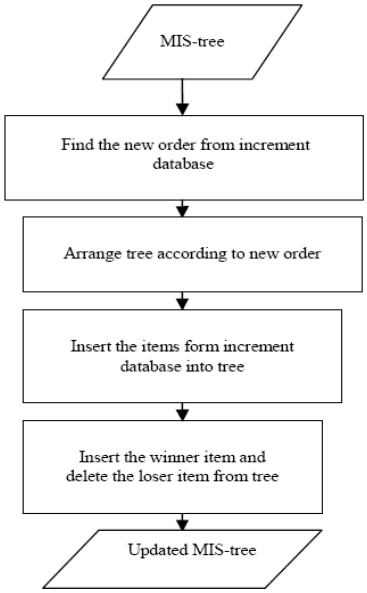


Fig. 8: Flow chart of the tree maintenance method after incremental update

Table 8: Transactional database (db)

Tid	Item	Item (ordered)
100	d,c,a,f	a,c,d,f
200	g,c,a,f,e	a,c,e,f,g
300	b,a,c,f,h	a,b,c,f,h
400	g,b,f	b,f,g
500	b,c	b,c

updating technique for maintenance of the MIS-tree. Database is subject to update in practice. The goal of this approach is to solve the efficient update problem of association rules after a nontrivial number of new records have been added to a database. There are several

Table 9: Incremental Database (db)

Tid	Item	Item (ordered)
600	b,d	b,d
700	b,c,d	c,b,d
800	a,c,d,f,g	c,a,f,d,g
900	a,c	c,a
1000	b,c,f,h	c,b,f,h

Table 10: MIS value of each item

Item	a	b	c	d	e	f	g	h
MIS	7	6	8	5	4	6	5	4

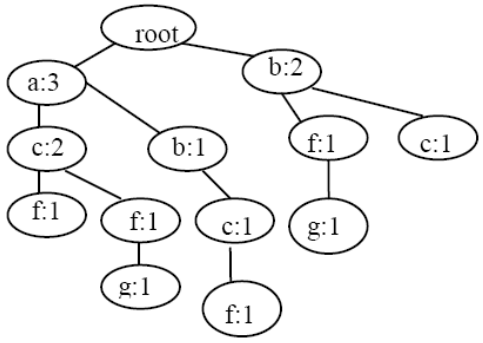


Fig. 9: The Complete MIS-tree from DB

important characteristics of update problem. The update problem can be reduced to find new set of large item sets. After that, the new association rules can be computed from the new large item sets. Generally speaking, an old large item set has the potential to become small in the updated database. Similarly, an old small item set could become large in the new database. In order to find the new large item sets, all the records in the database, including those from the original database, have to be checked. One possible approach to the update problem is to re-run the association rule mining on whole updated database. This approach, though simple, has some obvious disadvantages. All the computation done initially at finding frequent item sets are wasted and all frequent item sets have to be computed again from scratch. That mean we have to rescan the whole updated database again in each time we want to add some new records in the original database. This is obviously a great disadvantage. The primary challenge of devising an effective maintenance algorithm for association rules is how to reuse the original frequent item sets and avoid the rescanning of original database DB. The whole approach is summarized in Fig. 8. We focus on the maintenance of MIS-tree, so that every time an incremental database is added to the original database, we can keep this tree in correct status without rescanning the original database DB. Let F be the set of MIN_frequent items in the original database DB and D be the number of transactions in DB. MIS (a_i) is the minimum item support of item a_i. The

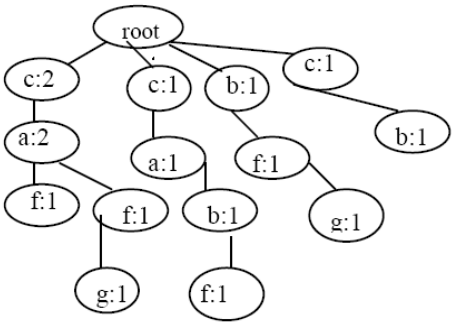


Fig 10: MIS-tree according to new order

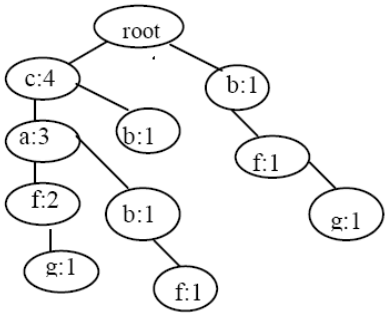


Fig. 11: MIS-tree after merging c, a & f

Table 11: deleted_node_info table

Item	count	Prefix path	Suffix path
d	1	{c, a, f}	NULL
e	1	{c,a,f,g}	NULL
h	1	{c,a,b,f}	NULL

value MIN in the database denote as the minimum of the MIS value of each item. MIN = min (MIS (a_1), MIS (a_2),….., MIS (a_n)), where a_1, a_2,….,a_i the items in DB. F is the set which contains the minimum frequent items whose support is no less than MIN value. After some update activities, an increment db of new transactions is added to the original database DB. Let d be the number of transactions in db. If any new items are not added to the database and minimum support value of each item remains unchanged then MIN value will be unchanged in the updated database. With respect to the same MIN value of the database, an item set X is the minimum frequent item in the updated database (DB ∪ db), if the support of X in (DB ∪ db) is no less than MIN i.e., X. support ≥ MIN * (D + d).

The following notations are used to define different terms:

F : MIN_frequent item set of DB
F' : MIN_frequent item set of (DB ∪ db)
$X.support_D$: support value of item set X in DB
$X.support_d$: support value of item set X in db
$X.support_{UD}$: support value of item set X in (DB ∪ db)

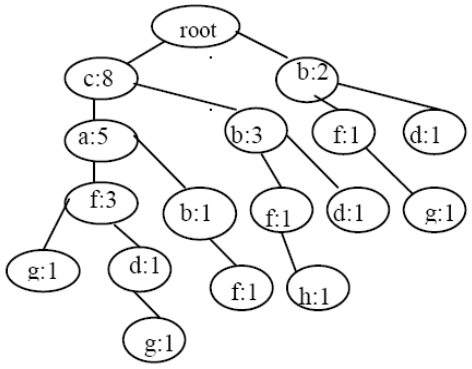


Fig. 12: The updated tree after inserting the incremented database

Table 12: Updated item_count (Support and MIS of each item in DB È db)

Item(X)	$X.support_{UD}$	MIS value
a	5	8 (80%)
b	6	8 (80%)
c	8	8 (80%)
d	4	6 (60%)
e	1	6 (60%)
f	6	4 (40%)
g	3	4 (40%)
h	2	4 (40%)

Example 2: Say, we have original database DB, incremented database db from Table 8 and 9. The MIS value of each item is shown in Table 10 and the MIS-tree constructed from DB is shown in Fig. 9. The proposed approach that maintain MIS-tree algorithm after incremental update can be summarized as follows:

First scan of database db and find the new order. Since we have found the new order of data items, so before inserting the data items from db we have to change the old MIS-tree according to new order

New order: c (8), a (7), b (6), f (6), d (5), g (5), h (4), e (4)
Old order: a (4), b (4), c (4), d (3), e (3), f (2), g (2), h (2)

The new MIS-tree and the compact MIS-tree is shown in Fig. 10 and 11. After arranging the MIS tree according to new order, the deleted_node_info table also has to be changed using the algorithm change_deleted_node_info which is given in Fig. 6. After applying the algorithm, deleted_node_info table will be changed and it is shown in Table 11. Now the MIS-tree is arranged in the correct order of updated database DB∪db. The MIS-tree after inserting the transaction from db is shown in Fig. 12. Then we have to derive MIN_frequent item set. The following properties are useful in the derivation of the MIN_frequent item set for the updated database.

Table 13: deleted_node info table

Item	Count	Prefix path	Suffix path
e	1	{c,a,f,g}	Null
h	2	{(c,a,b,f), (c,b,f)}	Null
g	3	{(c,a,f),(c,a,f,d),(b,f)}	Null

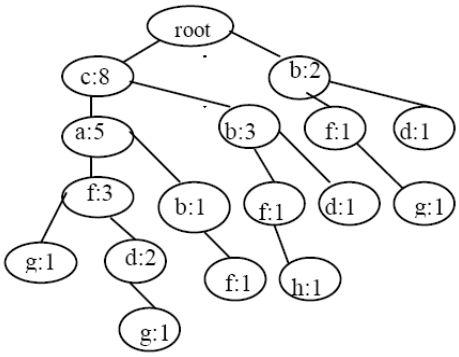


Fig. 13: MIS-tree after recovering data item d

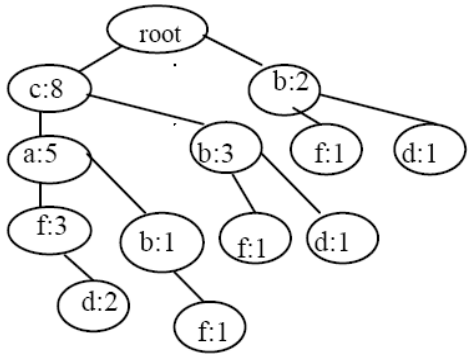


Fig. 14: MIS-tree after deleting g, h

Property 1: An itemset X in the original MIN_frequent item set F can be a loser in the updated database (DB ∪ db) (i.e., not in the F′ if and only if X. support UD<MIN* (D + d).

Property 2: An item set X not in original MIN_frequent item set F can become a winner in the updated database (DB union db) (i.e., being included in the F′) if and only if $X.support_{UD} \geq MIN * (D + d)$. From the tree, we can get the support count and MIS value of each item (for the updated database) and the updated item_count is given in Table 12.

Here MIN * (D + d) = (40 %) * 10 = 4. Hence, from Table 12 and according to definition of minimum frequent item set, MIN_frequent item set for the updated database should be:

$$F' = \{(c:8),(b:6),(f:6),(a:5),(d:4)\}$$

Table 14: Parameter settings for synthetic data generation

|D|	Number of transactions in database DB
d	Number of transactions in the increment db
|T|	Mean size of the transactions
|I|	Mean size of the maximal potentially large item sets
|L	Number of potentially large itemsets
N	Number of items

From property 1 and property 2, we find that (d: 4) is the winner data item and (g: 3) and (h:2) are the loser data items. Now we have deleted_node_info table which stores the updated information about deleted node item from the MIS-tree with their prefix list and suffix list. Here, we need to recover data item (d: 1) in the original MIS-tree structure. According to the table, the prefix path of d is {(c, a, f)} and there is no item in the suffix list of d. We can insert d in the MIS-tree by traversing the path {(c, a, f)} and obviously d will be the last node of that path. As soon as we recover the data item "d" we will delete the entry of the node item name "d" from the deleted_node prefix table. After that, we have to delete the loser items from the tree and store them in the deleted_node_info table. After recovering (d: 1) and deleting (g: 3), (h: 2) the tree is shown in Fig. 13 and 14. The updated deleted_node_info table is shown in Table 13.

EXPERIMENTAL RESULTS

For the purpose of analyzing and demonstrating the efficiency and effectiveness of the proposed methods, we conducted some experiments at the computer laboratory of the Department of Computer Science and Engineering, University of Dhaka in 2010. We have investigated the performance of the maintenance algorithm for updating the MIS- tree when tuning MS without following the restrictions included in the existing approach. In addition, we also focused the superiority of the proposed approach for incremental update of data base. All the experiments are performed on a Pentium IV 2.0GHz CPU with 4 GB of memory and running Windows XP. As data sets are concerned, we used both synthetic and real life datasets. The synthetic data is generated by using the IBM data generator, Agrawal and Srikant (1994) which is widely used for evaluating association rule mining algorithms. The parameters of the experiments are shown in Table 14. Besides, we have used BMS-POS, as our real-life dataset which was used in the KDD-Cup 2000 competition, Zheng *et al*. (2001). The BMS-POS dataset contains point of-sale data from a large electronics retailer over several years. In our experiments, we have used the method proposed by Bing Liu *et al*. (1999) to assign MIS values to items. In this method, the actual frequencies of the items in the DB used as the basis for MIS assignments. The formula can be stated as follows:

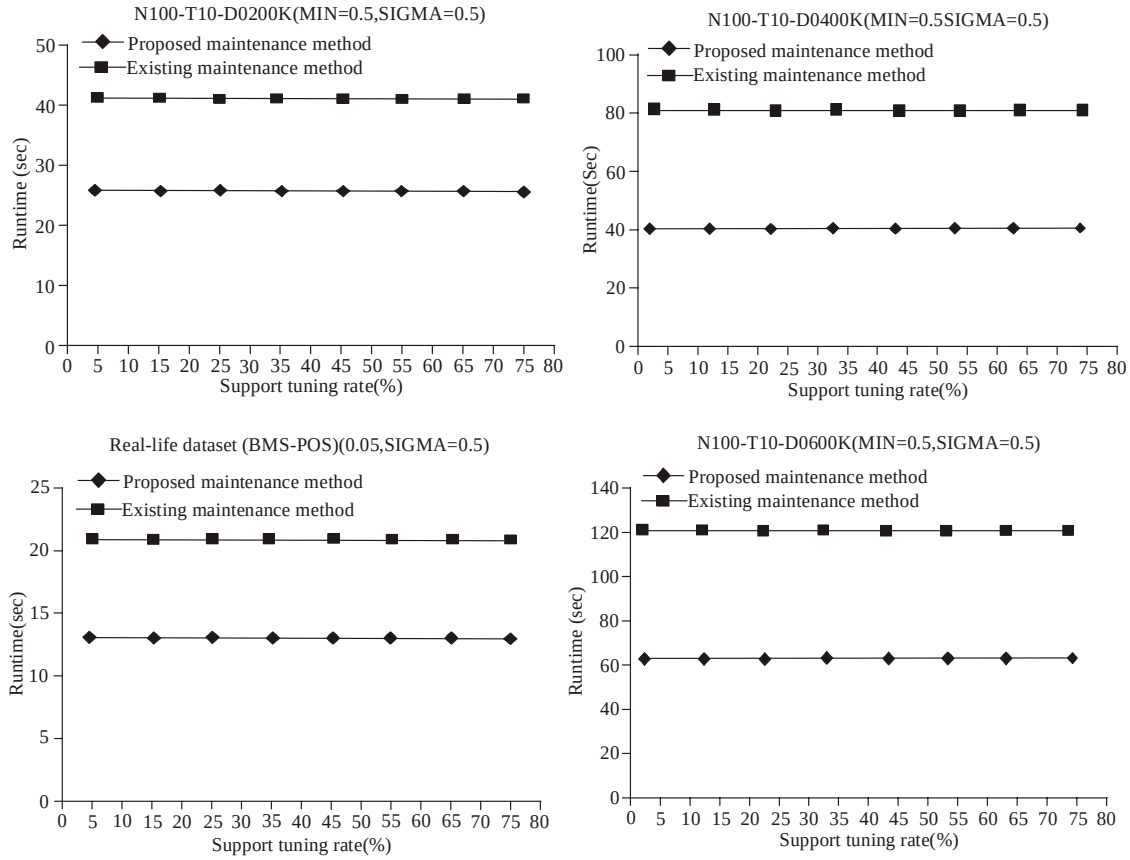


Fig. 15: Experimental results with MS tuning process violating the restrictions

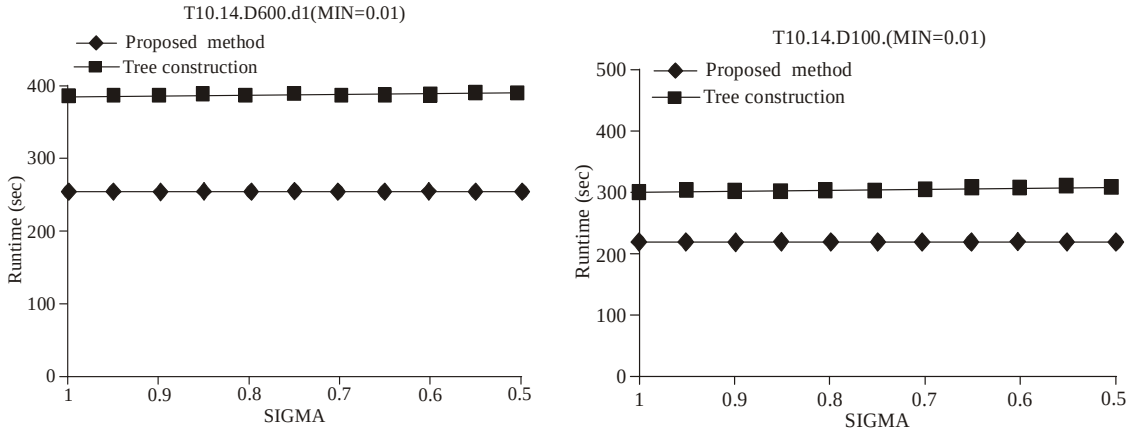


Fig. 16: Experimental results with incremental update of database

$$MIS(a_i) = \begin{cases} M(a_i) & M(a_i) > Min \\ Min & Otherwise \end{cases}$$

$$M(a_i) = \sigma \times f(a_i)$$

M (a_i) is the actual frequency of item a_i in the DB. MIN denotes the smallest MIS value of all items. σ ($0 \leq \sigma \leq 1$) is a parameter that controls how the MIS value for items should be related to their frequencies. If σ = 0, we have only one MS, which is the same as the traditional association rule mining. To test the performance of our

proposed tree maintenance algorithm when tuning MS, we compare it with the existing tree maintenance algorithm. The MIN value and σ are set to 0.05 and 0.5%, respectively. We randomly choose items in F with probability varied from 5 to 80%. The new MIS value of each chosen item will be set by randomly selecting a value from the range [old× (1-0.05), old× (1+0.05)],where old denotes the original MIS value. The results are shown in Fig. 15. All experiments show that the saving is very significant in practice. It demonstrates that the proposed method possesses better performance than existing one.

Experiments for incremental update of database: To test the performance of our tree maintenance algorithm for incremental update, we compare it with the new construction of the MIS-tree. The way we create our increment is a straight forward extension of the technique used to synthesize the database. A database of size (D + d) is first generated and then the first D transactions are stored in the database DB and the remaining d transactions stored in the increment db. Since all the transactions are generated from the same statistical pattern, it models very well real life updates. The results in Figure 16 show that in average using our MIS-tree maintenance method is able to save much runtime of reconstructing the MIS-tree.

DISCUSSION AND CONCLUSION

In this research, we have developed algorithms for maintenance of the tree after support tuning and incremental update of database without rescanning database. We have implemented our maintenance algorithm for MS tuning violating the restrictions used in the existing maintenance method, Ya-Han and Yen-Liang (2006). The results indicate that in all cases our proposed approach performs better than the existing approach. Besides, we also examined our maintenance algorithm for incremental update. Experimental results show that our method is faster than the method of reconstructing the MIS tree. In short, the research has two main results. First, we solve the problem occurred in the MIS tree algorithm, Ya-Han and Yen-Liang (2006) that it cannot tune supports of each item with the full flexibility without rescanning the database again. Second, we develop an efficient maintenance algorithm after incremental update of database which also performs better than the reconstruction of MIS tree algorithm after update of database. There are number of directions where improvement can take place in future studies. Currently, we only consider the problem of frequent pattern generation. Since the ultimate goal of association rule mining problem is to generate rules from frequent patterns, this research can be extended to find the association rules with multiple minimum supports. We hope to improve our method to mine other kinds of knowledge under the constraint of multiple MS rather than setting a single threshold for all items. Because many kinds of that knowledge can be discovered from database that contain multiple items, all these types of knowledge can be extended naturally by setting different support thresholds for different items

ACKNOWLEDGMENT

This research was supported by the Department of Computer Science and Engineering, University of Dhaka. We would like to thank the department for providing us Lab facilities and research journals to complete the research.

REFERENCES

Agarwal, R.C., C.C. Agarwal and V.V.V. Prasad, 2001. A tree projection algorithm for generation of frequent item sets. J. Parall. Dist. Comp., 61(3): 350-371.

Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. Proceedings of ACM SIGMOD International Conference on Management of Data, Washington DC, USA, pp: 207-216.

Agrawal, R. and R. Srikant, 1994. Fast algorithms for mining association rules. Proceedings of the 20th Very Large Data Bases Conference (VLDB'94), Santiago Chile, pp: 487-499.

Bing, L., H. Wynne, and M. Yming, 1999. Mining association rules with multiple minimum supports. ACM-SIGKDD International Conference on knowledge Discovery and Data Mining, (KDD-99), San Diego, CA, USA.

Han,J., J.Pei,Y.Yinand RMao,2004. Miningfrequentpatt ernswithoutcan didategeneration: Afrequentpatternt reeapproach. Data. Min. Knowl. Disc., 8(1): 5387.

Mingjun, S. and R. Sanguthevar, 2006. A transaction mapping algorithm for frequent itemsets mining. IEEE T. Knowl. Data Eng., 18(4): 472-481.

Ya-Han, H. and C. Yen-Liang, 2006. Mining association rules with multiple minimum supports: A new mining algorithm and a support tuning mechanism. Decis. Support Syst., 42: 1-24.

Zheng, Z., R. Kohavi, L. Mason, 2001. Real world performance of association rule algorithms. Proceedings of the 7th ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA USA, pp: 401-406.