A Genetic Algorithm Based Searching of Maximal Frequent Itemsets

Jen-peng Huang
Southern Taiwan Univ. of Technology, Tainan, Taiwan

Che-Tsung Yang
Southern Taiwan Univ. of Technology, Tainan, Taiwan

Chih-Hsiung Fu
Computer & Network Center Bureau of Education, Tainan, Taiwan

Abstract

We present a novel approach to generate maximal frequent itemsets using principles of genetic algorithm. The new approach, we called GAMax is an implementation using genetic algorithm to find the maximal frequent itemsets in transaction database. The objective is to identify maximal frequent itemsets in lexicographic tree, needless to enumerate all the frequent itemsets level by level. In natures of GAs, the significant contribution of GAMax is: providing a general algorithm for generating frequent itemsets that is scale independent to the size of database. We also expect GAMax to open new visions in combining these two flourishing domains of research.

***Keywords**: Genetic Algorithm, Association Rules, Maximal Frequent Itemsets, Data Mining.*

1. Introduction:

Association rules mining including two stages: first is to find the frequent itemsets and the second is to generate interesting rules. The task of stage 1: finding frequent itemsets is the most time-consuming and becoming the most concentrated issue in performance improvements. Several approaches [1, 2, 3, 4, 5, 6 and 7] proposed to find the exact maximal frequent itemsets and have being attractive in the recent.

Genetic algorithms are general solutions of optimization problems. Through well-defined fitness functions and operators, GA manipulates the specific meaningful codes in search space iteratively to determine the optimal solution. We transit the problems of generate the maximal frequent itemsets of association rules mining into optimization problems and solve using genetic algorithm. This paper provides an alternative for determining maximal frequent itemsets which time consumes independently to the size of transaction database we refer to "***scale independent***". Furthermore, it also combines two flourishing domains of research naturally to provide a new visions.

This paper is organized as follows. In section 2, we make a brief review of algorithms for generating the maximal frequent itemsets. Introducing the principles of genetic algorithms and implementations of data mining are also included. In section 3, descriptions of *GAMax* are introduced. Section 4 concludes a summary. Finally, section 5 gives some directions of future research and development. Related references are shown in section 6.

2. Related works

We are going to introduce some maximal frequent itemsets algorithms and principle concepts of genetic algorithm briefly in this session. The implementations of using genetic algorithms in data mining are another topic of this session.

2.1 Maximal Frequent Itemsets

The follows are algorithms for generating maximal frequent itemsets.

Pincer-Search [1] is a bi-direction traversal method with both top-down and bottom-up. When any frequent or infrequent itemsets is determined, pruning methods applied using the following two properties. 1: All the subsets of frequent itemsets must be frequent and needless to validate. 2: All the supersets of infrequent must be infrequent and neither to validate.

MaxMiner [2] performs a bread-first traversal and look-ahead pruning. It uses item re-ordering heuristic to increase the effectiveness of superset frequency pruning.

DepthProject [3] employs a transaction projection mechanism for counting the support of itemsets. It finds long itemsets relies on a depth-first search method in a lexicographic tree of items with look-ahead pruning. To reduce the size of search space, *DepthProject* trims infrequent items out of each node's tail. *DepthProject* would require post-pruning to eliminate non-maximal patterns.

Mafia [4] integrates a depth-first traversal of itemset lattice with pruning strategies to remove non-maximal itemsets efficiently.

Algorithm	Traversal Method	Pruning	Database Representation
Pincer-Search	Bottom-up + Top-down	Downward closure	Horizontal
MaxMiner	BFS	Look-ahead	Horizontal
DepthProject	DFS	Look-ahead	Vertical
GenMax	Backtrack search based	Pruning Optimization	Vertical
MAFIA	DFS	Pruning Strategies	Vertical bit vector
MinMax	DFS + Backtrack search	Downward closure	Vertical Tid

Table 1: Comparison between algorithms of generating MFI

Mafia uses vertical bit-vector data format, compression and projection of bitmaps to achieve better performance. The obtained MFI through *Mafia* also need post-pruning.

GenMax [5] is a backtrack-based search algorithm with a number of optimizations of pruning. *GenMax* is a method for enumerate the exact set of maximal patterns.

MinMax [6] is based on depth-first traversal and iterative. It combines a vertical tidset representation of data with pruning mechanisms. It backtracks to the proper ancestor directly needless level by level.

2.2 Principles of Genetic Algorithm

Genetic algorithm was proposed by John Holland in 1975. John Holland considered problems from genotype view and finding the optimal solutions by simulating the competence of individuals. Generally speaking, genetic algorithms are simulations of evolution. In most cases, they can be considered as probabilistic optimization methods which are based on the principles of evolution. The GA encodes potential solutions as strings of 0 and 1 called chromosomes. Each chromosome has a specific fitness value from fitness function. With a set of initial chromosomes, the GA will reserve the more promising individuals and converge to an optimal solution.

Genetic algorithm consists of four basic components:

Selection: Mechanism for selecting individuals for reproduction according to their fitness that is objective function value.

Crossover: Method of merging the genetic information of two individuals; if the coding is chosen properly, good parents produces good children.

Mutation: In real evolution, the genetic material can by changed randomly by erroneous reproduction or other deformations of genes, e.g. by gamma radiation. In genetic algorithms, mutation can be realized as a random deformation of the strings with a certain probability. The positive effect is preservation of genetic diversity and, as an effect, that local maxima can be avoided.

Sampling: Procedure which computes a new generation from the previous one and its offspring.

2.3 Genetic-based Mining

The genetic algorithm can be applied to any kind of optimization problems. It has been proven to be a general searching approach with robustness and scalability in practice. It is the reason why we attempt to find the maximal frequent itemsets using GAs. The main instinct character of *GAMax* is avoiding the time increasing tremendously due to the size increasing of transaction database.

In [7], it presents an approach and results using genetic algorithm to search a large set of transaction database. The goal of this implementation focuses on determining the four out of 100 possible items that are most appeared together in transaction database. Through experimentation with a variety of fitness functions, crossover operators and mutation operators, the results provided better understandings insight the effects of applying GAs to association rules mining.

3. Description of GAMax

It is necessary to re-consider problems of generating maximal frequent itemsets by means of the points of solving optimization problems previously. We state the problem in the way of optimization problems and then illustrate the principles of *GAMax*.

3.1 Problem statement

The problem is addressed consists of using genetic algorithm to search a transaction database to find the set of maximal frequent itemsets. *I* represents the items defines as: *I*= {1, 2… N}. Itemset *X* containing k items and belongs to itemset *I* is called by *k-itemsets*. *X* is a frequent itemsets, denote by *FI*, if the support of *X* is greater than the specific threshold. If *X* is frequent and no superset of *X* is frequent, we say *X* is a maximal frequent itemsets and the set of all frequent itemsets is denoted by *MFI*.

In solving a problem, we are usually looking for some solutions which will be the best among others. The space of all feasible solutions is called by search space. The search space of *GAMax* is a lexicographical tree [2]. The lexicographical tree is an abstract representation of the large itemsets with respect to an existing ordering. It is defined in the following way:

(1)All the nodes in the tree are corresponding to an itemset.

(2)Let I = {i_1, i_2, i_k} be an itemset, where i_1, i_2, i_k are in lexicographical order. The parent of I is itemset {i_1, i_2… i_{k-1}.}.

A searching tree in lexicographical order corresponding to the transaction database has the following properties:

(1) It is a left leaf most tree.

(2) The nodes that are closer from the root have the higher support than the farer ones.

The problem of finding maximal frequent itemsets can be viewed as to determine a specific cut on the lexicographical tree. All the nodes above the cut are frequent and are infrequent which are below [4], shown in Figure 1.

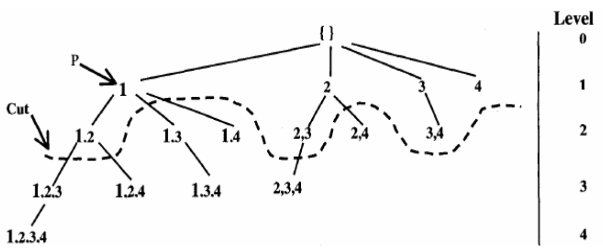

Figure 1: Lexicographical tree of 4 items

To figure out our search space in *GAMax* clearly, we construct a new tree with ***support oriented***. On the *support oriented* tree, the length from the root represents its support in transaction database. The more far away from the root, the smaller support value it is. Each node represents an itemset and may be a frequent one depends on its support. We define a circle which radius is equal to the minimum support and refer to "*minimum border*". The area which is enclosed by minimum support which referred to "*positive border*" and concluding all the frequent itemsets in it. Search all the nearest nodes within the *positive border* by branches are the *MFIs* we intend to find, for examples: (1, 2), (2, 3), (3, 4), (4), shown as Figure2.

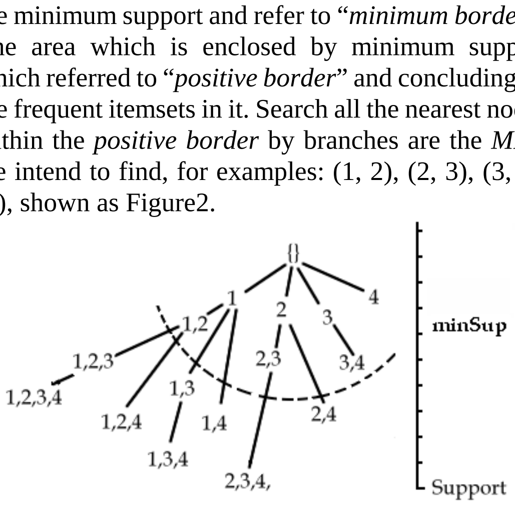

Figure 2: Searching tree of 4 items (support oriented)

It is supposed that there exists a maximal frequent itemset on each branch of tree. If it does, we can find the most many maximal frequent itemsets which the cardinality of *MFI* is $2^{(n-1)}$. Otherwise, the numbers of *MFIs* will less than $2^{(n-1)}$. Since a variety of numbers of *MFIs* in scenarios, we define ***Branch Equivalence*** (***BE***) to determine when to terminate the process of GAMax which will be described later.

3.2 Chromosome coding

The coding scheme of *GAMax* is determined under the following considerations:

1. It is able to express all possible solutions.
2. No illegal chromosomes will be generated after crossover.
3. It is easy to investigate supports of chromosomes and compute fitness values.

d_1	d_2	…	d_i	…	d_n
Item 1	Item 2		Item ***i***		Item ***n***

($d_i \in [0, 1], i \in [0, n]$)

Figure 3: Chromosome coding scheme

The length of chromosome is determined by the numbers of items. Each chromosome in *GAMax* represents an itemset that exists in transaction database probably. It needs an n-bit string to represent all the possible itemsets while there are n items in the transaction database. Each node on the lexicography tree got its unique code of chromosome.

3.3 Fitness function

The fitness function of *GAMax* is intended to reserve the individual which is the most close to the *minimum border* from those which are enclosed in the *positive border* and to eliminate individuals that are far away from the *minimum border* or out of the *positive border*. In heuristic, the fitness function should be illustrated as figure 3.

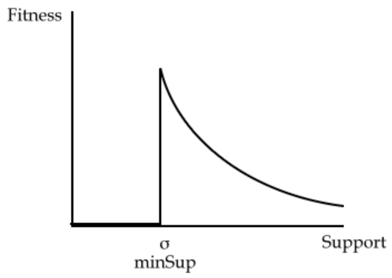

Figure 4: Fitness function

According to this concept, we define the fitness function of *GAMax* as follows:

$$F = \sum_{i=1}^{n} \frac{1}{\sigma i - \sigma} \quad while \quad \sigma i > \sigma$$
$$F = \alpha \quad while \ \sigma i = \sigma$$
$$F = 0 \quad while \ \sigma i < \sigma$$

The fitness values are expected to be lower when the difference between potential itemsets and minimum support are great and yield to higher probabilities of elimination. The fitness values of itemsets not pass the threshold are equal to zero and have to be eliminated. We assign fitness a specific value α when the supports of itemsets are equal to minimum support.

In other alternatives, the fitness functions are based on the consideration: The more far away from the *minimum border* should yield lower fitness values to fasten the speed of evolutions. The other two types of fitness functions are designed to express more emphasis on the difference between itemsets and minimum support. We introduce a *weight* that derives from the support ratio of itemsets and minimum support and the third fitness function utilizes the weight as power, as follows:

3.4 Operators of GAMax

GAMax adopts the *roulette wheel selection* that reproduces chromosomes according to their proportion of fitness values respectively. Operation of crossover is cutting two strings at a randomly chosen position and swapping the two tails, which refers to "one-point crossover".

Mutation of chromosomes happens according the specific probability we defined. Under this specific probability, some bits of individuals of new generations change randomly.

$$F = \sum_{i=1}^{n} \left(\frac{1}{\sigma i - \sigma}\right)^{w} \quad while \quad \sigma i > \sigma$$

$$F = \sum_{i=1}^{n} \left(\frac{1}{\sigma i - \sigma}\right) * \mathrm{w} \quad while \quad \sigma i > \sigma$$

3.5 Branch Equivalence

The branch equivalence is defined as follow:

$$BE=2^{(d-1)}$$

d represents the degree of nodes. The degree of leaf nodes are equal to zero, we define the ***BE*** value of leaf nodes as zero.

Branch equivalence is a specific defined measurement to express the weight of a node in the searching space. By means of ***BE*** value, ***GAMax*** can monitor the completeness of task of searching ***MFIs***.

Each node has its *branch Equivalence* value that depends on the numbers of branches that it encloses respectively. We utilize the summation of branch equivalence of the found *MFIs* and *NFIs* as the condition of terminate. The *NFIs* is the collection of the shortest non-frequent itemsets, we utilize to determine the termination of ***GAMAx*** in cooperate with the *MFIs*.

3.6 Condition of terminal

It is supposed that each branch on the tree with n items can find a maximal frequent itemset and total numbers will be $2^{(n-1)}$ at most. This is the extremely scenario when the *positive border* is maximal, a example with 4 items shown as figure 4.

In this situation, we got the most many of ***MFIs*** and the total ***BE*** value of all ***MFIs*** is equal to $2^{(n-1)}$ also the most many too.

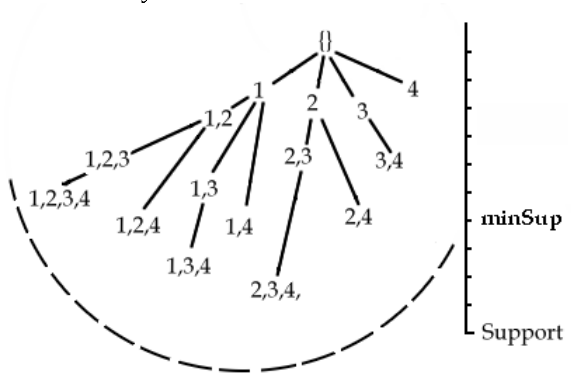

Figure 5: Scenario of the largest *positive border*

MFI	Coding	*BE_MFI*
{1,2,3,4}	1111	1
{1,2,4}	1101	1
{1,3,4}	1011	1
{1,4}	1001	1
{2,3,4}	0111	1
{2,4}	0101	1
{3,4}	0011	1
{4}	0001	1
BE		8

Table 2: *BE* value in the largest *positive border*

The other extremely scenario with the largest *negative border* is shown as figure 5.

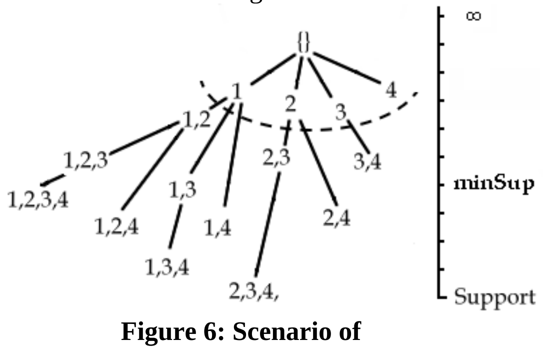

Figure 6: Scenario of the largest *negative border*

NFI	Coding	*BE_NFI*
{1}	1000	4
{2}	0100	2
{3}	0010	1
{4}	0001	1
BE		8

Table 3: *BE* value in the largest *negative border*

The general scenarios are most likely as which shown in figure 6. The general situations are mixed scenarios between the largest *positive border* and the largest *negative border*.

In general situations, ***GAMax*** sums the *BE* values of *MFIs* and *NFIs* and terminated while the summation value of all *MFIs* and *NFIs* reaches to $2^{(n-1)}$.

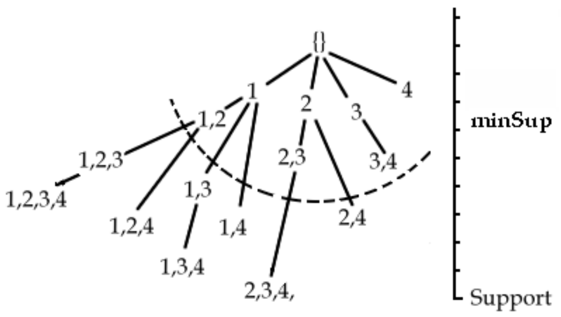

Figure 7: General scenario of *GAMax*

MFI	Coding	*BE_MFI*
(1,2)	1100	2
(2,3)	0110	1
(3,4)	0011	1
(4)	0001	1
BE		5

Table 4: *BE_MFI* value

NFI	Coding	*BE_NFI*
(1,3)	1010	1
(1,4)	1001	1
(2,4)	0001	1
BE		3

Table 5: *BE_NFI* value

3.7 Maintenance of MFIs and NFIs

If there is any frequent or infrequent itemsets has been confirmed in the process *GAMax*, it utilizes the following components to validate and append to *MFIs* or *NFIs*. The main concerns of these components are to avoid appending any redundant itemsets to *MFIs* or *NFIs*. It is necessary to check whether exist parent or child relations between current itemset and the existing sets. ***GAMax*** has to keep the accurate sets of *MFIs* and *NFIs* for determinate whether to terminate or not.

```
MFI_Add (Current frequent F, MFI) {
//Check frequent itemsets in the MFI
if (F or any superset of F is in the MFI)
  Discard F
else if (any subset of F is in the MFI)
    Delete the subset;
    Add F to MFI;
  else
    Add F to MFI;
```

Figure 8: Algorithm of *MFI_Add*

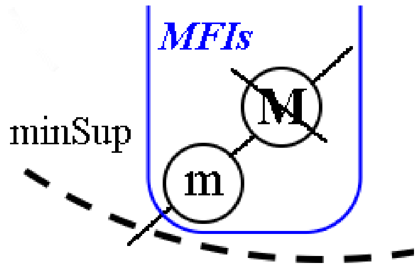

Figure 9: Conceptual diagram of *MFI_Add*

```
NFI_Add (Current infrequent NF, NFI) {
//Check infrequent itemsets in the NFI
if (NF or any subset of NF is in the NFI)
  Discard NF
else if (any superset of NF is in the NFI)
          Delete the superset;
          Add NF to NFI;
        else
          Add NF to NFI;
      }
```

Figure 10: Algorithm of *NFI_Add*

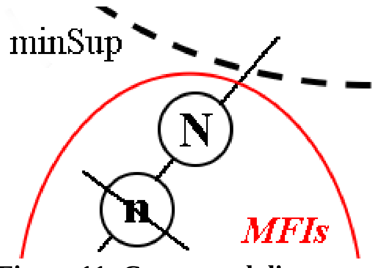

Figure 11: Conceptual diagram of *NFI_Add*

```
Cross _Validate (Current infrequent F, NFI) {
//Check infrequent itemsets in the NFI
if (NF or any subset of NF is in the NFI)
  Discard NF
else if (any superset of NF is in the NFI)
    Delete the superset;
    Add NF to NFI;
  else
    Add NF to NFI;
}
```

Figure 12: Algorithm of *Cross_Validate*

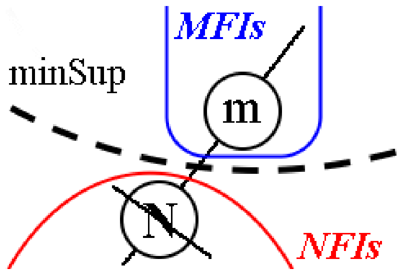

Figure 13: Conceptual diagram of *Cross_Validate*

3.8 Process of GAMax

Step 1: Set the initial value of *BE* to zero.
Step 2: Generate the initial population of GAMax.
Step 3: Terminate and output the set of *MFIs* if the total value of *BE* reaches 2 $^{(n-1)}$.
Step 4: Compute fitness value of individuals according their support in transaction database.
Step 5: Perform *NFI_Add* while any infrequent itemset is found and update the value of *BE_NFI* correspondingly.
Step 6: Perform *MFI_Add* while any evolution converged and update the value of *BE_MFI* correspondingly.
Step 7: Validate and remove redundant *NFI* while any *MFI* has been appended.
Step 8: Go to step 4 and into the next iteration of GAMax with the newly generated chromosomes.

4. Conclusions

There are some familiarities between problems of generating the maximal frequent itemsets and problems of optimization. Those familiarities are why we propose this novel approach of generating the maximal frequent itemsets based on GA. Though some researches have been proposed their implementations of GAs to data mining in recent, the algorithm to identify the exact maximal frequent itemsets has not been proposed before *GAMax*. Since *GAMax* is an algorithm based on GA, it is nature to provide a general and robust solution. It also is a method of scale independent to the size of database and is achieve improvement on performance especially when the databases are large.

5. Future works

Experiment implementations to provide evidence of real data are encouraged and proceeding. The comparisons between fitness functions are expected and helpful to insight the *GAMax*. It is a worthy effort to achieve better understandings of *GAMax*.

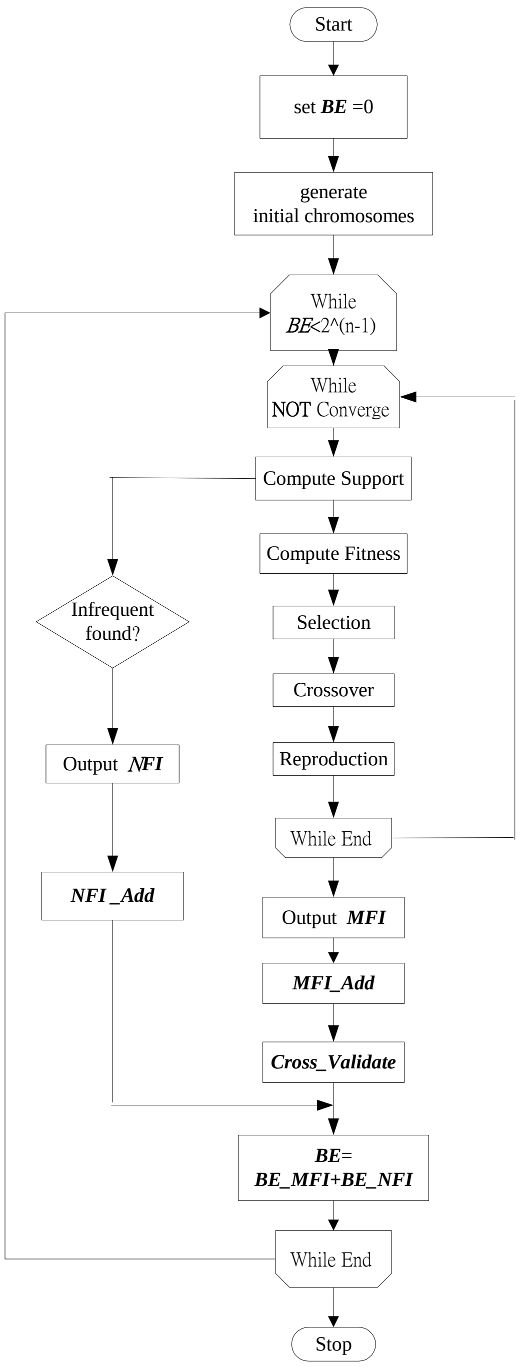

Figure 14: Process diagram of GAMax

6. References

[1] D-I. Lin, Z. M. Kedem. , Pincer-search: A new algorithm for discovering the maximum frequent set. In 6th Intl. Conf. Extending Database Technology, March 1998.
[2] R. J. Bayardo. Efficiently mining long patterns from databases. In ACM SIGMOD Conf., June 1998.
[3] R. Agrawal, C. Aggarwal, and V. Prasad. Depth First Generation of Long Patterns. In ACM SIGKDD Conf., Aug. 2000.

[4] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: a maximal frequent itemset algorithm for transactional databases. In Intl. Conf. on Data Engineering, Apr. 2001.
[5] K. Gouda, M. J. Zaki. Efficiently Mining Maximal Frequent Itemsets. In 1st IEEE Intl. Conf. On data mining, Nov. 2001
[6] Hui Wang, Qinghua Chuanxiang Ma, and Kenli Li., A maximal Frequent Itemset Algorithm.
[7] Charles L.Karr, L. Michael Freeman. Industrial applications of genetic algorithms p160-194. Boca Raton, FL,CRC Press 1999.