International Journal of Engineering Applied Sciences and Technology, 2017
Vol. 2, Issue 5, ISSN No. 2455-2143, Pages 101-113
Published Online March-April 2017 in IJEAST (http://www.ijeast.com)

NEWGENMAX: A NOVEL ALGORITHM FOR MINING MAXIMAL FREQUENT ITEMSETS USING THE CONCEPT OF SUBSET CHECKING

Shalini Bhaskar Bajaj
Department of Computer Science and Engineering
Amity University Haryana

Abstract— **Frequent itemset identification from a given dataset is an important research area. For finding frequent itemsets use of closed/maximal frequent itemsets is proposed in the literature as the memory required to store closed/maximal frequent itemsets is less. In this paper an effort has been made for finding maximal frequent itemsets in place of frequent itemsets. The proposed algorithm NewGenMax finds maximal frequent itemsets in order to save memory space in a reduced time from the list of local maximal frequent itemset list. NewGenMax algorithm is compared to GenMax algorithm in terms of the number of iterations required and the execution time. In order to verify the supremacy of the proposed algorithm NewGenMax over the existing algorithm GenMax the experiments have been performed on both synthetic and real datasets. The results obtained are encouraging.**

Keywords—**prefix tree, maximal frequent itemsets, transactions, support**

I. INTRODUCTION

Mining is the process of extracting valid, previously unknown, comprehensible and actionable information from large databases and using it to make crucial business decision. Association rule mining is a type of mining used to identify certain kinds of association among the items in the database. Frequent itemset mining [2, 3, 6, 7, 8] and rule generation [16] are the two subtasks of association rule mining. Mining of frequent itemsets is a fundamental and essential need in many data mining applications such as the discovery of association rules, strong rules, correlations, multidimensional patterns, and many other important discovery tasks, etc. Many applications like inductive databases and query expansion require fast implementations of frequent itemset mining. Most of the approaches for frequent itemset mining enumerate candidate itemsets, determine their support and prune candidates that fail to reach the user-specified minimum support. These approaches often results in generating a large number of frequent itemsets that takes more memory space. Candidate generation-and-test methodology or the Apriori technique is the base technique of frequent itemset mining algorithms.

In order to reduce the memory space requirement, maximal frequent itemsets are identified. Maximal frequent itemsets are those frequent itemsets which do not have any subset in the frequent itemset list and they store information about all frequent itemsets in a precise manner. Mining of frequent patterns is a basic problem in data mining. Most of the frequent itemset mining algorithms work by checking the superset for each itemset. This takes more memory space and is time consuming. The objective of the paper is to propose an algorithm that can save memory space by reducing the number of iterations identified while generating maximal frequent patterns from the list of local maximal frequent itemsets.

II. BASIC DEFINITIONS AND RELATED EXISTING WORK

Before discussing the proposed algorithm it is important to get familiarized with the basic terminology:

Association Rule Mining: Association Rule Mining is a popular and well researched method for discovering interesting relations between variables in large databases. The task of mining association rules consists of two steps which involves finding the set of all frequent itemsets followed by testing and generating all high confidence rules among these itemsets.

Itemset, I: An itemset is the set of m items {i1, i2… im}, where m=0, 1, 2, 3, …

Closed Itemset: An itemset is said to be closed if there does not exist any superset that has the same support.

Database, D: It denotes a database of transactions where each transaction has a unique identifier (tid) and contains a set of items (itemsets).

Tidset, T: The set of all transaction identifiers, tids is denoted by T = {t1, t2, …, tm}.

The set t (X) $\subseteq$ T, consists of all the transaction identifiers which contain X as a subset is called the tidset of X.

K-itemset: An itemset with k items is called a k-itemset.

Example: {A, C, D, T, W} represents a list of 1-itemset or F1 and {AC, CD, AD} denotes a list of a 2-itemset or F2.

Support, $\sigma(X)$: The number of transactions in which an itemset X occurs as a subset is termed as support. Thus, $\sigma(X) = |t(X)|$

Minimum Support, min_sup: Minimum support is the predefined threshold support value by the user.

Frequent Itemset, FI: An itemset X is frequent if its support value is more than or equal to the min_sup value, i.e. $\sigma(X) >= min\_sup$. Therefore, a frequent itemset is the one that occurs in at least a user-specified percentage of the database. Here, Fk denotes the set of frequent k-itemsets.

Local Maximal Frequent Itemset, LMFI: This list of local maximal frequent itemsets contains those maximal frequent itemsets that can potentially be the superset of candidates that are to be generated from the itemset.

Maximal Frequent Itemset, MFI: A frequent itemset is said to be maximal, if it is not a subset of any other frequent itemset, $|MFI| << |FI|$. These are those frequent itemsets which do not have a frequent superset in the LMFI list.

Frequent itemset mining is the most researched field of frequent pattern mining. Many algorithms use frequent itemset to identify the maximal frequent itemset. The original problem was to discover association rules, where the main step was to find maximum frequently occurring itemsets. Among all the frequent itemset mining algorithms, the majority of them claims to be efficient and follow the anti-monotone property, i.e. if a pattern is found to be frequent then all of its non-empty subsets will be frequent. In other words, if a pattern or itemset is not frequent, then none of its supersets can be frequent.

2.1 Overview of Existing Algorithms

The candidate generation-and-test methodology, called the Apriori [2] technique was the first technique to compute frequent patterns based on the anti-monotone property. MaxMiner [3] employs a breadth-first traversal of the search space and it reduces database scanning by employing a look ahead pruning strategy. FP-growth [10] uses an extended FP-tree [12] structure to store the database in compressed form. DepthProject [1] finds large itemsets by using depth first search on a lexicographic tree of itemsets. Mafia [5] uses three pruning strategies to remove non-maximal sets. Prefix-tree [13] or Trie [4] structure, known as an FP-tree is used for storing compressed information about frequent itemsets and implemented to mine frequent itemsets. DCI-Closed [11] proposes a general technique for promptly detecting and discarding duplicate closed itemsets, without the need of keeping in the main memory the whole set of closed patterns. GenMax [9, 15] uses a new format called diffset for fast frequency testing and progressive focusing for maximality checking.

Apriori was proposed by Agrawal et al. in the year 1993. Many of the proposed itemset mining algorithms are variant of Apriori [14] which employs a bottom-up, breadth-first search that enumerates every single frequent itemset. In many applications especially in dense datasets with long frequent patterns enumerating all possible $2m-2$ subsets of a m-length pattern is computationally unfeasible.

In the year 1998, Bayardo proposed MaxMiner [3] that employs a breadth-first traversal of the search space for finding the maximal frequent itemsets. It quickly narrows the search by using efficient pruning techniques. It also reduces the database scanning by employing a look-ahead pruning strategy, i.e. if a node with all its extensions is determined to be frequent then there is no need to further process that node. It employs item (re)ordering heuristic to increase the effectiveness of superset-frequency pruning.

Han et al. proposed tree based algorithm named FP-Growth [10] in the year 2000. The FP-tree structure is a compressed representation of all the transactions in the database. It uses a recursive divide-and-conquer and database projection approach to mine long patterns. Since it enumerates all frequent patterns it is impractical when pattern length is long. The FP-growth uses this FP-Tree as the basic data structure for a compact representation of all relevant frequency information of a database and thus removes the infrequent items. Every branch of the FP-tree represents a frequent itemset, and the nodes along the branches are stored in decreasing order of frequency of the corresponding items, with leaves representing the least frequent items. FP-growth identifies the support of each and every item in the transaction and therefore prunes the infrequent items.

In the same year Agarwal et al. proposed DepthProject algorithm [1] that finds long itemsets using a depth first search of a lexicographic tree of item-sets, and uses a counting method based on transaction projections along its branches. This projection is equivalent to a horizontal version of the tidsets at a given node in the search tree. DepthProject also uses the look-ahead pruning method with item reordering. It returns a superset of the MFI and would require post-pruning to eliminate non-maximal patterns.

In the year 2001, Burdick proposed Mafia algorithm [5] that uses three pruning strategies to remove non-maximal sets. The first strategy is the look-ahead pruning which was earlier used in MaxMiner. The second is to check if a new set is subsumed by an existing maximal set. The last strategy checks if $t(X) \subseteq t(Y)$ where X and Y are the subsets of the existing maximal set. If so, X is considered together with Y for the extension. Mafia uses vertical bit-vector data format and compression of bitmaps to improve the overall performance. Mafia mines a superset of the FI, and requires a post pruning step to eliminate non-maximal patterns.

A fast and memory efficient algorithm DCI-Closed [11] was proposed by Lucchese et al. in the year 2004 to mine frequent closed itemsets. The paper proposes a general technique for promptly detecting and discarding duplicate closed itemsets, without the need of keeping in the main memory the whole set of closed patterns. To remove duplicity, a particular visit of the lattice of frequent sets is used to identify unique generators of each equivalence class. This algorithm finds generators and computes their closure. As soon as a generator is found, its closure is computed and new generators are built as supersets of the closed itemset discovered so far. For any closed itemset Y, it is possible to find a sequence of order preserving generators in order to climb a sequence of closure itemsets and arrive at Y.

Surprising results of trie-based FIM algorithms [4] proposed by Ferenc Bodon et al. in the year 2004 were published. Prefix Tree (Trie) is a popular data structure in FIM algorithms which is an ordered tree data structure that is used to store an array or strings over an alphabet and efficiently retrieve words of a dictionary. It is memory-efficient and allows fast construction and information retrieval. Many trie-related techniques can be applied in FIM algorithms to improve efficiency for fast management. The itemset that is obtained by removing infrequent items from T is known as the filtered transaction of T. It is useless to store the same filtered transactions multiple times. Instead store them once and employ counters which store the multiplicities. This way the memory is saved and run-time can be significantly improved. The size of the FP-tree that stores filtered transactions is declared to be "substantially smaller than the size of database". A trie thus stores the same prefixes only once.

In 2005, GenMax algorithm [9] was proposed by Zali et al. It utilizes a backtracking search for efficiently enumerating all maximal patterns. It uses a number of optimizations to quickly prune away a large portion of the subset search space. It uses a novel progressive focusing technique to eliminate non-maximal itemsets, and uses the diffset propagation for fast frequency checking. It first describes the backtracking paradigm in the context of enumerating all frequent patterns, and then subsequently modifies this procedure to enumerate the MFI. This method for finding the maximal elements include the work of iteratively attempting to extend a working pattern until failure by maintaining a candidate set of maximal patterns which help in reducing the number of database scans, by eliminating non-maximal sets early. The maximal candidate set is a superset of the maximal patterns, and in general, the overhead of maintaining it can be very high. GenMax integrates pruning with mining and returns the exact MFI. This involves a list of algorithms and optimizations used during the calculation of MFI. The optimizations of GenMax contain two steps:

Superset checking optimization: The main efficiency of GenMax stems from the fact that it eliminates branches that are subsumed by an already mined maximal pattern.

Frequency testing optimization: GenMax uses a vertical database format, where we have available for each itemset, its tidset- the set of all transaction tids where it occurs.

GenMax is illustrated with the help of an example, the transaction database contains itemsets BDUX, DEX, BDUX, BDEX, BDEUX, DEU and the minimum support (min_sup) value is taken to be 3. Consider the backtracking algorithm for mining all frequent patterns as discussed in GenMax. The main loop tries to extend Il (which is initially empty {}) with every item x in the current combine set Cl which initially contains {B,D,E,U,X}.

The first step is to compute Il+1, which is simply Il {} extended with x{B}. The second step is to extract the new possible set of extensions, Pl+1,{D,E,U,X} which consists only of items y in Cl that follow x. The third step is to create a new combine set for the next pass, consisting of valid extensions. An extension is valid if the resulting itemset is frequent. The combine set, Cl+1, {D,U,X}thus consists of those items in the possible set that produce a frequent itemset when used to extend Il+1 Any item not in the combine set {E} refers to a pruned subtree. The final step is to recursively call the backtrack routine for each extension. So, the prefix tree obtained is:

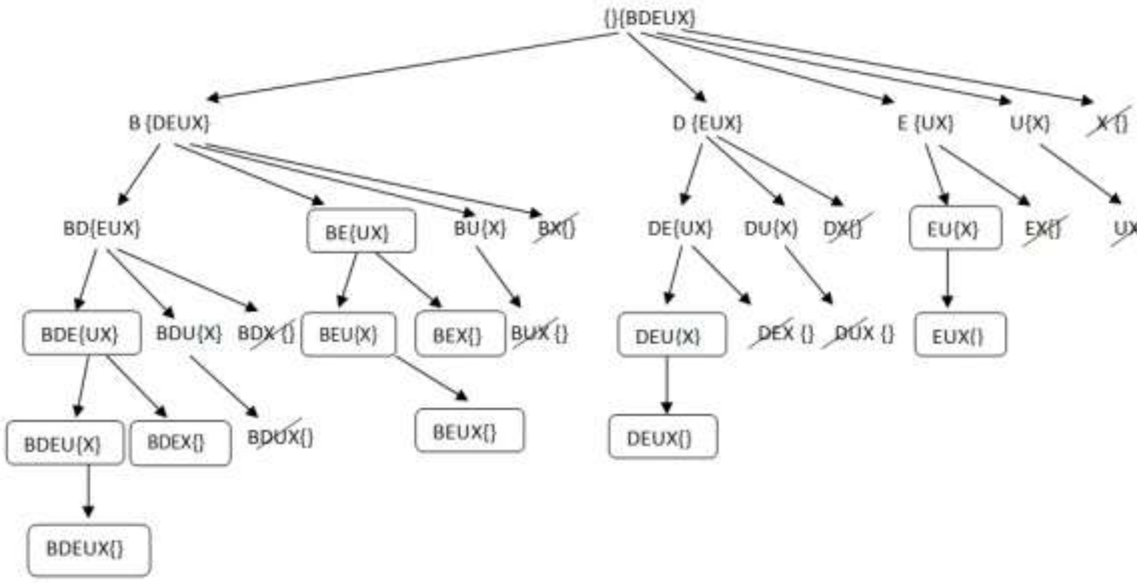


Figure 2.1: Prefix tree generated at minimum support equal to 3

The Figure 2.7 shows the prefix tree where the circled items represent the infrequent itemsets, which were pruned away from the tree. It returns the LMFI list as BDUX, BDX, BUX, BX, DEX, DUX, DX, EX, UX, X which is shown with the crossed lines, as the largest frequent itemset in every branch of the tree.

The iterations identified while generating the MFI list from the LMFI list using GenMax Algorithm are:

LMFI={BDUX, BDX, BUX, BX, DEX, DUX, DX, EX, UX, X}

LMFI={BDUX, BDX, BUX, BX, DEX, DUX, DX, EX, UX, X}

LMFI={ BDUX, BUX, BX,DEX, DUX, DX, EX, UX, X}

LMFI={ BDUX, BX, DEX, DUX, DX, EX, UX, X}

LMFI={ BDUX,DEX, DUX, DX, EX, UX, X}

LMFI={ BDUX,DEX, DX, EX, UX, X}

LMFI={ BDUX, DEX, EX,UX, X}

LMFI={ BDUX, DEX, UX, X}

LMFI={ BDUX, DEX, X}

LMFI={ BDUX, DEX }

Hence, MFI = { BDUX, DEX} can be obtained in 10 steps for the given example.

III. PROPOSED ALGORITHM

This section discusses the proposed algorithm, NewGenMax that improves the efficiency for mining MFI. The proposed algorithm tries to overcome the shortcomings of the GenMax algorithm by introducing the concept of subset checking. With the increase in the size of LMFI (Local Maximal Frequent Itemsets) list in GenMax, the time for checking superset increases and at the same time the size of prefix tree grows larger. Thus making excess memory utilization and also takes more steps which is not satisfactory.

3.1 NewGenMax Algorithm

NewGenMax assumes the input dataset to be in the vertical tidset format. For reducing the number of intermediate steps or iterations, following steps must be followed:

(i) Sort the LMFI in decreasing order of length of itemsets.

(ii) For each itemset I_i in LMFI.

(iii) Generate all possible subsets of Ii and store it in S.

(iv) Take the set difference of set LMFI with S and store it as LMFI.

(v) Return MFI=LMFI

The working of the proposed algorithm is discussed in Figure 3.1 where the input parameter Transaction denotes the dataset of transactions and min_sup defines the user predefined threshold support value.

```
void NewGenMax (Transaction, min_sup){
for (each transaction){
  string st=read_file()                    //File read
  sym_list = check_symbol(st)
//let sym_list stores the list of symbols obtained from function check_symbol
}                    // for loop closed
LMFI= tree ()      //returns LMFI list by using symbol list
LMFI = createMFI()
          //update LMFI list and return LMFI = MFI
}          //main loop closed
```

Figure 3.1: Algorithm for NewGenMax

In NewGenMax three pointer structures are maintained, *transaction point towards the transaction list, *fi points to the frequent item list obtained and *symbol points to the symbol list obtained from the transaction database. Each structure uses cur, front and prev to point the current, front and previous position value of the list.

Initially, the input transaction database file is read where each line of database represents a distinct transaction. Transaction is taken in the form of a string and then the function checksymbol() is called for each transaction string. Figure 3.2 describes function checksymbol() that check for the symbols occuring in all transactions using function insert_symbol() for the length of itemlist and results in a sorted symbol list denoted by sym_list.

```
check_symbol (char * S){
for (length of itemlist in each transaction){
 insert_symbol (S)
}          //for loop closed
}          //main loop closed
```

Figure 3.2: Function check_symbol()

The function insert_symbol () shown in Figure 3.3 sorts the symbol list which is returned as an output of function check_symbol().

```
 insert_symbol (char *s){
for (i= 0 to length of sym_list){
  if  s is there in sym_list then break
  else insert s at position i in the sym_list
}                         //for loop closed
}                         //main loop closed
```

Figure 3.3: Function insert_symbol()

The function tree() shown in Figure 3.4 check all the branches of a tree for all the symbols obtained through check_symbol().

```
tree(){
for(i=sym_list){
  for(j= sym_list){
     Check(for each sym_list value from position i ,j )
                                        //returns LMFI list
}      // inner for loop closed
}      //outer for loop closed
}      //function closed
```

Figure 3.4: Function tree()

Function Check () takes the input symbol list as shown in Figure 3.4 and prunes the infrequent subtrees from every branch of the prefix tree depending upon the min_sup value as defined in Figure 3.5 and returns the list of local maximal frequent itemsets list.

```
int Check(char *string,int val){
 strcpy(symbol_string,each value of sym_list));
{
if((strcmp(string,symbol_string)==0)and(Count(string,val)==1))
then send(string)  //adds string to the LMFI list
 else if(Count(string,val)==1){
   for(c=val+1 to length of symbol)
                         //traverses for all the subtrees
   strcpy(st,ADD(string,val))
   if(Check(st,c)==1){
      adds the char to sym_list
    }        //while closed
  }          //for closed
 }           //if closed
}            //main loop closed
```

Figure 3.5: Function Check()

Function Count() used in Figure 3.5 returns 1 if the string passed is frequent else it return 0. The function createMFI() discussed in Figure 3.6 generates the MFI list from the LMFI list.

```
createMFI(){
```

```
t=0; start: for(each i=length of frequent item){
for(each j=1+length of frequent item)
 if(t==i) increment t
if(the itemset at position i contains the itemset at position j) then delete itemset at position j and goto start
   }}
```

Figure 3.6: Function createMFI()

3.2 An example to illustrate working of NewGenMax Algorithm

To illustrate the proposed algorithm, consider an example database given in Table 3.1. Each transaction is traversed with the help of function check_symbol() shown in Figure 3.2 that results in a list containing all the distinct symbols occurred in the transaction database in a sorted manner.

Table 3.1: Transactional Database

TID	Itemsets
1	BDUX
2	DEX
3	BDUX
4	BDEX
5	BDEXU
6	DEU

Five different symbols obtained from the given database are {B, D, E, U, X}. The frequent and maximal frequent itemsets with their respective itemsize are shown in Table 3.2 at min_sup value equal to 3.

Table 3.2: Frequent and Maximal Itemset Database

Itemset Size	Frequent Itemsets min_Sup = 3	Maximal Itemsets min_Sup = 3
1	B, D, E, U, X	
2	BD, BU, BX, DE, DU, DX, EX, UX	
3	BDU, BDX, BUX, DUX, DEX	DEX
4	BDUX	BDUX

Using the database given in Table 3.1, prefix tree generated at min_sup equal to 3 is given in Figure 3.7 and at minimum support equal to 3 is shown in Figure 3.7.

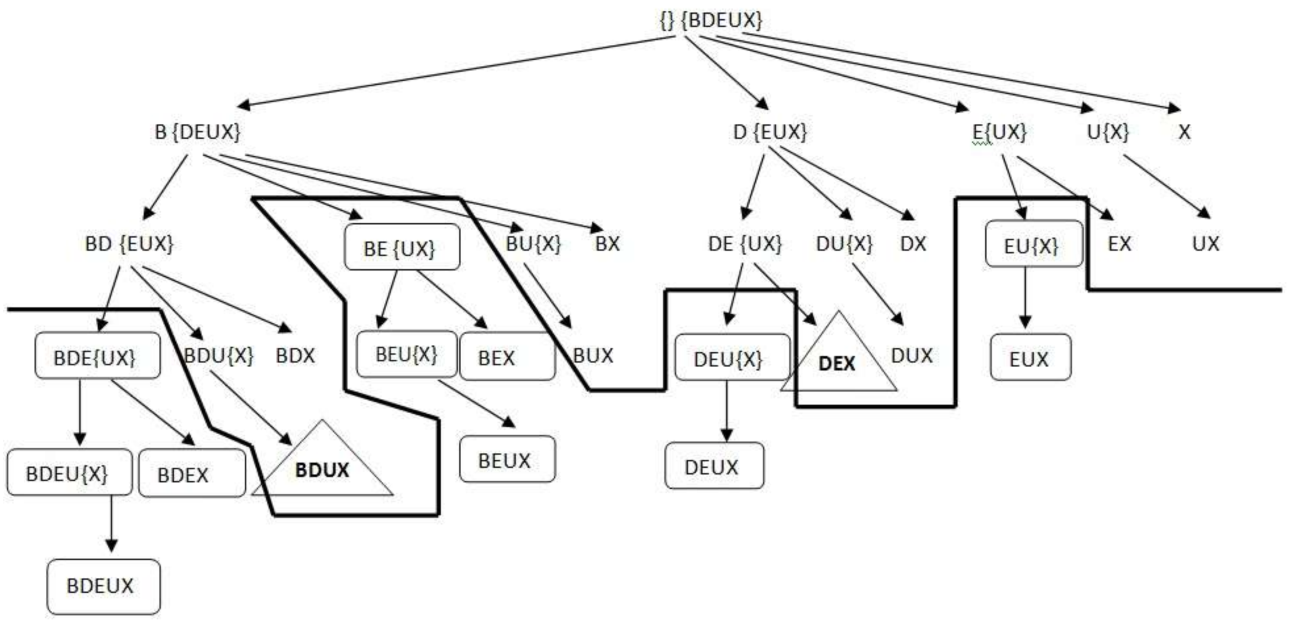


Figure 3.7: Prefix tree generated at mininmum support equal to 3

In Figure 3.8, circles indicate the maximal frequent itemsets and the crossed lines shows the infrequent itemsets. Due to the downward closure property (all subsets of a frequent itemset must be frequent) the frequent itemsets form a border (shown by the bold line in Figure 3.8), such that all frequent itemsets lie above the border, while all infrequent itemsets lie below it.

The list of local maximal frequent itemsets generated from the prefix tree contains:

LMFI= {BDUX, BDX, BUX, DEX, DUX, BX, DX, EX, UX, X}

The proposed algorithm NewGenMax generates less number of iterations in order to generate MFI from LMFI. LMFI list is read from left to right. After reading each itemset, the subsets of the itemset need to be stored in a new list S. Set difference of list S with the LMFI list results in an updated LMFI list. The process is repeated till the last itemset of LMFI is scanned. From the LMFI list given above, the first itemset identified is BDUX. The subsets list S generated from BDUX is { {}, B, D, U, X, BD, BU, BX, DU, DX, UX, BDU, BDX, BUX, DUX, BDUX}. Taking the set difference of LMFI list with list S, updates the LMFI list to {BDUX, DEX, EX}.

In the similar fashion, the subsets of all the itemsets in the LMFI list are removed from the LMFI list. This process continues until the whole LMFI list is scrolled, resulting in final LMFI list having itemsets {BDUX, DEX}.

Steps generated while converting LMFI list into MFI list using NewGenMax Algorithm are:

LMFI={BDUX, BDX, BUX, DEX, DUX, BX, DX, EX, UX, X}

LMFI={BDUX, DEX, EX}

LMFI={BDUX, DEX}

This updated LMFI list is then returned as MFI which can be obtained in only 3 iterations. Hence, MFI = {BDUX, DEX}.

IV. PERFORMANCE EVALUATION

In this section, a systematic and realistic set of experiments were performed to show the performance evaluation of proposed algorithm over the existing algorithm. Experiments were performed on an Intel I3 processor with 2GB of memory, running Ubuntu. Section 4.1 shows the datasets that were used in the performance evaluation and Section 4.2 refers to the experiments conducted.

4.1 Datasets Used

Performance of the proposed algorithm has been done with the existing algorithm on both synthetic and real datasets. Typically, the real datasets are very dense, i.e. they produce many long frequent itemsets even for high values of support.

The synthetic datasets were generated using IBM Synthetic Data Generator. Table 4.1 shows the synthetic datasets generated along with their nomenclature where T denotes the approximate number of transactions in the datasets generated, L denotes average number of items per transaction and N denotes total number of items in the dataset.

Table 4.1: Datasets taken from IBM Synthetic Data Generator

Dataset	Total number of Transactions (T)	Average length of Transactions (L)	Total number of distinct Items (N)
T100L10N10	100	10	10
T100L20N10	100	20	10
T100L30N10	100	30	10
T200L10N10	200	10	10
T200L20N10	200	20	10
T200L30N10	200	30	10
T300L10N10	300	10	10
T300L20N10	300	20	10
T300L30N10	300	30	10
T400L10N10	400	10	10
T400L20N10	400	20	10
T400L30N10	400	30	10
T500L10N10	500	10	10
T500L20N10	500	20	10
T500L30N10	500	30	10

4.2 Results

The performance of the two algorithms, significantly vary with the two parameters: execution time and the number of steps identified while generating MFI from LMFI which further depends upon the dataset characteristics. On the basis of average length of itemset per transactions, synthetic datasets were divided into three categories viz. Category A, Category B and Category C. The average number of items per transactions in datasets belonging to Category A, Category B and Category C are 10k, 20k and 30k respectively.

4.2.1 Synthetic Datasets

Table-4.2, Table-4.3 and Table-4.4 shows the comparison for the proposed algorithm and existing algorithm on datasets belonging to Category A, Category B and Category C respectively. The execution time comprises input file reading time, time used for the filtering, sorting and recording of items, time used for sorting of transactions, intersecting time and output file writing time.

Figure 4.1(a) and Figure 4.1(b) corresponding to Table 4.2, presents the comparison for execution time and the number of steps identified while converting LMFI into MFI respectively. In the similar way, Figure 4.2(a) and Figure 4.2(b) were drawn for Table 4.3 and Figure 4.3(a) and Figure 4.3(b) were drawn corresponding to Table 4.4

Table 4.2: Comparison of execution time and number of iterations required in generating MFI from LMFI for both NewGenMax and GenMax on datasets belonging to Category A

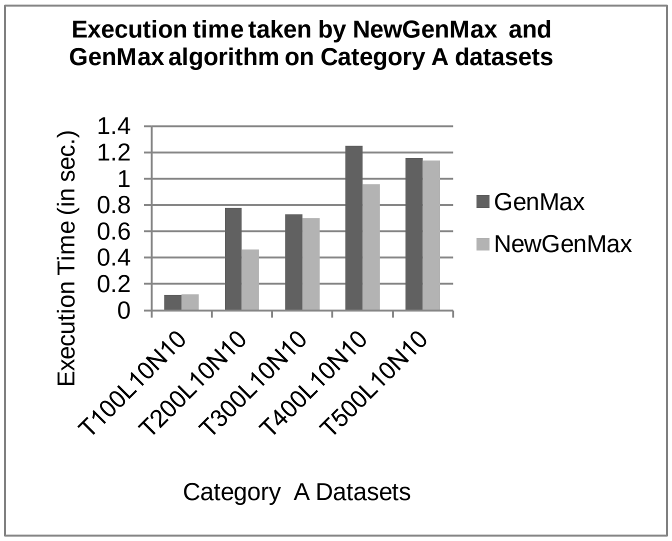
Category A Datasets	**GenMax**		**NewGenMax**	
	Execution Time	**No. of iterations in generating MFI from LMFI**	**Execution Time**	**No. of iterations in generating MFI from LMFI**
T100L10N10	0.115194	164	0.121289	33
T200L10N10	0.778955	167	0.461328	34
T300L10N10	0.729365	164	0.700329	41
T400L10N10	1.2489	166	0.95662	50
T500L10N10	1.15662	164	1.13839	49

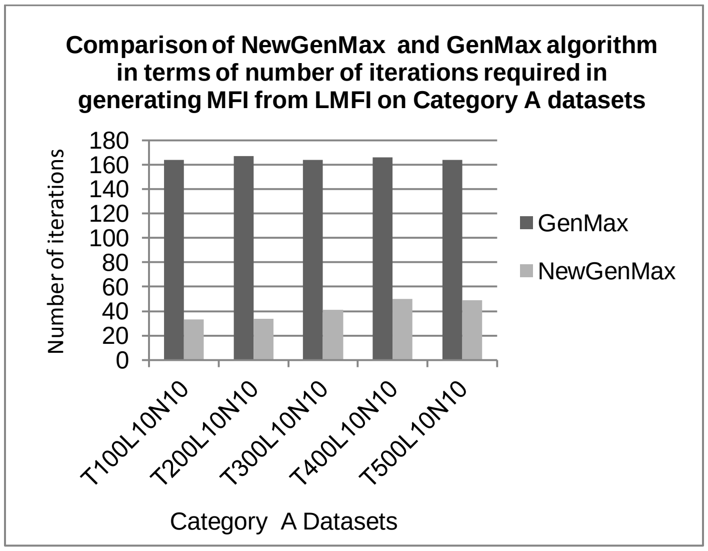
Table 4.3: Comparison of execution time and number of iterations required in generating MFI from LMFI for both NewGenMax and GenMax on datasets belonging to Category B

Category B Datasets	**GenMax**		**NewGenMax**	
	Execution Time	**No. of iterations in generating MFI from LMFI**	**Execution Time**	**No. of iterations in generating MFI from LMFI**
T100L20N10	1.13659	1134	0.832613	226
T200L20N10	1.67913	1136	1.66313	227
T300L20N10	2.8432	1124	2.56942	281
T400L20N10	3.76112	1128	3.39873	338
T500L20N10	4.22238	1129	4.2037	339

Table 4.4: Comparison of execution time and number of iterations required in generating MFI from LMFI for both NewGenMax and GenMax on datasets belonging to Category C

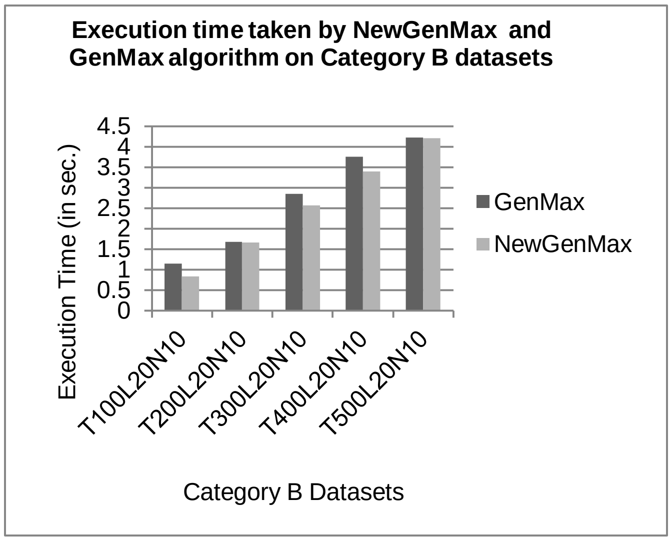
Category C Datasets	**GenMax**		**NewGenMax**	
	Executio n Time	**No. of iterations in generating MFI from LMFI**	**Execution Time**	**No. of iterations in generating MFI from LMFI**
T100L30N10	1.6092	2358	1.63354	471
T200L30N10	3.7386	2276	3.26916	455
T300L30N10	4.91607	2270	4.84378	567
T400L30N10	6.50973	2278	6.54509	683
T500L30N10	8.20685	2282	8.05916	685

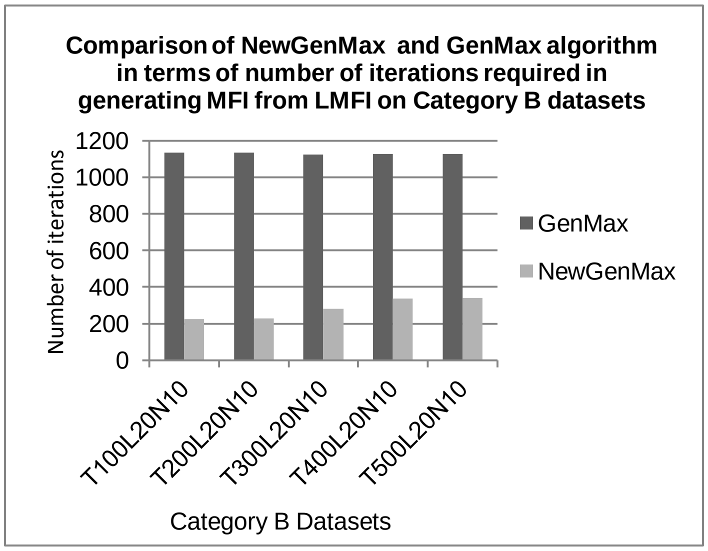




(a) **(b)**

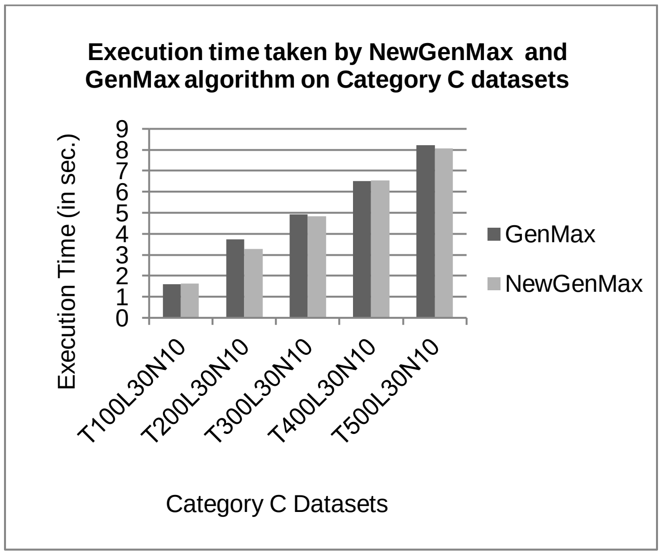
Figure 4.1 (a) Execution Time taken by NewGenMax and GenMax on Category A datasets, (b) Number of iterations required in generating MFI from LMFI by NewGenMax and GenMax on Category A datasets

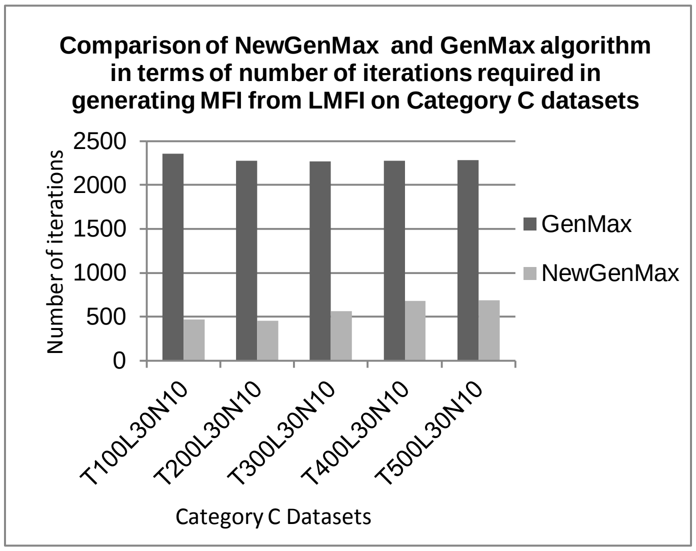




(a) **(b)**

Figure 4.2 (a) Execution Time taken by NewGenMax and GenMax on Category B datasets, (b) Number of iterations required in generating MFI from LMFI by NewGenMax and GenMax on Category B datasets





(a) **(b)**

Figure 4.3 (a) Execution Time taken by NewGenMax and GenMax on Category C datasets, (b) Number of iterations required in generating MFI from LMFI by NewGenMax and GenMax on Category C datasets

From these figures, it can be inferred that proposed algorithm NewGenMax gives better results for all the three types of synthetic datasets discussed in Table 4.2, Table 4.3 and Table 4.4. From figures, it can also be concluded that NewGenMax converts LMFI into MFI in less number of iterations leading to reduced execution time.

4.2.2 Real Datasets

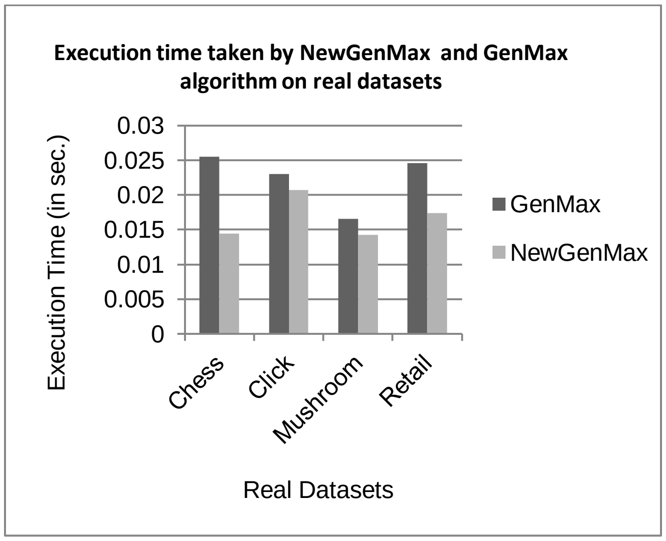
This section gives the performance evaluation of NewGenMax with GenMax on real datasets. Chess and mushroom datasets are taken fom the UCI (University of California, Irvine) Machine Learning Repository, click and retail datasets are taken from FIMI(Frequent Itemset Mining Implementation) Repository where the click dataset contains data related to real time browsing pattern whereas dataset named retail contains the (anonymized) retail market basket data from the anonymous Belgian retail store.

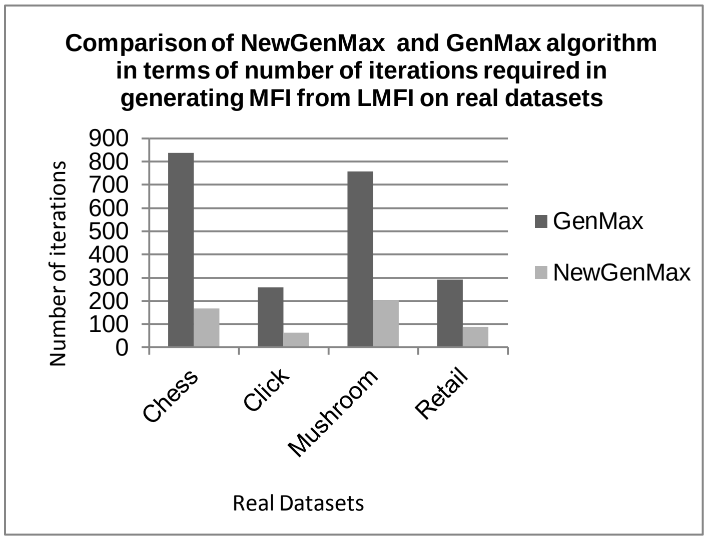
Figure 4.4(a) and 4.4(b) corresponding to Table 4.5, presents the comparison of execution time and the number of iterations needed to obtain MFI from LMFI respectively.

From these figures, it can be inferred that the proposed algorithm gives better performance on real datasets than the existing algorithm.

Table 4.5: Comparison of execution time and number of iterations required in generating MFI from LMFI for both NewGenMax and GenMax on Real Datasets

Real Datasets	GenMax		NewGenMax	
	Execution Time	No. of iterations in generating MFI from LMFI	Execution Time	No. of iterations in generating MFI from LMFI
Chess	0.0254998	838	0.0144269	168
Click	0.023021	260	0.020721	62
Mushroom	0.016521	758	0.0142591	205
Retail	0.024615	293	0.0173669	88





(a) **(b)**

Figure 4.4 (a) Execution Time taken by NewGenMax and GenMax on real datasets, (b) Number of iterations required in generating MFI from LMFI by NewGenMax and GenMax on real datasets

V. APPLICATION AREAS

The proposed frequent itemset mining algorithm can be applied in the areas of market basket analysis, web log analysis, cross Marketing, catalog design, product assortment decisions and intrusion detection System. Section-5.1 discusses Market Basket Analysis in detail.

5.1 Market Basket Analysis (MBA) and Association Rule mining

MBA is a mathematical modeling technique based upon the theory that it identifies customers purchasing habits. It provides insight into the combination of products within a customer's 'basket' viz further termed as a transaction. Top progressive retailers are using the MBA to win margin and market share that will help them increase their success and provide them with the edge that they need. As retailing is becoming a high performance sport, retailers are seeking a competitive edge through technology. MBA, also known as affinity analysis, has emerged in the evolution of retail merchandising and promotion. It allows leading retailers to quickly and easily look at the size, contents, and value of their customer's market basket to understand the patterns in how products are purchased together and also offers more advanced capabilities to interact with the transaction data to discover patterns, affinities and associations. Ultimately, the purchasing insights provide the potential to create cross sell propositions:

- Which product combinations are bought like monitor, central processing unit(CPU), keyboard and mouse
- When they are purchased

Developing this understanding enables businesses to promote their most profitable products. It can also encourage customers to buy items that might have otherwise been overlooked or missed. MBA delivers a list of potentially interesting products (based on a profile of what other "similar" customers have ordered). They are seeking to encourage the purchase of additional items and thereby increase the average basket value. A major task of talented merchants is to pick the profit generating items and discard the losing items. This type of analysis is certainly not the exclusive domain of the supermarkets. Transaction database in some applications can be very large. It may be simple enough to sort items by their profit and make the selection whereas some transaction database requires sophisticated analysis. For example, Wal-Mart in Hedberg quoted about 20 million sales transactions per day.

However, a very important aspect is ignored in market analysis viz the cross selling effect. There can be items that do not generate much profit by themselves but they are the catalysts for the sales of other profitable items. Advanced implementations of MBA encouraged retailers to drill down into customer buying patterns over time to precisely target and understand specific combinations of products, departments, brands, categories and even time of day. A 1% lift in sales or 0.1% improvement in margin, can tip the balance between success, survival, or failure. But below a retailer's top line sales, success requires constant fine tuning of the controls available to the retail disciplines, such as planning, buying, advertising, promotions, assortments, site selection, etc.

With an MBA, leading retailers can drive more profitable advertising and promotions, attract more customers, increase the value of the market basket, and much more. Leading practices in the MBA include more profitable advertising and promotions, more precise targeting of offers, attracting more traffic into the store, increasing the size and value of the market basket, testing and learning by using the market place as a laboratory, determining the magic price point for the store, matching the inventory to the customer need, etc that results in the optimized store layout which improves success across the board.

Usually the transaction dataset format consists of transaction ID and its corresponding itemsets where the transaction ID uniquely identifies each transaction and the item list shows a list of items that were purchased. Most of the retail shops have a format: Tid < item list >

In order to convert the market basket format into NewGenMax input format, consider every product in the market as an item, each customer's basket as a transaction and the set of products within that basket as an itemset. As some transaction ID is given to each transaction in order to handle large databases easily , that makes us keeping the transaction ID instead of transaction name. Similarly, every item name is replaced with its unique ID like T1 <1, 3, 5> means items with ID 1, 3 and 5 are contained in transaction 1 that is denoted by T1. Table 5.1 shows five transactions with their respective itemsets:

Table 5.1: Format of Retail Shop Database

TID	Itemsets
1	1, 3, 5
2	1, 7, 19
3	8, 13, 26, 52
4	2, 5, 13, 16, 28, 40
5	18, 27, 33, 34, 36, 52

VI. CONCLUSIONS

This paper presents a novel algorithm NewGenMax for finding maximal frequent itemsets. NewGenMax is giving better performance than the GenMax algorithm. This has been maded possible by reducing the number of iterations while converting local maximal frequent itemsets into maximal frequent itemsets.

In NewGenMax, the procedure of calculating subset for each itemset in the LMFI list and taking its set difference results in less number of iterations. Based on our theoretical and experimental analysis, NewGenMax generates the same MFI as in GenMax but with the earlier pruning of non-maximal itemsets, the time for scrolling the whole LMFI list decreases leading to less memory utilization. Both NewGenMax and GenMax are implemented and their performance is evaluated on the basis of execution time and the number of iterations generated while converting MFI from LMFI.

VII. REFERENCES

[1]R. Agrawal, C. Aggarwal, and V. Prasad, “Depth First Generation of Long Patterns” in ACM SIGKDD Conference, August 2000.

[2]R. Agrawal, T Imielinski, A Swami, “Mining association rules between sets of items in large databases” in ACM SIGMOD Conference, Washington, pp 207–216, 1993.

[3]R. J. Bayardo, “Efficiently mining long patterns from databases” in ACM SIGMOD Conf., June 1998.

[4]F. Bodon, “Surprising Results of Trie-based FIM Algorithms” in Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI '04 in Brighton, UK, November 1, 2004, Vol-126.

[5]D. Burdick, M. Calimlim, and J. Gehrke, “MAFIA: a maximal frequent itemset algorithm for transactional databases” in International Conference on Data Engineering, April, 2001.

[6]B. Chandra, S. Bhaskar, “A Novel Approach for Finding Frequent Itemsets in Data Stream”, Int. J. Intell. Sys., 28(3), pp. 217-241, 2013

[7]B. Chandra, S. Bhaskar, “A novel approach for finding frequent itemmsets in high speed data streams”, FSKD 2011, pp. 40-44

[8] B. Chandra, S. Bhaskar, “Patterned Growth algorithm using Hub-Averaging without pre-assigned weights”, SMC 2011, pp. 3518-3523

[9]K. Gouda and Mohd. J. Zaki, “Genmax: An Efficient Algorithm for Mining Maximal Frequent Itemsets”, in IEEE International Conference on Data Mining and Knowledge Discovery, vol. 11 Springer Science and Business Media,pp. 1-20,2005.

[10]J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation” in Proceedings of ACM SIGMOD, pages 1–12, May 2000.

[11]C. Lucchese, S. Orlando and R. Perego, “DCI Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets” in Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI '04, Vol-126.

[12]G. Grahne, and J. Zhu, “Fast Algorithms for Frequent Itemset Mining Using FP-Trees” IEEE Transactions on Knowledge and Data Engineering, Vol. 17, NO. 10, October 2005 1347

[13]G. Grahne and J. Zhu, “Efficiently Using Prefix-trees in Mining Frequent Itemsets ” in Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI '03.

[14]W. A. Kosters and W. Pijls, “Apriori, A Depth First Implementation” in Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, FIMI '03, Vol-90.

[15]C. Sathya and C. Chandrasekar, “Indexed Enhancement on GenMax Algorithm for Fast and Less Memory Utilized Pruning of MFI and CFI” in International Journal of Computer Applications 41(Vol. 16):37-41, New York, USA, March 2012.

[16]S. B. Bajaj, “ARAS: Efficient generation of Association Rules Using Antecedent Support, FSKD 2014, pp. 289-294