Indexed Enhancement on GenMax Algorithm for Fast and Less Memory Utilized Pruning of MFI and CFI

C.Sathya
Assistant Professor in Computer Application
Vellalar College for Women
Erode, Tamil Nadu, India

C. Chandrasekar
Associate Professor in Computer Science
Periyar University
Salem, Tamil Nadu, India

ABSTRACT

The essential problem in many data mining applications is mining frequent item sets such as the discovery of association rules, patterns, and many other important discovery tasks. Fast and less memory utilization for solving the problems of frequent item sets are highly required in transactional databases. Methods for mining frequent item sets have been implemented using a prefix-tree structure, known as an FP-tree, for storing compressed information about frequent item sets which is too large to fit in memory. GenMax, a search based algorithm is used for mining maximal frequent item sets. GenMax uses a number of optimizations to prune the search space. It uses a technique called progressive focusing to perform maximal checking, and differential set propagation to perform fast frequency computation. The proposal in this paper present an improved index based enhancement on GenMax algorithm for effective fast and less memory utilized pruning of maximal frequent item sets and closed frequent item sets. The proposed model reduce the number of disk I/Os and make frequent item set mining scale to large transactional databases. Experimental results shows a comparison of improved index based GenMax and existing GenMax for efficient pruning of maximal frequent and closed frequent item sets in terms of item precision and fastness.

General Terms
Computer Science, Data Mining, Association Rule Mining
Keywords
Indexing, Indexed GenMax, Itemset Mining

1. INTRODUCTION

In recent Data mining research, mining of association rules from large data sets has been addressed efficiently. Mining frequent itemsets is an initial requirement for mining association rules. Frequent item set mining has many applications such as association rule mining, inductive databases, and query expansion[1]. From these applications, fast implementations of frequent itemset mining problems are needed. For a given large data base of set of items transactions, we need to find all frequent itemsets, where a frequent itemset is one that occurs in at least a user-specified percentage of the data base. Many of the proposed itemset mining algorithms are a variant of Apriori, which employs a bottom-up, breadthfirst search, that enumerates every single frequent itemset[2]. In many applications (especially in dense data) with long frequent patterns enumerating all possible 2m - 2 subsets of a m length pattern (m can easily be 30 or 40 or longer) is computationally unfeasible. Thus, there has been recent interest in mining maximal frequent patterns in these ”hard” dense databases.

The problem of mining maximal frequent patterns can be formally stated as follows[3]: Let I = {i1, i2, i3, . . . , i m } be a set of m distinct items. Let D denote a database of transactions, where each transaction has a unique identifier (tid) and contains a set of items. The set of all tids is denoted T = {t1, t2, tm, . . . , tn }. A set X $\subseteq$ I is also called an temset. An itemset with k items is called a k-itemset. The set t (X) $\subseteq$ T, consisting of all .the transaction tids which contain X as a subset, is called the tidset of X. For. convenience we write an itemset {A, C, W} as ACW. and its tidset { 1,3,4,5} as t (X) = 1345. The support of an itemset X denoted by σ(x) is the number of transactions in which that itemset occurs as a subset. This σ(x) = | t(x)|. An itemset is frequent if its support is more than or equal to some threshold minimum support value. We denote the set of all frequent itemset as FI. A frequent itemset is called maximal if it is not a subset of any other frequent itemset. The set of all maximal frequent itemset is denoted by MFI.

Bayardo [4] devised the MaxMiner algorithm to efficiently mine long patters from databases. Which abandons a bottom-up traversal and uses a look ahead technique to identify long patterns along with its subsets. Agrawal et. al. [5] formulated the DepthProject, that aims at depth first generation of long patterns. It is fast when compared to the previous MaxMiner. Burdick et. al. [6] proposed the MAFIA algorithm, which integrates depth-first traversal of itemset lattice with effective pruning mechanism. It generates the maximal frequent itemsets for the transactional databases. This algorithm mines the superset of maximal frequent itemset and requires a post pruning step to eliminate non maximal patterns.

Karam Gouda et. al. [7] formulated the GenMax algorithm that merges pruning with mining and returns the exact maximal frequent itemsets. The GenMax algorithm 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 also a novel progressive focusing technique is used to eliminate non-maximal itemsets easily. Diffset propagation is followed here for fast frequency checking We require an algorithm that also enumerates the set of all maximal frequent itemsets combined with the closed frequent itemsets for easy generation of association rules. Another promising direction is to mine only closed sets [8]; a set is closed if it has no superset with the same frequency. The set of all such closed frequent itemsets is called as CFI. Nevertheless, for some of the dense datasets, even the set of all closed patterns would grow to be too large. The only recourse is to mine the maximal patterns in such domains. Knowing all maximal patterns (and their frequencies) allows us to reconstruct the set of frequent patterns. Knowing all closed patterns and their frequencies allows us to reconstruct the set of all frequent patterns and their frequencies. In this paper, indexed based [9] progressive deepening method is developed by extension of the GenMax algorithm for mining maximal frequent item set. The method first finds frequent data items at the top most level and then progressively deepens the mining process into their frequent descendants. One important assumption that we have made in this study is to explore only the descendants of the frequent items, since we consider if an item occurs rarely, its descendants will occur even less frequently and, thus, are uninteresting. It also enumerates all the possible closed frequent item sets and construct an index for each MFI and CFI generated for fast and less memory utilized pruning of frequent item sets.

2. EXISTING WORK ON GENMAX

2.1 Backtracking Search

GenMax uses backtracking search to enumerate the maximal frequent itemsets (MFI). The backtracking paradigm starts with enumerating all frequent patterns. Backtracking algorithms are useful for many combinatorial problems where the solution can be represented as a set I = $\{i_o,i_1, ...\}$, where each i_j is chosen from a finite possible set, P_j. Initially I is empty; it is extended one item at a time, as the search space is traversed. The length of l is the same as the depth of the corresponding node in the search tree. Given a partial solution of length 1, $I_l = \{i_o, i_1, ..., i_{l-1}\}$, the possible values for the next item i_1 comes from a subset C_l which is a subset of P_l called the combine set. If y Є $P_l - C_l$, then nodes in the subtree with root node $I_1 = \{i_o, i_l,..., i_{l-1}, y\}$ will not be considered by the backtracking algorithm. Since such subtrees have been pruned away from the original search space, the determination of C_l is also called as pruning.

```
$\mathbf{FI-Backtrack}(I_l, C_l, l)$
$\text{For each } x \in C_l$
 $I_{l+1} = I \, U \, \{x\}$
        $P_{l+1} = \{y : y \in C_l \text{ and } y > x\}$
          $C_{l+1} = FI\_Combine(I_{l+1}, P_{l+1})$
$FI-Backtrack(I_{l+1}, C_{l+1}, l+1)$
$\mathbf{FI\_Combine}(I_{l+1}, P_{l+1})$
   $C = \varnothing$
   $\text{For each } y \in P_{l+1}$
$\text{If } I_{l+1} \cup \{y\} \text{ is frequent}$
   $C = C \cup \{y\}$
 Return C
```

Fig 1: Backtrack Algorithm

Consider the backtracking algorithm for mining all frequent patterns, shown in Figure 1. The main loop tries extending I_l with every item x in the current combine set C_l. The first step is to compute I_{l+1}, which is simply I_l extended with x. The second step is to extract the new possible set of extensions, P_{l+1}, which consists only of items y in C_l 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, C_{l+1}, thus consists of those items in the possible set, that produce a frequent itemset when used to extend $I_{l+1.}$ Any item not in the combine set refers to a pruned subtree. The final step is to recursively call the backtrack routine for each extension. Thus the presented backtrack method performs a depth-first traversal of the search space.

2.2 MFI Mining algorithm

There are two main ingredients to develop an efficient MFI algorithm. The first is the set of techniques used to remove entire branches of the search space, and the second is the representation used to perform fast frequency computations. The basic MFI enumeration code used in GenMax is a straightforward extension of FI-Backtrack**.** The main addition is the superset checking to eliminate non-maximal itemsets, as shown in Figure 2.

```
$\text{Invocation : } MFI-Backtrack(\varnothing, F_l, 0)$
$\mathbf{MFI\ -Backtrack}(I_{1,} C_{1,} l)$
$\text{For each } x \in C_l$
$I_{l+1} = I \cup \{x\}$
$P_{l+1} = \{y : y \in C_1 \text{ and } y > x\}$
$\text{If } I_{l+1} \cup P_{1+1} \text{ has a superset in MFI}$
$\text{Return // all subsequent branches pruned}$
$C_{l+1} = FI- \text{ combine } (I_{l+1,} P_{1+1})$
$\text{If } C_{l+1} \text{ is empty}$
 $\text{If } I_{l+1} \text{has no super set in MFI}$
      $MFI = MFI \cup I_{l+1}$
$\text{Else}$
$MFI- Backtrack(I_{l+1,} C_{l+1,} l+1)$
```

Fig 2: Backtrack Algorithm for Mining MFI

After the construction of the possible set to check if I_{l+1}U P_{1+1} is subsumed by an existing maximal set. If so, the current and all subsequent items in C_l can be pruned away. After creating the new combine set, if it is empty and I_{l+1} is not a subset of any maximal pattern, it is added to the MFI. If the combine set is non-empty a recursive call is made to check further extensions.

2.3 Optimizing the Algorithm

The next step is to substantially speed up the subset checking process. The main idea is to progressively narrow down the maximal itemsets of interest as recursive calls are made. In other words, we construct for each invocation of MFI-Backtrack a list of local maximal frequent itemsets, $LMFI_l$. This list contains the maximal sets that can potentially be supersets of candidates that are to be generated from the itemset I_l. The only such maximal sets are those that contain all items in I_l. This way, instead of checking if I_{l+1}U P_{1+1} is contained in the full current **MFI,** we check only in LM FI_1 the local set of relevant maximal itemsets. This technique called progressive focusing, is extremely powerful in narrowing the search to only the most relevant maximal itemsets, making superset checking practical on dense datasets. Figure 3 shows the pseudo-code for GenMax that incorporates this optimization. Any new maximal itemsets from a recursive call are incorporated in the current $LMFI_l$.

```
Invocation : LMFI backtrack $(\varnothing, F_l, \varnothing, 0)$
$LMFI_l$ is an output parameter
**LMFI – Backtrack** $(I_l, C_1, LMFI_1, l)$
For each $x \in C_1$
    $I_{l+1} = I \cup \{x\}$
    $P_{l+1} = \{y : y \in C_1 \text{ and } y > x\}$
    if $I_{l+1} \cup P_{l+1}$ has a superset in $LMFI_1$
     Return //subsequent branches pruned
    $LMFI_{l+1} = \varnothing$

    $C_{l+1} = Fl-combine(I_{l+1}, P_{1+1})$
    If $C_{l+1}$ is empty
    If $I_{l+1}$ has no superset in $LMFI_l$
     $LMFI_1 = LMFI_l \cup I_{l+1}$
    Else $LMFI_l = \{M \in LMFI_l, x \in M\}$
     LMFI – Backtrack $(I_{l+1}, C_{1+1}, LMFI_{1+1}, l+1)$
    $LMFI_1 = LMFI_1 \cup LMFI_{l+1}$
```

Fig 3: Mining MFI with Progressive focusing

2.4 Differential Sets Propagation

Despite the many advantages of the vertical format, when the tidset cardinality gets very large (e.g., for very frequent items) the intersection time starts to become inordinately large. Furthermore, the size of intermediate tidsets generated for frequent patterns can also become very large to fit into main memory. GenMax uses a new format called Differential Sets (diffsets) for fast frequency testing. The main idea of diffsets is to avoid storing the entire tidset of each element in the combine set. Instead we keep track of only the differences between the tidset of itemset I_1 and the tidset of an element x in the combine set. These differences in tids are stored in what we call the diffset, which is a difference of two tidsets at the root level or a difference of two diffsets at later levels. Furthermore, these differences are propagated all the way from a node to its children starting from the root.

```
**Fl – diffset – combine** $(I_{l+1}, P_{l+1})$
$C = \varnothing$
For each $y \in P_{l+1}$
$y' = y$
 If level $= 0$ then $d(y') = t(I_{l+1}) - t(y)$

Else $d(y') = d(y) - d(I_{l+1})$
If $\sigma(y') \geq min\_sup$
$C = C \cup \{y'\}$
Return C
```

Fig 4: Diffset Propagation

2.5 Final Algorithm

The complete GenMax algorithm is shown in Figure 5, which ties in all the optimizations mentioned above. GenMax assumes that the input dataset is in the vertical tidset format. First GenMax computes the set of frequent items and the frequent 2-itemsets, using a vertical-to-horizontal recovery method. This information is used to reorder the items in the initial combine list to minimize the search tree size that is generated. GenMax uses the progressive focusing technique of LMFI-Backtrack, combined with diffset propagation of FI-diffset-combine to produce the exact set of all maximal frequent itemsets.

```
**GenMax** Compute $F_1$ and $F_2$
Compute $IF(x)$ for each item $x \in F_1$
Sort $F_1$ (decreasing in $IF(x)$, increasing in $\sigma(x)$)
$MFI = \varnothing$
LMFI – Backtrack $(\varnothing, F_1, MFI, 0)$ // Use diffsets
Return MFI
```

Fig 5: GenMax Algorithm

3. THE PROPOSED ALGORITHM

3.1 Mining Closed Frequent Itemset CFI

It is important to point out the relationship between frequent itemsets, closed frequent itemsets and maximal frequent itemsets. As mentioned earlier closed and maximal frequent itemsets are subsets of frequent itemsets but maximal frequent itemsets are a more compact representation because it is a subset of closed frequent itemsets. Therefore all frequent itemsets are uniquely determined by the Closed itemsets and can be determined by the join operation on the frequent concepts. Closed frequent itemsets are more widely used than maximal frequent itemset because when efficiency is more important that space, they provide us with the support of the subsets so no additional pass is needed to find this information.

For any Closed Itemset X, there exists a Closed Tidset Y, with the property :Y = t(X). The Pair X × Y is called a Concept**.** Therefore all frequent itemsets are uniquely determined by the Closed itemsets and can be determined by the join operation on the frequent concepts. A large number of generalized frequent itemsets may cause of high computational time. Instead of mining all generalized frequent itemsets, we can mine only a small set of generalized closed frequent itemsets and then result in reducing computational time. Here we proposed an indexed algorithm, by applying some constraints and conditional properties to efficiently enumerate only generalized closed frequent itemsets. The advantage of this approach becomes more dominant when minimum support is low and/or the dataset is dense. This approach makes us possible to mine the data in real situations.

3.2 Indexed Enhancement on GenMax Algorithm

Indexed GenMax performs a novel search for closed sets using subset properties of differential sets. The initial invocation is with a indexed class at a give tree node. All differences for pairs of elements are computed. However in addition to checking for frequency, indexed GenMax eliminates branches and grows item sets using subset relationships among differential sets. This proposed Index based GenMax is the efficient item pruning model. The Indexed GenMax consists of indexed structure for tid set, GenMax maximal frequent item sets generator, differential sets, and resultant closed (minimal) frequent items with performance factors such as memory utilization and pruning time. The improved indexed based GenMax Algorithm as designed for the implementation process is presented below.

```
Input:   Car Data transaction set TS,
         minimum support value (min_supp),
         item set.
Output:  Complete set of frequent items (FI),
         GenMFI (maximal frequent item set),
         GenCFI (closed frequent item),
         Index of GenMFI and GenCFI.
1. INITIALIZATION
1.1. Let DI = {i1,i2,...in} be a set of n distinct items
1.2. Let TS denote a database of transaction set,
         where each transaction has unique identifier (tid)
         and a set of items
1.3. The set of all tids is denoted S = {t1,t2,..,tm}
1.4. A set x Є I is also called an item set
1.5. An item set with k items is called a k-item set
and let the items be denoted by i in item set x.
1.6. Initialize min_supp and GMFI, GCFI to 0.
2. PROCESS (transaction set)
2.1. Scan the transaction set TS.
2.2. By using x and min_supp, FI and NFI are generated.
3. OBTAINING GMFI
3.1. Frequent item set (FI) and NFI obtained from 2.2
3.2. Generate GMFI
3.3.     for all item sets x Є FI
3.4.     for all item sets s Є NFI
3.5.         if x is a subset of s
3.6.         while superset(candidate item set) Є GMFI
3.7.             GMFI = 0
3.8.         while superset(candidate item set) € GMFI
3.9.             candidate item set = GMFI
3.10.        endif
4. OBTAINING GCFI
4.1. Let list=(i, count)
4.2.     for each x Є FI
4.3.      if list <> superset in GCFI & support = min_sup
4.4.         GCFI = GCFI U list
4.5.     endif
5. PERFORM INDEXING ON GMFI and GCFI
5.1.     Using GMFI and GCFI
5.2.         for n frequent items
5.3.         store frequent item in index[j].list in vector (vec)
5.4.           scan the transaction set
5.5.           if vec < = IS
5.6.                 Perform(Tid,i)
5.7.           Endif
6. QUERY PROCESSING (Frequent item pruning)
6.1.     For (i <=x)
6.2.         perform step 5 for indexed GMFI and GCFI
6.3.   If (i=x)
6.4.   Resultant frequent item dependent object is obtained
6.5.       Otherwise perform with next item
6.6.     Endif
```

Fig 5: Indexed GenMax Algorithm

The implementation of Indexed GenMax for mining multiple-level item sets i.e., maximal and closed, which uses a transaction id based information encoded indexed table instead of the original transaction table. This is because a data mining query is usually in relevance to only a portion of the transaction database of indexed hierarchy structure, instead of all the items. It is beneficial to first collect the relevant set of data and then work repeatedly on the task-relevant set. Encoding can be performed during the collection of task-relevant data and, thus, there is no extra encoding pass required. Besides, an encoded string, which represents a position in a indexed table, requires fewer bits than the corresponding object-identifier or bar-code. Therefore, it is often beneficial to use an encoded indexed table, although our method does not rely on the derivation of such an encoded table because the encoding can always be performed on the pruning process itself. The steps of implementation are briefed as below.

Step 1: Calculate frequent item sets at each concept level, until no more Frequent Item sets can be found
Step 2: Indexing of Frequent Item(FI) set
Step 3: Pruning the Index of FI for Closed and Maximal FI
Step 4: Progressive Focusing approach on indexed structure
Step 4.1: Fast retrieval of Frequent Item sets
Step 4.2: Identification of Precise Frequent Item sets

Uniform Support is the same minimum support for all levels i.e., one minimum support threshold. It is not needed to examine item sets containing any item whose ancestors do not have minimum support in the indexed GenMax model. If support threshold is too high, it misses low level associations. If the support threshold is too low, it generates too many high level associations. Reduced Support is the reduced minimum support at lower levels to evaluate the closed frequent items with efficient index structure for different possible strategies to prune the item sets from large scalable data mining application such as market basket analysis, genome property structures, medical disease diagnosis etc.,

4. PERFORMANCE EVALUATION

To analyze the performance of the indexed GenMax item pruning, we used Pentium Dual core processor 2.5 GHz with 2 GB of main memory running under Windows XP. The training sample consists of transactions in car databases taken from UCI repository. The total number of items, being 100, the number of potentially frequent item sets to be 30, and the total number of transactions were 3000. The scale-up tests on total number of items, average size of transactions, and total number of tuples, also performed to verify the efficiency of the indexed GenMax item pruning model which showed satisfactory results briefed in below sections.

4.1 Mining Frequent Item Set

The transaction database is converted into an indexed encoded transaction table, according to the information above the generalized items in the item description. The maximal level of the concept hierarchy in the item table is set to 4. The number of the top level nodes keeps increasing until the total number of items reaches. The fan-outs at the lower levels are selected based on the normal distribution, and with a variance of 2.0. Not every strong rule so discovered (i.e., passing the minimum support and minimum confidence thresholds) is interesting enough to be presented to users. Two interestingness measures are proposed to filter redundant rules and unnecessary rules. The indexed based GenMax prune more precise frequent item set compared to that of existing GenMax algorithm (shown in Fig 6).

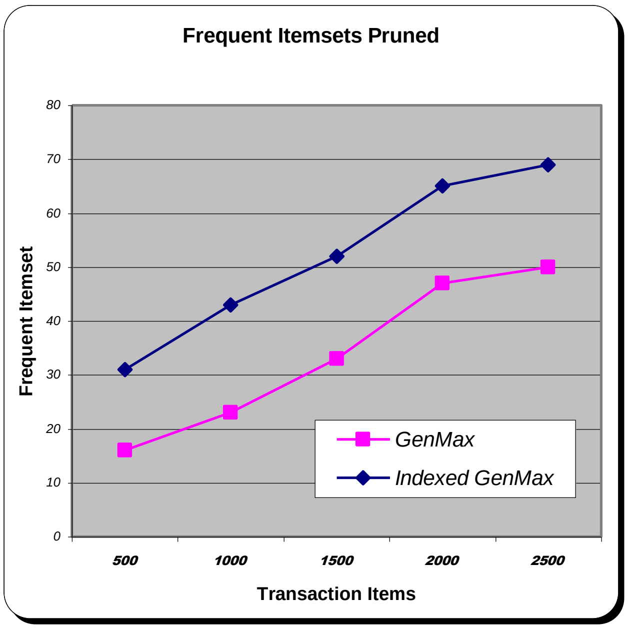

Fig 6: Frequent Itemsets Pruned by Indexed GenMax

4.2 Pruning Time for Indexed based GenMax

The pruning time for indexed based GenMax is low for mining maximal and closed frequent item set compared to that of existing GenMax (shown in Fig 7). The reduction of time in pruning for indexed GenMax is achieved due to the segregation of multilevel hierarchies of the item sets and mainly due to the indexing of transaction id based on the complete item set. However in case of GenMax the pruning is done for the frequent pattern on all levels of the item set. In case of large transaction, without indexing sequence, exiting GenMax have to prune the frequent patterns from top to bottom for any nearest neighbor pattern also which consumes more time nearly 18% than the indexed GenMax.

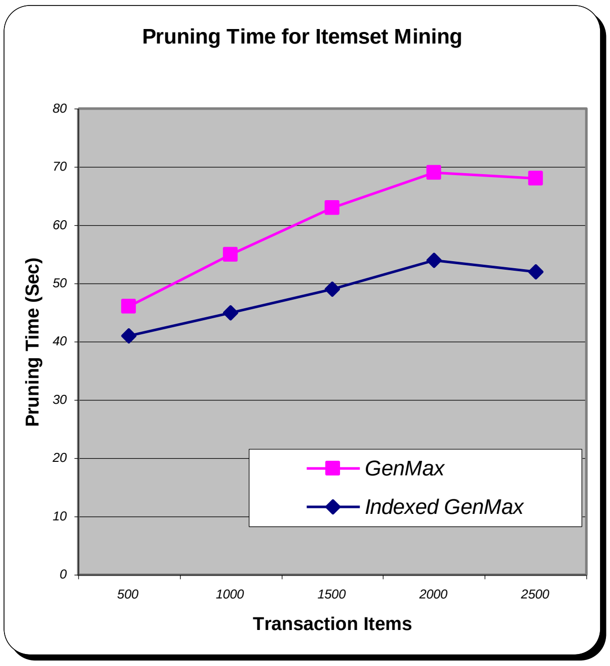

Fig 7: Pruning time for mining MFI and CFI

5. CONCLUSION

The proposal in our work has implemented an improved and an index based enhancement on GenMax algorithm. This Indexed GenMax algorithm is used for effective fast and less memory utilized pruning of maximal frequent item and closed frequent item sets. The extension induces a search tree on the set of frequent closed item sets thereby we can completely enumerate closed item sets without duplications. The memory use of mining the maximal frequent item set does not depend on the number of frequent closed item sets, even if there are many frequent closed item sets. The top-down progressive deepening technique using indexing structure is developed for mining maximal and closed frequent items. The better performance of indexed GenMax algorithm depicted in the result section worked on different distributions of data. Our performance study shows that the indexed GenMax may have the best performance when compared to the conventional GenMax.

6. REFERENCES

[1] Go'sta Grahne, and Jianfei Zhu, " Fast Algorithms for Frequent Itemset Mining Using FP-Trees " IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 10, OCTOBER 2005 1347

[2] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Proceedings of ACM SIGMOD'00, pages 1–12, May 2000.

[3] G. Grahne and J. Zhu. High performance mining of maxi-mal frequent itemsets. In SIAM'03 Workshop on High Performance Data Mining: Pervasive and Data Stream Mining, May 2003.

[4] R. J. Bayardo, "Efficiently mining long patterns from databases", In ACM SIGMOD Conference, June 1998.

[5] R. Agrawal, C. Aggarwal, and V. Prasad, "Depth First Generation of Long Patterns", In ACM SIGKDD Conference, August, 2000.

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

[7] Karam Gouda and Mohammed j. Zaki, "GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets" , In IEEE International Conference on Data Mining and Knowledge Discovery, Volume 11, pp. 1–20, 2005.

[8] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In ACM SIGMOD'00 Workshop on Research Issues in Data Mining and Knowledge Discovery, pages 21–30, 2000.

[9] E. Baralis, T. Cerquitelli, and S. Chiusano, "Index Support for Frequent Itemset Mining in a Relational DBMS", In proceedings of 21st International Conference on Data Engineering (ICDE '05), Tokyo, pp. 754-765, April 2000.