A Novel Approach to Mining Maximal Frequent Itemsets Based on Genetic Algorithm

Mir Md. Jahangir Kabir, Shuxiang Xu, Byeong Ho Kang, Zongyuan Zhao

Abstract -- We present a new approach based on Genetic Algorithm to generate maximal frequent itemsets from large databases. This new algorithm called GeneticMax is heuristic which mimics natural selection approaches to finding maximal frequent itemsets in an efficient way. The search strategy of this algorithm uses lexicographic tree that avoids level by level searching, which finally reduces the time required to mine maximal frequent itemsets in a linear way.

Our implementation of the search strategy includes bitmap representation of the nodes in a lexicographic tree and from superset-subset relationship of the nodes it identifies frequent itemsets. Since this new algorithm uses the principles of Genetic Algorithm, it performs global search and its time complexity is less than that of other algorithms, for the reason that genetic algorithm is based on greedy approach. We separate the effect of each step of this algorithm by experimental analysis on real databases including Tic Tac Toe, Zoo, a 10000x8 Database, and so on. Our experimental results show that this approach is efficient and scalable for different sizes of itemsets. It accesses a major database to calculate a support value for fewer number of nodes to find frequent itemsets even when the search space is very large, which dramatically reduces the search time.

Index Terms-- Data mining, genetic algorithm, lexicographic tree, maximal frequent itemsets.

I. INTRODUCTION

Mining frequent itemsets is one of the fundamental and essential issues in various data mining applications such as consumer market-basket problem, discovery of association rules, deducing patterns and correlations, network invasion detection and other important data mining tasks. The problem is formulated as follows: a set of items and a large collection of transactions have been given, each transaction is a subset of these items, find all frequent itemsets. The number of frequent itemsets is defined by a user specified percentage value of the database.

This problem concerns Association Rule mining which includes two steps: first, mining frequent itemsets from a large database, and second, generating association rules or correlation relationship among a large set of data items. Nowadays, huge amounts of data are collected and stored by industries, who are interested in mining frequent itemsets from large databases. The discovery of association rules among large amount of business transactions helps industries make business decisions [1]–[3].

Let \( D = \{ t_1, t_2, t_3, \ldots, t_k, \ldots t_n \} \) be the database, where \( t_1, t_2, \ldots, t_n \) are the number of transactions in the database. Each transaction \( t_n \) is a set of items \( I = \{ i_1, i_2, \ldots, i_k, \ldots i_n \} \), where \( i_1 \) is item number 1, \( i_n \) is item number \( n \) and so on. Transaction \( t_n \) is represented as a binary vector. If \( t_n[k] = 1 \), then it means that \( t_n \) bought the item \( i_k \), otherwise \( t_n[k] = 0 \). Let \( X \) be a set of few items in \( I \) i.e. \( X \subseteq I \). The set \( t_n(X) \subseteq I \) is true for all items in itemset \( X \) for transaction \( t_n \). The support value of an item is how many times the item appears in the transaction database as a subset. The support value of an itemset is denoted by \( \sigma(X) = \left| \left\{ t_1(X) + t_2(X) + t_3(X) + \ldots + t_n(X) \right\} \right| / |D| \). Here \( t_n(X) \) gives the binary value. If the examined itemset \( X \) appears as a subset in a transaction \( t_n \), then \( t_n(X) = 1 \), otherwise \( t_n(X) = 0 \).

An itemset with 1 item is called a 1-itemset, an itemset with \( k \)-items is called a \( k \)-itemset. An itemset is called frequent if its support value is more than or equal to a user defined threshold value, which is denoted by \( \min_{\text{supp}} \) (minimum support) i.e. \( \sigma(X) \geq \min_{\text{supp}} \). We denote frequent itemsets by FI. If an itemset \( X \) is frequent and no superset of \( X \) is frequent then we can claim that \( X \) is a maximal frequent itemset and we denote the set of all maximal frequent itemsets by MFI.

To generate MFI form a large database is a most time consuming task in the present day. In this paper, we present a novel approach to finding maximal frequent itemsets from large databases by using the principles of Genetic Algorithm (GA).

The major advantage of GA based approach is that it performs global search and its time complexity is less than that of other algorithms. Another advantage is that it generates frequent itemsets independent of the size of a database. This work differs from existing research in the following aspects: 1) Our new approach named GeneticMax uses a lexicographic tree as a search tree and it does not need to enumerate frequent itemsets level by level.

All the authors are with the School of Computing and Information Systems, University of Tasmania, Australia.
Mir Md. Jahangir Kabir (telephone: +61-3-63243695, e-mail: mnjkabir@utas.edu.au).
Shuxiang Xu (telephone: +61-3-63243416, e-mail: Shuxiang.Xu@utas.edu.au).
Byeong Ho Kang (e-mail: Byeong.Kang@utas.edu.au).
Zongyuan Zhao (e-mail: Zongyuan.Zhao@utas.edu.au).
level; 2) This approach uses the principles of Genetic Algorithm which randomly generates chromosomes. If a generated chromosome is in a positive boundary area, then all the subsets of this chromosome are automatically pruned. If a generated chromosome is in a negative boundary area, then all the supersets of this chromosome are automatically pruned. This technique dramatically reduces the time for accessing a large database to calculate the support value of unnecessary chromosomes to find frequent itemsets.

II. RELATED WORKS

It is well known that Apriori algorithm generates a candidate set and tests it in a breadth-fast manner. It discovers all the frequent itemsets at level k before moving to its next level (k+1). It counts the support value of each node in level k and prunes those nodes if the support values of those nodes do not satisfy a user define support value. It generates candidate itemsets at each level and scans the database so frequently that it is costly, especially when there exists a long pattern [2].

Breadth first traversal (a level by level search strategy on a search space) is applied for a MaxMiner search algorithm. To prune the branches of a tree it performs a look-ahead method. MaxMiner uses breadth first approach for limiting the number of passes over the database but look-ahead, which involves superset pruning, works better for depth first search methods [4].

MAFIA, proposed by Burdick, Calimlim, and Gehrke [5], extends the idea of DepthProject. Similar to DepthProject, MAFIA also uses vertical bitmap representation where the support value/count of an itemset is based on AND operations among the itemsets. Look-ahead pruning methodology which was first used by MaxMiner is also used by MAFIA. The last checking method of MAFIA is easy to test. Without counting $A \cup C$, it allows us to conclude that $\{A, C\}$ is frequent. This technique is defined as Parent Equivalence Pruning in.

In [3], Gouda and Zaki proposed a novel approach called GENMAX to find maximal itemsets. In this approach they used a novel technique called Progressive Focusing. Researchers of GENMAX concluded that, through experimental results this algorithm performs better than existing algorithms on different types of databases.

Bilal Alataş and Erhan Akin [6] designed an efficient genetic algorithm as a search strategy to mine both positive and negative quantitative association rules. Association rules are deduced from frequent patterns. This method mined the association rules without generating frequent itemsets. The proposed genetic algorithm does not depend on minimum support and confidence value which is hard to define for a database. A new genetic operator named uniform operator is used in this approach which ensures genetic diversity.

In [7], quick response data mining model based on genetic algorithm has been designed. This approach gives more flexibility to the user. Long frequent itemsets are generated because of the higher relationship among data tuples. This approach avoids considering huge candidate itemsets. It only scan the database for those frequent itemsets users are more interested.

To mine quantitative association rules researchers proposed a new algorithm which is based on genetic algorithm named QUANTMINER [8], [9]. By optimizing support and confidence value, this system dynamically identify good intervals in association rules. Researchers applied this algorithm in different data sets and showed the usefulness of this algorithm as a data mining tool.

III. THE IDEA OF FAST RESPONSE

If data tuples contain long itemsets it generates huge candidate itemsets which finally reduces the efficiency of a solution. A long itemset enumerates combinatorial number of shorter, frequent sub-itemsets. For example, a data tuple contains 50 itemsets, such as \{i_1, i_2, i_3, ..., i_{50}\} which enumerate $\binom{50}{1}$ frequent 1-itemsets: \{i_1, i_2, ..., i_{50}\}. $\binom{50}{2}$ frequent 2-itemsets: \{i_1, i_2\}, \{i_1, i_3\}, ..., \{i_{1}, i_{50}\}, \{i_{2}, i_{3}\}, \{i_{2}, i_{4}\}, ..., \{i_{2}, i_{50}\} and so on.

Lemma 1: If the length of an itemset is $n$, then it enumerates $2^n - 1$ frequent sub-itemsets.

This is too huge for a computer to compute and store if the length of an itemset is long. For each sub-itemset, Apriori algorithm needs to be used to scan the database and calculate the support value of that itemset which increases the computational time of the algorithm and decreases the efficiency of it. To overcome this low efficiency of Apriori algorithm we consider superset-subset relationship. An itemset $I$ is called maximal frequent itemset if the super itemset of $I$, denoted $\hat{I}$, is not frequent such that $I \subseteq \hat{I}$. Here $\hat{I}$ is an infrequent itemset based on a support value defined by a user.

Lemma 2: If an itemset $I$ is a frequent itemset then all the subsets of $I$ are frequent, based on a support value which is defined by a user.

For example, if an itemset $I = \{1, 2, 3\}$ in set $S = \{1, 2, 3, 4\}$ is frequent, i.e. $\sigma(I) \geq \min_{supp}$, then all the subsets of $I$, i.e. \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\} are frequent itemsets, based on a support value defined by a user. In our new approach GeneticMax, if the generated chromosome is $I = \{1, 2, 3\}$ and it satisfies a user define support value then it will not test all the subsets of $I$ which dramatically reduces computational time for scanning the database.
IV. DESCRIPTION OF GeneticMax

A. Itemsets Mapping to Chromosomes

GeneticMax maps itemsets onto a chromosome code. Each node in the lexicographic tree represents different itemsets and all the nodes in the tree get a unique chromosome code. The main feature of chromosome coding is: 1. It is easy to calculate the support value since GeneticMax uses bitmap representation of the database, 2. Generate all the possible nodes.

Fig. 1 shows an example of a lexicographic tree which considers lexicographic ordering for four items. The root of the tree is an empty set and each k-level contains k-items [5]. In each level, k-itemsets maintain lexicographic ordering with the tail nodes containing items lexicographically larger than elements of the head node. The support value of the head node is more than that of the tail node. It can be seen that the nodes closer to the root are more frequent than those far from the root. There is a non-linear line (called a cut) in the tree which separates frequent itemsets from infrequent ones.

Fig. 1. Lexicographic tree of four items

For GeneticMax, we introduce a new tree which is based on user define support values. The line is defined by a user define support value and the area above the line is referred to as a positive area and the area below the line is referred to as a negative area. All the nodes in a positive area are frequent whereas all the nodes in a negative area are infrequent. If we redesign the tree of Fig. 1, the lexicographic tree of these four items would be as shown in Fig. 2.

The length of a chromosome is fixed. If a database contains n items, then the length of all the generated chromosomes are always n.

Fig. 2. Lexicographic tree of four items based on an user define support value

The chromosomes look like the following:

| $V_{item_1}$ | $V_{item_2}$ | ... | $V_{item_n}$ |

For example: A database D contains 4 items and 1000 transactions. GeneticMax will generate $(2^4 - 1) = 15$ nodes in a lexicographic tree with each node representing one or more transactional patterns among 1000 transactions of database D. Let’s say, transaction k contains $item_{2}$ and $item_{4}$. This transaction represents the node (2,4) and the chromosome coding of this node is: 0101.

B. Lifetime of GeneticMax

The lifetime of GeneticMax depends on user’s selection of a generation. The higher the generation number the higher the probability for getting a correct solution. But there is a threshold value for a generation: after the threshold is reached the solution remains the same.

V. GeneticMax FOR EFFICIENT MFI MINING

There are five main requirements for developing an efficient MFI mining algorithm. We need a set of techniques which fulfil these requirements:

1) It will not scan a database more than once for a specific itemset.
2) If X is an itemset in a positive boundary area and there are no supersets of X and it has already been tested, then all the subsets of X are pruned and defined as invalid data sets.
3) If X is an itemset in a negative boundary area and there are no subsets of X and it has already been tested, then all the supersets of X are pruned and defined as invalid data sets.
4) It should maintain an interactive mining process, where users can change the threshold to get different sets of MFI.
5) It gives correct solutions for different sizes of databases.
Apriori algorithm and FP-Tree do not satisfy requirements 1, 2, 3 and 4 respectively. GeneticMax fulfils all the above requirements.

A. Procedures of GeneticMax

Step 1: Set a generation number.
Step 2: Generate population of GeneticMax.
Step 3: Check the FI_Superset_Member and NFI_Subset_Member array for superset and subset checking of this generated chromosome.
Step 4: If it finds any superset or subset in FI_Superset_Member or NFI_Subset_Member respectively, then go to Step 2.
Step 5: Compute a fitness value of individuals according to their support values in database D.
Step 6: Perform FI_Member_Add, and if any frequent itemsets are found then update FI_Superset_Member.
Step 7: Perform NFI_Member_Add, and if any infrequent itemsets are found then update NFI_Subset_Member.
Step 8: Go to Step 3 with newly generated chromosome until it exceeds the generation number which was set by Step 1.

B. Mining the superset in a positive boundary area

For an itemset X, if there is any subset of X in FI_Superset_Member, then this method is called to replace that subset by its superset X. This method is also applicable if X is a new frequent item with no subset in FI_Superset_Member.

C. Mining the subset in negative boundary area

For an itemset X, if there is any superset of X in NFI_Subset_Member, then this method is called to replace that superset by its subset X. This method is also applicable if X is a new infrequent item and it has no superset in NFI_Subset_Member.

D. GeneticMax Pruning Methods

// Invocation: Check_Member_for_Item (I, FI_Superset_Member, NFI_Subset_Member)
1. If any superset of I is in FI_Superset_Member
   2. Discard I
3. Else if any subset of I is in NFI_Subset_Member
   4. Discard I
5. Else scan the database to calculate support value for I
6. If support value ≥ user define support value
   7. Invoke FI_Member_Add
8. Else invoke NFI_Member_Add

Check_Member_for_Item function incorporates three techniques:

1. Superset Checking Techniques:
   Checking to see whether a given chromosome is a superset in a positive boundary area. Further pruning happens if a given itemset is not a superset in the positive boundary area.

2. Subset Checking Techniques:
   Checking to see whether a given chromosome is a subset in a negative boundary area. Further pruning happens if a given itemset is not a subset in the negative boundary area.

3. Unchecked itemset checking techniques:
   If an itemset is neither a superset in a positive boundary area and nor a subset in a negative boundary area, then this itemset is referred to as an “unchecked” itemset and needs to be tested. For this unchecked itemset, GeneticMax scans the database and sets the itemset in FI_Superset_Member or NFI_Subset_Member according to the user defined support value.

VI. EXPERIMENTAL RESULTS

The experiments were performed on an Intel(R) core i5-3210M CPU @2.50GHz, 4 GB RAM running on Windows 7 Enterprise. Microsoft Visual Studio 2012 was used to compile the code of GeneticMax. Three datasets including Tic Tac Toe, 10000×8, Zoo were used to test GeneticMax. These data sets were taken from the University of California at Irvine (UCI) machine learning repository (http://archive.ics.uci.edu/ml/datasets.html).

Different support values were applied on these datasets to check how many nodes have been tested and the numbers of chromosomes have been generated to get the exact number of maximal
frequent itemsets, run times, and so on. Here run time is the total execution time. The purpose of this new approach is for convergence to a solution as fast as possible. A full experiment of GeneticMax on these databases was conducted, demonstrating GeneticMax’s ability to yield solutions rapidly by accessing the databases for a few numbers of nodes in a lexicographic tree. Unlike Apriori GeneticMax generates a chromosome X in any level which satisfies a minimum support value, then all the other subsets of X in any level will be automatically pruned which dramatically reduces the time for accessing a large database. This is also true the other way around: if GeneticMax generates a chromosome Y in any level which does not satisfy a minimum support value, then all the other supersets of Y in any level will be automatically pruned.

From the experimental results as shown in Table 1, we seem to be able to conclude that if the number of generation is increased then it increases the frequent itemsets. For example, for the 10000x8 database, generation 100 produced 9 frequent itemsets whereas generation 140 produced 8 frequent itemsets. In other words, generation 100 resulted in more than 9 frequent itemsets. On the other hand, generation 140 resulted in more than 8 frequent itemsets. If we compare these two generations, we could conclude that generation 100 still did not find some frequent itemsets. When we increased the number of generation to 140, it found some itemsets missed by generation 100. Generation 150 gave the same result as generation 140. So users can use generation 140 as a threshold value for the 10000x8 database. This is also true for TicTacToe, generation 1200 and 1300 gave the same result which contains the maximal frequent itemsets. So for TicTacToe, user can use generation 1200 as a threshold value.

The results on Table 2 shows the comparison between number of node in a lexicographic tree and number of nodes tested for getting maximal frequent itemsets. For 10000x8, there are 255 itemsets and GeneticMax accessed only 39 itemsets in the main database to get the maximal frequent itemsets. Since GeneticMax uses the principles of genetic algorithm and prunes invalid chromosomes based on superset-subset relationship, it dramatically reduces the number of itemsets out of a database for getting the support value to mine maximal frequent itemsets. The advantage of using those principles in GeneticMax is showed in Table 2, where (255-39) = 216 nodes were not examined to get the support value from database 10000x8 to get the exact number of maximal frequent itemsets. Only 39 were examined to get the final solution. For TicTacToe, only 114 nodes were examined to get the final solution (the other 397 nodes were not required).

As we can see from Fig. 4, the runtime of GeneticMax increases with respect to the

![Fig. 4. Run time versus Generation for TicTacToe](image)

The results on Table 2 shows the comparison between number of node in a lexicographic tree and number of nodes tested for getting maximal frequent itemsets. For 10000x8, there are 255 itemsets and GeneticMax accessed only 39 itemsets in the main database to get the maximal frequent itemsets. Since GeneticMax uses the principles of genetic algorithm and prunes invalid chromosomes based on superset-subset relationship, it dramatically reduces the number of itemsets out of a database for getting the support value to mine maximal frequent itemsets. The advantage of using those principles in GeneticMax is showed in Table 2, where (255-39) = 216 nodes were not examined to get the support value from database 10000x8 to get the exact number of maximal frequent itemsets. Only 39 were examined to get the final solution. For TicTacToe, only 114 nodes were examined to get the final solution (the other 397 nodes were not required).

As we can see from Fig. 4, the runtime of GeneticMax increases with respect to the

![Fig. 4. Run time versus Generation for TicTacToe](image)

Table 1: The experimental results of GeneticMax for two different databases

<table>
<thead>
<tr>
<th>Database</th>
<th>Records</th>
<th>Items</th>
<th>Support (%)</th>
<th>Generation</th>
<th>Frequent Itemsets</th>
<th>Time (s)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000x8</td>
<td>10000</td>
<td>8</td>
<td>20</td>
<td>100</td>
<td>9</td>
<td>10.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>140</td>
<td>8</td>
<td>21.67</td>
<td>This generation contains MFI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>150</td>
<td>8</td>
<td>25.10</td>
<td></td>
</tr>
<tr>
<td>TicTacToe</td>
<td>958</td>
<td>9</td>
<td>16</td>
<td>100</td>
<td>6</td>
<td>10.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>250</td>
<td>7</td>
<td>17.528</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>500</td>
<td>10</td>
<td>43.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1100</td>
<td>23</td>
<td>78.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1200</td>
<td>24</td>
<td>95.60</td>
<td>Both Generations provide the same result</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1300</td>
<td>24</td>
<td>115.66</td>
<td></td>
</tr>
</tbody>
</table>
VII. CONCLUSIONS AND SUMMARY

In this paper we propose a new approach (named GeneticMax) based on Genetic Algorithm to mine maximal frequent itemsets in an efficient way. We have conducted experiments on different real data sets. The experimental results demonstrate several advantages of our algorithm in comparison with other existing algorithms:

1) It accesses a large database for fewer nodes to calculate a support value to find maximal frequent itemsets.
2) It shows the power of using evolutionary algorithm for generating frequent itemsets from lexicographic trees. A whole database is projected onto a lexicographic tree based on a user defined support value.
3) The experimental analysis of GeneticMax shows the effect of generations of chromosomes and pruning all the subsets and supersets in both positive and negative boundary areas, which dramatically reduces search space and cost of counting support value of itemsets.
4) The above advantages of GeneticMax increase the scalability of this algorithm.

The performance study shows that this algorithm mines different sizes of patterns in large databases in an efficient way, performs better than other candidate pattern generation and evolutionary based algorithms.

The goal of our future research is to explore parallel aspects of GeneticMax algorithm through more experimentation. We will develop this approach further in such a way that it will avoid generating invalid chromosomes and increase the generation of more valid chromosomes, which will increase its speed in converging to a solution.

REFERENCES


<table>
<thead>
<tr>
<th>Database</th>
<th>Items</th>
<th>Support (%)</th>
<th>No. of nodes in the Lexicographic Tree ($2^{\text{den} - 1}$)</th>
<th>No. of nodes tested for getting MFI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000x8</td>
<td>8</td>
<td>20</td>
<td>255</td>
<td>39</td>
</tr>
<tr>
<td>TicTacToe</td>
<td>9</td>
<td>16</td>
<td>511</td>
<td>114</td>
</tr>
<tr>
<td>Zoo</td>
<td>17</td>
<td>50</td>
<td>131072</td>
<td>361</td>
</tr>
</tbody>
</table>