IT in Industry, vol. 3, no. 3, 2015 Published online 15-Oct-2015

GeneticMax: An Efficient Approach to Mining Maximal Frequent Itemsets Based on Genetic Algorithms

Mir Md. Jahangir Kabir, Shuxiang Xu, Byeong Ho Kang, Zongyuan Zhao
School of Engineering and ICT
University of Tasmania
Launceston, Australia
{mmjkabir, Shuxiang.Xu, Byeong.Kang, Zongyuan.Zhao}@utas.edu.au

***Abstract*—This paper presents a new approach based on genetic algorithms (GAs) to generate maximal frequent itemsets (MFIs) from large datasets. This new algorithm, GeneticMax, is heuristic which mimics natural selection approaches for finding MFIs in an efficient way. The search strategy of this algorithm uses a lexicographic tree that avoids level by level searching which reduces the time required to mine the MFIs in a linear way. Our implementation of the search strategy includes bitmap representation of the nodes in a lexicographic tree and identifying frequent itemsets (FIs) from superset-subset relationships of nodes. This new algorithm uses the principles of GAs to perform global searches. The time complexity is less than many of the other algorithms since it uses a non-deterministic approach. We separate the effect of each step of this algorithm by experimental analysis on real datasets such as Tic-Tac-Toe, Zoo, and a 10000×8 dataset. Our experimental results showed that this approach is efficient and scalable for different sizes of itemsets. It accesses a major dataset to calculate a support value for fewer number of nodes to find the FIs even when the search space is very large, dramatically reducing the search time. The proposed algorithm shows how evolutionary method can be used on real datasets to find all the MFIs in an efficient way.**

Keywords*—*data mining; genetic algorithm; maximal frequent itemset; lexicographic tree

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 intrusion detection and other important data mining tasks. The problem is formulated as follows. Given a set of items and a large collection of transactions, 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 (support value) of the datasets.

This problem of association rule mining includes two steps: first, mining frequent itemsets from a large dataset, and second, generating association rules or correlation 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 datasets. The discovery of association rules among a large number of business transactions helps industries make business decisions [1–4]. A common real life application is the market basket analysis, in which retailers seek to understand the purchase behavior of customers. The data analysis attempts to find interesting hidden relationships among products purchased by customers through association rule mining or frequent pattern mining. For example, by using frequent pattern mining, a shop manager may discover that butter, milk and bread are frequently purchased together by customers. In another example, one association rule may indicate that when a customer buys coffee he would also buy milk. Such information can then be used for cross-selling and up-selling, sales promotions, store design, and discount plans. Similarly, a video shop manager can use frequent pattern mining and/or association rule mining to recommend related videos or games when a customer has hired or bought a specific video or game, to attract the customer to return. Web administrators can use frequent pattern or association rule mining to understand particular collections of web pages that are viewed together by a group of web users. This sort of interesting relationship (e.g., correlation, association) among data items helps managers make relevant decisions [5].

Let $\mathbf{D} = \{t_1, t_2, t_3, \ldots, t_i, \ldots, t_n\}$ is a dataset, where $t_1, t_2, t_3, \ldots, t_n$ are transactions. There are n transactions in total in the dataset. Each transaction t_i contains a subset of items $\mathbf{I} = \{i_1, i_2, i_3, \ldots, i_k, \ldots, i_m\}$, where i_1 is item number 1, i_2 is item number 2 and so on. Transaction t_i is represented as a binary vector. If $t_i[k] = 1$ then it means that t_i bought item i_k, otherwise, $t_i[k] = 0$. Let $\mathbf{X}$ be a subset of items in $\mathbf{I}$ i.e. $\mathbf{X} \subseteq \mathbf{I}$ .The set $t_i(\mathbf{X}) \subseteq \mathbf{I}$ is true for all items in itemset $\mathbf{X}$ for transaction t_i. The support value of an item is how many times the item appears in the transaction datasets as a subset. The support value of an itemset is denoted by $\delta(\mathbf{X}) = |\{t_1(\mathbf{X}) + t_2(\mathbf{X}) + t_3(\mathbf{X}) + \ldots + t_{n-1}(\mathbf{X}) + t_n(\mathbf{X})\}| \,/\, |\mathbf{D}|$

Here $t_i(\mathbf{X})$ gives the binary value. If the examined itemset $\mathbf{X}$ appears as a subset in a transaction t_i, then $t_i(\mathbf{X}) = 1$, otherwise $t_i(\mathbf{X}) = 0$. An itemset with 1 item is called a 1-itemset, and 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_supp* (minimum support) i.e. $\delta(\mathbf{X}) \geq$ *min_supp*, where $\delta(\mathbf{X})$ is a

ISSN (Print): 2204-0595
ISSN (Online): 2203-1731

support value of an examined itemset. We denote the 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 sets of all maximal frequent itemsets by **MFI** [6].

To generate the **MFI**s from a large dataset is time consuming. In this paper, we present a novel approach to finding **MFI**s from large datasets by using a GA. The length of an FI depends on its relationship among the itemsets. There are several advantages of a GA based approach that performs global searches. The time complexity of GA-based approach is significantly less than that of many other algorithms. Another advantage is that a GA-based approach is able to generate frequent itemsets from a large dataset. This work differs from the existing research in the following aspects:

1. Our GeneticMax uses a lexicographic tree as a search tree, and it does not need to enumerate frequent itemsets level by level.
2. This approach uses the principles of GA which randomly generates chromosomes. If a generated chromosome is frequent, then all the subsets of this chromosome are automatically pruned. If a generated chromosome is infrequent, then all the supersets of this chromosome are automatically pruned. This technique dramatically reduces the time for accessing a large datasets to calculate the support value of unnecessary chromosomes to find frequent itemsets.

II. Related Work

In data mining research, frequent pattern mining is one of the challenging and focused areas for over a decade. A large number of literature and research works have been dedicated to this research area, and significant progress has been made because of these efforts. Progress has been made in sequential pattern mining, correlation and structured pattern mining, scalable and efficient algorithms are designed for mining frequent itemsets and so on [6]. The scope of data analysis has been expanded by the research of frequent pattern mining and have profound impact on the methodologies and applications of data mining for further exploration. Though there has been plenty of progress made in frequent pattern mining, there are still some challenging issues need to be resolved. These critical research issues include:

Scalable mining methods, which are focused topics in frequent pattern mining research, have been extensively studied. Current mining methods are used to derive sets of frequent patterns. These sets of frequent patterns are too huge to use effectively. To reduce these huge sets, researchers have proposed several methods such as maximal patterns, representative patterns, closed patterns, condensed patterns and so on. But it is still undefined for a specific application which pattern set provides compactness and the quality of representation. Much investigation is needed to reduce the size of derived sets of patterns and to increase the quality of preserved patterns.

Some applications prefer approximate frequent patterns though current studies show that efficient methods are available for mining a complete and explicit set of frequent patterns. In bioinformatics to match with the biological entities, one could be interested in searching long sequence patterns in DNA analysis. Much investigation is needed to design efficient methods to make this mining more competent than the present tools available in bioinformatics.

Classification is another major task in data mining. In data mining, classifications using frequent patterns means which frequent patterns are more adequate over another. In the future, researcher should design a method in such a way that effective frequent patterns are mined directly from data [6].

Some applications require in depth understanding of patterns and interpretation of those patterns. Most of the researchers have focused on discovering frequent patterns but have given less attention to analysing and interpreting those patterns. The semantic analysis of a pattern includes the meaning of that pattern, the typical transactions that pattern considers and so on. Finding the reasons behind the frequency of a specific pattern is termed as contextual analysis of a frequent pattern. For example, a pattern could be frequent depending on specific time duration, location, weather and so on. To improve the interpretability, effectiveness and usability of a frequent pattern, it is necessary to have a deep understanding of frequent patterns [7].

It is well known that the 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 at level *k* and prunes those nodes if the support values of those nodes do not satisfy a user-defined support value. It generates candidate itemsets at each level and scans the datasets so frequently that it becomes costly, especially when there exists a long pattern [1].

The Pincer-search algorithm [8] traverses a lattice through a bi-directional method that follows both top-down and bottom-up approaches. To find the maximal frequent itemsets, pruning is applied based on the following rules:

1. All the subsets of frequent itemsets are pruned
2. All the supersets of infrequent itemsets are pruned.

The breadth-first traversal strategy (a level by level search strategy on the search space) was used in the MaxMiner search algorithm. To prune the branches of a tree, MaxMiner employs a look-ahead method. It uses the breadth-first approach to limit the number of passes over the datasets but with look-ahead which involves superset pruning, works better for depth-first search methods [9].

The DepthProject performs depth-first traversal on a lexicographic tree along with variations of superset pruning. To order child nodes, it applies dynamic reordering methods. By trimming infrequent items out of each node’s tail, it reduces the size of the search space. To eliminate non-maximal frequent itemsets the DepthProject would require post-pruning methods [10].

MAFIA, proposed by Burdick, Calimlim, and Gehrke [11], extends the idea of DepthProject. Similar to the DepthProject, MAFIA also uses vertical bitmap representation where the support value/count of an itemset is based on the bitwise AND operations among the itemsets.

An example of four items in a data tuple and the datasets are given in Fig. 1. Bitvectors for the 1-itemset **A**, **B**, **C**, **D** of Fig. 1.are 10111, 01001, 11110, and 11011, respectively. To get the support value/count of the itemset it needs to apply bitwise AND (&) operation between the bitvectors of the itemsets. For the above example, the result of the bitwise AND operation of bitvectors of itemsets **A** and **C** is 10111 & 11110 = 10110. The support value or count of an item is the number of 1's in the bitvector. Here the support value of the 2-itemset {**A**, **C**} is 3. To bitwise AND another bitvector **D** with the previous result of the bitwise AND operation of {**A**, **C**}, is 10110 & 11011= 10010. The support value of the 3-itemset {**A**, **C**, **D**} is 2. The search strategy of MAFIA integrates depth first method to traverse the tree to find maximal frequent itemsets along with effective pruning methodology. The look-ahead pruning methodology, first used by MaxMiner, was also used by MAFIA. The last checking method of MAFIA is easy to test. Without counting **A** ∪ **C**, it allows us to conclude that {**A**, **C**} is frequent. This technique is defined as Parent Equivalence Pruning in [11].

Gouda and Zaki proposed a novel approach called GENMAX to find maximal itemsets in [12]. In their approach, they used a novel technique called Progressive Focusing. This technique maintains local maximal frequent itemsets (LMFI) which are used for making comparison with newly found frequent itemsets (FI). Non-maximal frequent itemsets are identified through this step, and it decreases the number of subset testing. GENMAX uses a vertical representation of a datasets and stores a transaction identifier set (TIS) for each itemset instead of a bitvector. The support value of an itemset is defined by the cardinality of an itemset's TIS. Researchers of GENMAX concluded that through experimental results the algorithm performs better than existing algorithms for different types of datasets.

Alataş and Akin designed an efficient genetic algorithm as a search strategy to mine both positive and negative quantitative association rules in [13]. Association rules are deduced from frequent patterns. Different from other methods, their approach is used to mine 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 dataset. A new genetic operator named uniform operator is used in this approach to ensure genetic diversity.

Another interesting problem in data mining is classification. Different lengths of the itemsets are classified into different groups based on the frequencies of the itemsets. Concise symbolic rules with higher accuracies are mined using neural networks. To get the required accuracy, the network is initially trained. Network pruning algorithm is used to prune redundant connections. Classification rules are generated through the result of the analysis of activation values of the hidden layers. Researchers noticed that the main drawback of using neural networks in different data mining test problems was the training time. Though it provides lower classification error rate than decision trees, it requires long training time [14, 15].

A	B	C	D
1	0	1	1
0	1	1	1
1	0	1	0
1	0	1	1
1	1	0	1

Fig. 1. Vertical bitmap representation.

A quick response data mining model based on genetic algorithm has been designed in [16]. This approach gives more flexibility to the user to mine frequent itemsets. Long frequent itemsets are generated because of the higher relationships among data tuples. If the Apriori algorithm is used to mine frequent itemsets from these tuples, it could take a huge amount of time due to frequent access to the datasets and a large number of candidate itemsets generated. This approach avoids considering huge candidate itemsets. It only scans the datasets for those frequent itemsets that users are more interested. This system uses a GA to mine itemsets and then shows them to the user. If the users are interested, then it scans the datasets to get the support values of those itemsets.

To mine quantitative association rules, a GA-based algorithm named QUANTMINER was proposed in [17, 18]. By optimizing the support and confidence values, this system dynamically identifies good intervals in association rules. Researchers applied this algorithm to different datasets and showed the usefulness of this algorithm as a data mining tool.

R. J. Kuo and C. W. Shih used a new meta-heuristic technique, the ant colony system [19] to mine large databases for efficient searching of association rules. Multi-dimensional constraints are considered in this approach. In addition, this approach also considered user's assign constraint. The results showed that it gave more condensed rules than the Apriori algorithm. The computational time of this approach is less than that of the Apriori algorithm. Though this system provides promising results, it still faces some issues that need to be resolved. After analysing the results, it was found that lots of similar rules were generated so a fuzzy approach was applied to merge those similar rules into one class.

To mine association rules, most researchers focused on ameliorating computational efficiency. To determine the threshold values of support and confidence which are the key factors for the association rule mining task, the researchers proposed a new approach based on the particle swarm optimization technique. Suitable fitness values and their corresponding support and confidence values of the identified swarms are searched through this approach. Their result showed that particle swarm optimization algorithm quickly finds suitable threshold fitness values of itemsets and quality rules are obtained in this way. Users can mine specific rules from a large database by setting support or fitness value. Since this technique free from support constraint, the main problem with this approach is users have no control over mining techniques. Apart from this their result only showed two or three-dimensional rules instead of more dimensional rules that could be interesting for the policy makers of the industry [20].

III. The Idea of Fast Response

If the data tuples contain long itemsets, huge candidate itemsets will be generated, and that reduces the efficiency of a solution. A long itemset enumerates a combinatorial number of shorter, frequent sub-itemsets. For example, a data tuple contains 50 itemsets, {i_1, i_2, i_3, ... ,i_{50}}, enumerates $\binom{50}{1}$ 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 can become too huge for a computer to compute and store if the length of an itemset is long. For each sub-itemset, the Apriori algorithm is used to scan the datasets and calculate the support value of that itemset. However, the Apriori algorithm increases the computational time of the algorithm and decreases the efficiency of it. To overcome this low efficiency of the Apriori algorithm we introduce a new approach that takes into consideration the superset-subset relationships. An itemset **I** is called maximal frequent itemset if the super itemset of **I**, denoted by **Í**, is not frequent such that **I** ⊆ **Í**. Here **Í** is an infrequent itemset based on a user-defined support value.

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

For example, if an itemset **I** = {1,2,3} in set **S** = {1,2,3,4} is frequent, i.e., δ(**I**) ≥ min_supp then all the subsets of **I**, i.e.,{1}, {2}, {3},{1,2}, {1,3}, {2,3} are frequent itemsets, based on the support value defined by the user. The Apriori algorithm scans the datasets for all the subsets of **X** to get the support value. It takes huge computational time if the length of an itemset is long. In our GeneticMax approach, if the generated chromosome is **I** = {1,2,3}, and it satisfies the user-defined support value then it will not test all the subsets of which dramatically reduces the computational time for scanning the datasets.

IV. Preliminaries

In this section, we will introduce some notations and conceptual diagrams that will be used throughout this paper. Initially, we describe datasets through bitmap and bipartite graphs. Then we graphically represent subsets of items using lexicographic trees.

A. Bipartite Graph and Bitmap Representation

If **U** and **V** are two disjoint sets of vertices and **E** is the set of edges which connect the vertices **U** and **V**, then we can represent a bipartite graph as a triple, i.e. **G** = (**U**, **V**, **E**) where **E** ⊆ **U**×**V**.

In a binary matrix, all elements are either 0 or 1. The mapping between binary matrices and datasets of transactions is straight forward. Consider a dataset **D** which consists of m transactions, {t_1, t_2, ..., t_{m-1}, t_m}, corresponding to rows; and n items, {i_1, i_2, ..., i_{n-1}, i_n}, corresponding to columns. The datasets **D** is an m×n matrix, where each entry is defined as a_{ij}. The value of a_{ij} is 1 if transaction t_i contains item i_j; otherwise,

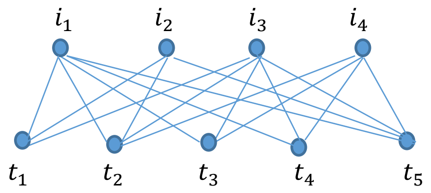

Fig. 2. Bipartite graph representation of the dataset **D.**

	i_1	i_2	i_3	i_4
t_1	1	1	1	0
t_2	1	1	1	1
t_3	1	0	1	1
t_4	1	0	1	1
t_5	1	1	1	1

Fig. 3. Binary matrix representation of the dataset **D**.

it is 0. Now we are mapping each transaction as a set of items from the binary matrices. For example, a dataset **D** which consists of the following transactions t_1, t_2, t_3, t_4, t_5 and items i_1, i_2, i_3, i_4 where t_1 = {i_1, i_2, i_3}, t_2 = {i_1, i_2, i_3, i_4}, t_3 = {i_1, i_3, i_4} and t_5 = {i_1, i_2, i_3, i_4}. Here all the items are different. Fig. 2. and Fig. 3. show the bipartite graph and the binary matrix of the dataset **D**.

From Fig. 2, if we map each transaction by items then the transactions are as follows:

t_1 = {1, 1, 1, 0} = 1110

t_2 = {1, 1, 1, 1} = 1111

t_3 = {1, 0, 1, 1} = 1011

t_4 = {1, 0, 1, 1} = 1011

t_5 = {1, 1, 1, 1} = 1111

B. Lexicographic Tree

Our research problem here is to find the maximal frequent itemsets from large datasets using a GA. Itemset **I** consists of n items, i.e. **I** = {i_1, i_2, i_3, ... ,i_n}. $\mathbf{X}_k$ represents an itemset containing k-items, where k = 1, 2, …, n and $\mathbf{X}_k$ ⊆ **I**. If k = 1, then $\mathbf{X}_k$ contains a 1-itemset, i.e. $\mathbf{X}_k$ = {i_1}. If k = 2, then $\mathbf{X}_k$ contains a 2-itemset, i.e. $\mathbf{X}_k$ = {i_3, i_4}, and so on. An itemset is called frequent if its support value satisfies a user-defined support value, and is denoted by **FI**. An itemset **X** is called maximal frequent itemset if it is frequent, and no superset of **X** satisfies any user defined support value, and is denoted by **MFI**.

In this paper, we will consider the search space that includes all feasible solutions. A lexicographic tree [11, 21] is the search space for GeneticMax. A lexicographic tree maintains lexicographic ordering of items of **I** in a datasets **D**. If item i occurs before item j in a datasets **D**, then it maintains lexicographic ordering, i.e. $i \leq_L j$. If two subsets $\mathbf{S}_1$ and $\mathbf{S}_2$, where $\mathbf{S}_1$ ⊆ $\mathbf{S}_2$ and $\mathbf{S}_1$, $\mathbf{S}_2$ ⊆ **S** then it maintains the following lexicographic order: $\mathbf{S}_1 \leq_L \mathbf{S}_2$. There is no lexicographic ordering relationship between the two subsets $\mathbf{S}_1$ and $\mathbf{S}_2$, If $\mathbf{S}_1$ and $\mathbf{S}_2$ are disjoint subsets.

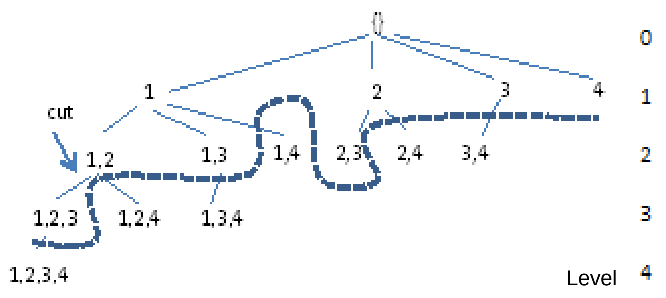

Fig. 4. Lexicographic tree of four items.

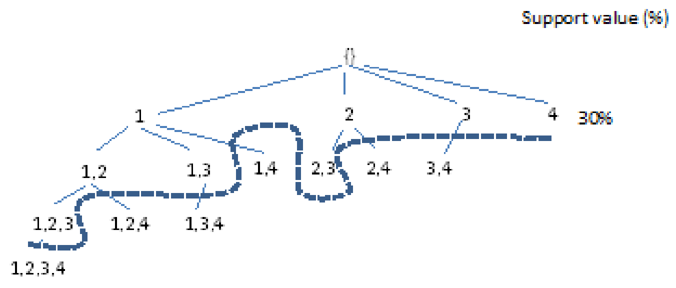

Fig. 5. Lexicographic tree of four items based on a user-defined support value.

Fig. 4 shows an example of a lexicographic tree that considers lexicographic ordering for four items. The root of the tree is an empty set and each *k*-level contains *k*-items. 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. The nodes that are above the cut are frequent itemsets and the elements below this cut are infrequent ones.

For GeneticMax, we introduce a new tree based on user-defined support values. The line is defined by a user-defined support value and the area above the line is referred to as positive area and below negative area. All the nodes in the positive area are frequent whereas those in the negative area are infrequent. In Fig. 5, the nodes within the positive boundary area have a minimum support value of 30%. GeneticMax introduces an array to store the frequent itemsets (called **FI**s). Among the frequent itemsets, the set containing the largest number of items is called the maximal frequent itemset. This maximal frequent itemset (stored in another special array) is called **MFI**. This algorithm searches frequent nodes within a positive area and tries to converge to a solution, i.e., finding maximal frequent itemsets as early as possible. Fig. 5 verifies Lemma 1, where there are 4 items, and it enumerates (2^4-1) = 15 nodes including the root node. With Apriori algorithm, one would test all the nodes at a specific level and generate a candidate set. This candidate set generation needs a long time for finding maximal frequent itemsets. For example, in Fig. 5 it tests the itemsets {1}, {2}, {3}, {4} in level 0 and find that all the itemsets are frequent since these nodes meet the minimum support value. Then it goes to the next level to scan the datasets to get the support values of {1, 2}, {1, 3}, {1, 4}, {2, 3},{2, 4}, {3, 4} and so on. On the next level, it prunes the itemsets {1, 4}, {2, 4}, {3, 4} since these nodes have support values that are less than the user-defined support value. On the other hand, unlike apriori algorithm, with GeneticMax we do not need to test all the nodes, which saves a huge amount of time even when the datasets is very large. For the current example, if the initially generated itemset is {1, 2, 3} then it scans the datasets and calculates the support value. If the support value of the generated itemset {1, 2, 3} is ≥ 30%, then it stores this itemset in a frequent itemset array called `FI_Superset_Member`. In the future it will not scan the datasets for {1}, {2}, {3}, {1, 2}, and {1, 3} since these itemsets are the subsets of the previously generated itemset {1, 2, 3}. If the generated itemsets are {1}, {2}, {3} or {1, 2} then it always checks the array `FI_Superset_Member`. If it finds any superset in `FI_Superset_Member`, GeneticMax will discard theses subsets, which substantially reduces the time for scanning the datasets to calculate the support values correspondingly.

Lemma 3: If **Y** is a superset of an itemset **X**, i.e., **X** ⊆ **Y** and if **Y** is a frequent itemset, then it can be claimed that **X** is a frequent itemset.

For example, {1, 2, 3} is a superset of itemset {1}, {2}, {3}, {1, 2} and {1, 3}. GeneticMax uses the principles of GA and follows the global search mechanism; therefore, a superset could be generated before generating a subset. In this example, if {1, 2, 3} is generated before its subsets (and stored in the array `FI_Superset_Member`), then all other generated subsets will be discarded.

Lemma 4: If **Y** is a superset of an itemset **X**, i.e., **X** ⊆ **Y,** and if **X** is an infrequent itemset, then it can be claimed that **Y** is an infrequent itemset.

For example, if the initially generated chromosome is {1, 4}, and the support value of this itemset is < 30%, then it is stored in a non-frequent itemset array called `NFI`. If the next generated itemset is {1, 3, 4}, the algorithm will check the `NFI` array, and if it finds any subset in this array, GeneticMax will discard itemset {1, 3, 4} for any future calculations.

Lemma 5: If **Z** is a superset of an itemset **X**, **Y**, i.e., **X**, **Y** ⊆ **Z** and if **Z** is an infrequent itemset, then we cannot determine whether **X** or **Y** is an infrequent itemset.

Lemma 5 is slightly different from Lemma 3. With the previous example, if {1, 2, 3} is a frequent itemset then all of its subsets must be frequent, i.e., {1},{2},{3},{1,2},{1,3},{2,3} are all frequent itemsets. But if {1, 2, 3} is an infrequent itemset then we cannot conclude that all of its subset are infrequent. In the above Fig. 5, {1, 4} is an infrequent itemset but its subsets {1} and {4} are frequent itemsets.

Lemma 6: If **Z** is a subset of itemsets **X** and **Y**, i.e. **Z** ⊆ **X**, **Z** ⊆ **Y;** and if **Z** is a frequent itemset, then its supersets **X** and **Y** could be either frequent or infrequent itemsets.

For example, itemset {1} in Fig. 5 is frequent. Although its superset {1, 3} is frequent, its superset {1, 4} is an infrequent itemset.

The main idea of GeneticMax is to find maximal frequent itemsets while converging to a solution as fast as possible. It sub-divides a whole lexicographic tree into two sub-areas based on a user-defined support value. GeneticMax can generate any chromosome in any sub-region. If it finds any superset in a positive boundary area, then it follows Lemma 3 and prunes all of its subsets. But if it finds any subset in a negative boundary area, then it follows Lemma 4 and prunes all of its supersets.

The main advantage of GeneticMax is its ability to converge quickly to a solution, and find all the supersets in a positive boundary area closer to the cut as fast as possible. In the above example, if {1, 2, 3} is generated before all of its subsets ({1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}) and found to be a frequent itemset, then it will discard those subsets (which are also frequent itemsets). If the next generated itemset is {1, 3, 4} it will check the `NFI_Subset_Member` array and does not find any subset there. GeneticMax will scan the datasets for this itemset to find its support value and store it in `NFI`. In the future, all the supersets of {1, 3, 4} will be discarded.

V. 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 features of chromosome coding:

1. It is easy to calculate the support values since GeneticMax uses bitmap representation of the datasets.

2. Generate all the possible nodes. If there are *n* items, it enumerates (2^n-1) itemsets or nodes in the lexicographic tree. If it needs, GeneticMax can generate (2^n-1) nodes in its lifetime.

The length of a chromosome is fixed. If a dataset contains *n* items, then the length of all the generated chromosomes are always *n*, as shown in Fig. 6.

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.

VI. GeneticMax for Efficient MFI Mining

There are five main requirements in developing an efficient maximal frequent itemsets mining algorithm. We need a set of techniques that fulfill these requirements:

1. It will not scan a dataset 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 datasets.

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 datasets.

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 datasets.

V_{item_1}	V_{item_2}	…	V_{item_n}

Fig. 6. Mapping items onto chromosomes $V_{item_{1...n}} \in [0,1]$.

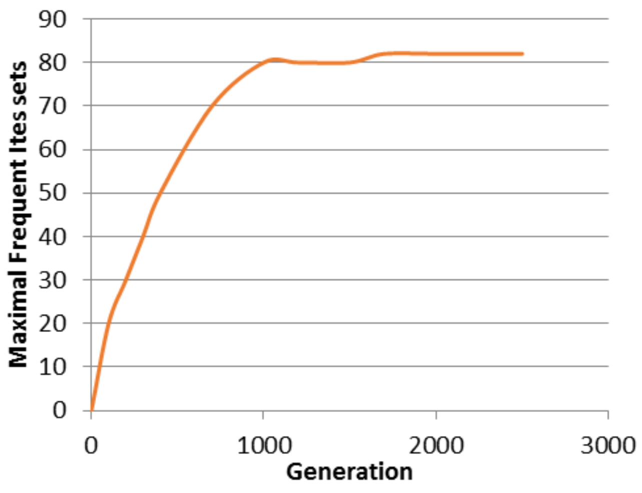

Fig. 7. The number of generations and maximal frequent itemsets.

Apriori algorithm and FP-Tree do not satisfy requirements 1, 2, 3 and 4 respectively. GeneticMax fulfills all the above requirements.

Genetic Algorithms, developed by Holland in 1975, are random search algorithms that generate populations iteratively [22]. This algorithm has been used to find approximate solutions for further optimization. A GA is a search heuristic, considers adaptive methods that are used to solve search as well as optimization problems. This algorithm is inspired by natural selection and the "survival of the fittest" mechanisms which were clearly stated by Charles Darwin in his book "The Origin of Species". The main concept of "survival of the fittest" mechanism is based on the fitness value; only the stronger individuals will survive in a competitive environment. A GA simulates the processes in natural population which are essential for evolution. A large number of researchers worked on GAs [23–26]. Naturally, individuals are competing for their food, shelter, water, clothes and so on. Even members of the same class often compete to attract partners. Those individuals are referred to as strong if they are successful in surviving and attracting a partner. A large number of offspring will be produced by strong individuals. On the other hand poorly performing individuals are referred to as weak and have less probability to produce newer offspring. "Superfit" offspring can be produced by the combination of good attributes from different parents. That is the fitness of this offspring is higher than the fitness of the parents. In this fashion, species becoming more and more well suited in the present environment.

GA plays a vital role in this study. Evolutionary algorithm based techniques are robust and can be used to solve a wide range of problems including those which can be difficult to solve by other methods. It is well known that GA cannot guarantee optimum solutions to any problem but rather it can find “acceptably good” solutions to a problem “quickly”. Existing methods that are working well as a solution for a particular problem, improvement of those methods can be done by hybridizing with GA.

A traditional GA generates an initial population and then computes the fitness value of that population. Two individuals are selected from the old generation and crossover and mutation operators are applied to produce two offsprings. Survivors who have the best fitness values are inserted in the new generation. If the population is converged to a solution, then the algorithm is terminated. In this algorithm, fitness function provides the fitness value of an offspring which is a specification of the offspring.

The main aim of using a GA for this problem is to reach good results by discarding bad solutions during generation of populations [27]. The basic steps of GeneticMax algorithm are as follows:

A. *Procedures of GeneticMax*

There are eight steps in the GeneticMax algorithm. They are listed as follows:

1. Set number of generations.
2. Generate a population.
3. Check the `FI_Superset_Member` and `NFI_Subset_Member` array for superset and subset checking of this generated chromosome.
4. If it finds any superset in `FI_Superset_Member,` or subset in `NFI_Subset_Member,` then go to Step 2.
5. Compute a fitness value of individuals according to their support values in dataset **D**.
6. Perform `FI_Member_Add`, and if any frequent item sets are found then update `FI_Superset_Member`.
7. Perform `NFI_Member_Add`, and if any infrequent itemsets are found then update `NFI_Subset_Member`.
8. Go to Step 3 with newly generated chromosome until it exceeds the generation number set in 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 (Fig. 8) 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 (Fig. 9) 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*

The `Check_Member_for_Item` function (Fig. 10) 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 datasets and sets the itemset in `FI_Superset_Member` or `NFI_Subset_Member` according to the user-defined support value.

```
//Invocation: FI_Member_Add(I_F I_Superset_Member)
1. If any subset of I_F is in I_Superset_Member
2.     Delete the Subset of I_F
3.     Add I_F in FI_Superset_Member
4. Else add I_F in FI_Superset_Member
```

Fig. 8. The `FI_Member_Add` function.

```
//Invocation: NFI_Member_Add(I_1F NFI_Subset_Member)
1. If any superset of I_1F is in NFI_Subset_Member
2.     Delete the Superset of I_1F
3.     Add I_1F in NFI_Subset_Member
4. Else add I_1F in NFI_Subset_Member
```

Fig. 9. The `NFI_Member_Add` function.

```
//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-defined support
       value
7.       Invoke FI_Member_Add
8.     Else invoke NFI_Member_Add
```

Fig. 10. The `Check_Member_for_Item` function.

VII. Experimental Results

The experiments were performed on an Intel(R) Core i5-3210M CPU @2.50GHz, with 4 GB of 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, and Zoo were used to test GeneticMax. Different support values were applied to 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. GeneticMax embeds two main features: i) superset-subset relationship in both positive and negative boundaries in a lexicographic tree for pruning invalid chromosomes, and ii) use of GA which uses a global search mechanism. The purpose of this new approach is to achieve convergence to a solution as fast as possible. A full experiment of GeneticMax on these datasets was conducted, demonstrating GenticMax's ability to yield solutions rapidly by accessing the datasets for a few number of nodes in a lexicographic tree.

As we see from the previous discussions, the Apriori algorithm tests all of the nodes in each level and prunes those nodes that do not satisfy a minimum support value. In GeneticMax, if it generates a chromosome X in any level that satisfies a minimum support value, then all the other subsets of X in any level will be automatically pruned. This approach dramatically reduces the time for accessing a large dataset. This is also true the other way around. If GeneticMax generates a chromosome Y in any level that does not satisfy a minimum support value, then all the other supersets of Y in any level will be automatically pruned.

We tested the algorithm on different datasets such as Tic-Tac-Toe, Zoo, 10000×8 and so on. These datasets were taken from the University of California, Irvine (UCI) machine learning repository (http://archive.ics.uci.edu/ml/datasets.html). From the experimental results as shown in Table 1, we can see that if the number of generations is increased, then it increases the frequent itemsets. For example, for the 10000×8 datasets, 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 10000×8 datasets. This is also true for Tic-Tac-Toe, generation 1200 and 1300 gave the same result that contains the maximal frequent itemsets. So for Tic-Tac-Toe, generation 1200 can be used as a threshold value.

Table 2 shows a comparison between the number of nodes in a lexicographic tree and the number of nodes tested for getting maximal frequent itemsets. For 10000×8, there are 255 itemsets and GeneticMax accessed only 39 itemsets in the main datasets 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 dataset 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 datasets 10000×8 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. 11, the runtime of GeneticMax increases with respect to the generation number of chromosomes. A lower support value that generates more frequent itemsets needs higher runtime whereas a higher support value that generate fewer frequent itemsets needs less computational time, as shown in Fig. 12.

VIII. Conclusion and Summary

In this paper, we proposed a new GA-based approach GeneticMax to mine maximal frequent itemsets in an efficient way. We have conducted thorough experiments on different real datasets. The experimental results demonstrated several advantages of our algorithm.

- It accesses a large datasets for a fewer number of nodes to calculate the support value to find maximal frequent itemsets.
- It shows the power of using an evolutionary algorithm for generating frequent itemsets from a lexicographic tree. The whole dataset is projected onto a lexicographic tree based on a user-defined support value.

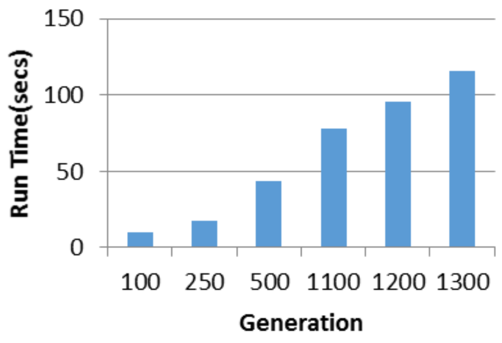

Fig. 11. Run time versus generation for Tic-Tac-Toe.

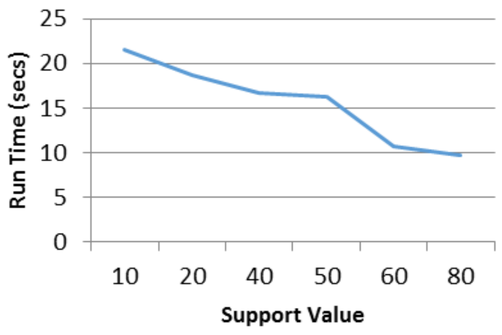

Fig. 12. Run time of GeneticMax using different support values.

TABLE I. THE EXPERIMENTAL RESULTS OF GENETICMAX FOR TWO DIFFERENT DATASETS

Database	Records	Items	Support (%)	Generation	Frequent Itemsets	Time (s)	Remarks
10000×8	10000	8	20	100	9	10.22	
				140	8	21.67	This generation contains **MFI**
				150	8	25.10	
TicTacToe	958	9	16	100	6	10.13	
				250	7	17.53	
				500	10	43.83	
				1100	23	78.20	
				1200	24	95.60	Both Generations provide the same result
				1300	24	115.66	

TABLE II. RESULTS SHOWING THE NUMBER OF TIMES THE DATABASE ACCESSED BY GENETICMAX

Database	Items	Support (%)	No. of nodes in the Lexicographic Tree $(2^{items} - 1)$	No. of nodes tested for getting MFI
10000×8	8	20	255	39
TicTacToe	9	16	511	114
Zoo	17	50	131072	361

- 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.
- The above advantages of GeneticMax increase the scalability of this algorithm.

We have implemented the GeneticMax algorithm and studied its performance. The performance study showed that this algorithm mines different sizes of patterns in real datasets in an efficient way and performs better than other candidate pattern generation and evolutionary based algorithms.

REFERENCES

[1] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.

[2] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” *Proc. 1993 ACM SIGMOD Int. Conf. Manag. Data - SIGMOD ’93*, May 1993, pp. 207–216.

[3] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, “A tree projection algorithm for generation of frequent itemsets,” *Parallel Distrib. Comput. Spec. Issue High Perform. Data Min.*, vol. 61, no. 3, pp. 350–371, 2001.

[4] J. J. Cameron and C. K. Leung, “Mining frequent patterns from precise and uncertain data,” *Comput. Syst.*, vol. 1, no. 1, pp. 3–22, 2011.

[5] M. M. J. Kabir, S. Xu, B. H. O. Kang, and Z. Zhao, “Association rule mining for both frequent and infrequent items using particle swarm optimization algorithm,” *Int. J. Comput. Sci. Eng.*, vol. 6, no. 7, pp. 221–231, 2014.

[6] J. Han, H. Cheng, D. Xin, and X. Yan, “Frequent pattern mining: current status and future directions,” *Data Min. Knowl. Discov.*, vol. 15, no. 1, pp. 55–86, Jan. 2007.

[7] Q. Mei, D. Xin, H. Cheng, J. Han, and C. Zhai, “Generating semantic annotations for frequent patterns with context analysis,” *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD ’06*, 2006, pp. 337–346.

[8] D.-I. Lin and Z. M. Kedem, “Pincer-Search: a new algorithm for discovering the maximal frequent set,” in *Proc. 6th International Conference on Extending Database Technology*, 1998, pp. 103–119.

[9] R. J. Bayardo, “Efficiently mining long patterns from databases,” *ACM SIGMOD*, pp. 85–93, 1998.

[10] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, “Depth first generation of long patterns,” *Proc. Sixth ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. - KDD ’00*, 2000, pp. 108–118.

[11] D. Burdick, M. Calimlim, and J. Gehrke, “MAFIA: a maximal frequent itemset algorithm for transactional databases,” *Proc. 17th Int. Conf. Data Eng.*, 2001, pp. 443–452.

[12] K. Gouda and M. J. Zaki, “GenMax : an efficient algorithm for mining,” *Data Min. Knowl. Discov.*, vol. 11, no. 3, pp. 223–242, 2005.

[13] B. Alataş and E. Akin, “An efficient genetic algorithm for automated mining of both positive and negative quantitative association rules,” *Soft Comput.*, vol. 10, no. 3, pp. 230–237, Apr. 2005.

[14] H. Lu, R. Setiono, and H. Liu, “Effective data mining using neural networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 6, pp. 957–961.

[15] S. Ghosh, A. Nag, D. Biswas, J. P. Singh, S. Biswas, D. Sarkar, and P. P. Sarkar, “Weather data mining using artificial neural network,” *2011 IEEE Recent Adv. Intell. Comput. Syst.*, pp. 192–195, Sep. 2011.

[16] W. Dou, J. Hu, K. Hirasawa, and G. Wu, “Quick response data mining model using genetic algorithm,” *2008 SICE Annu. Conf.*, Aug. 2008, pp. 1214–1219.

[17] A. Salleb-Aouissi, C. Vrain, C. Nortet, X. Kong, and D. Cassard, “QuantMiner for mining quantitative association rules,” *Mach. Learn. Res.*, vol. 14, no. 1, pp. 3153–3157, 2013.

[18] A. Salleb-Aouissi, C. Vrain, and C. Nortet, “QuantMiner : a Genetic Algorithm for mining quantitative association rules,” *Proc. 20th International Joint Conference on Artificial Intelligence*, 2007, pp. 1035–1040.

[19] R. J. Kuo and C. W. Shih, “Association rule mining through the ant colony system for national health insurance research database in Taiwan,” *Comput. Math. with Appl.*, vol. 54, no. 11–12, pp. 1303–1318, Dec. 2007.

[20] R. J. Kuo, C. M. Chao, and Y. T. Chiu, “Application of particle swarm optimization to association rule mining,” *Appl. Soft Comput.*, vol. 11, no. 1, pp. 326–336, 2011.

[21] J. P. Huang, C. T. Yang, and C. H. Fu, “A Genetic Algorithm based searching of maximal frequent itemsets,” in *International Conference on Artificial Intelligence*, 2004, pp. 548–554.

[22] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[23] D. Beasley, D. R. Bull, and R. R. Martin, “An overview of Genetic Algorithms : Part 1 , Fundamentals,” *Univ. Comput.*, vol. 15, no. 2, pp. 58–69, 1993.

[24] M. Srinivas and L. M. Patnaik, “Genetic Algorithms: a survey,” *Computer (Long. Beach. Calif).*, vol. 27, no. 6, pp. 17–26, 1994.

[25] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[26] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer, 1992.

[27] A. A. Freitas, “A survey of evolutionary algorithms for data mining and knowledge discovery,” in *Advances in Evolutionary Computing*, Springer Berlin-Heidelberg, 2003, pp. 819–845.