Applications of Genetic Algorithms

MARIUS-CONSTANTIN POPESCU[1] LILIANA POPESCU[2] NIKOS MASTORAKIS[3]
[1]Faculty of Electromechanical and Environmental Engineering, University of Craiova
Decebal Bv, No.107, 200440, Craiova
[2]"Elena Cuza" College of Craiova
ROMANIA
[3]Technical University of Sofia
BULGARIA
mrpopescu@em.ucv.ro lpopi2001@yahoo.com mastor@wses.org

Abstract: In this paper were presented the main directions of genetic algorithms. There is a large class of interesting problems that have not yet been developed fast algorithms. Many of these problems are problems which occur frequently optimized in applications. Genetic algorithms are part of Heuristic algorithms, applying them successfully if problems do not admit polynomial-time algorithms. Genetic algorithms, as the name suggests, are inspired from nature, specifically of the way through genetic recombination improves a species.

Key-Words: Genetic operators, Genetic programming, Objective function.

1 Introduction

Giving is a problem poorly optimized is always possible to find an efficient algorithm whose solution is almost optimal [2], [3], [9], [11]. For some stupid problems we can use optimized algorithms probabilistic. These algorithms do not guarantee optimal value, but the elections random enough weaknesses of errors can be made so that we can overcome them. There are many practical problems for such optimized algorithms for a high quality became available. In general, any abstract process to be accomplished can be thought of as a problem-solving, which, in turn, may be perceived as a search space with potential solutions. How are we looking for the best solutions, we can look at this task as a process optimized. For small spaces, classical methods are sufficient executive, large spaces for special techniques of artificial intelligence should be taken into account.

Genetic algorithms are among these techniques, they are stochastic algorithms whose search methods molds some natural phenomena [1]. The idea behind genetic algorithms is to do what nature does. Some fundamental principles of genetics are borrowed and used artificially to build search algorithms that are robust and require minimum information about the problem. Genetic algorithms were made using the process of adaptation. They operate, in particular, with binary strings and use a recombination operator and a mutation. Mutation by changing a (gene) from a chromosome, and by crossing change genetic material between two parents, if parents are represented by strings of five bits, for example (0, 0, 0, 0, 0) and (1, 1, 1, 1, 1), crossing two vectors can result in descendants (0, 0, 1, 1, 1) and (1, 1, 0, 0, 0) (this is an example of such called cross-point with a notch).

The fitness of an individual is assigned in proportion to the value function corresponding to the individual criteria, individuals are selected for the next generation on the basis of their fitness. We stated previously that genetic algorithms work with strings of bits representing the parameters and not the parameters them selves. After created a new series (a new solution) through the genetic operators must evaluate it. In most cases, the fitness is just the criterion function for that solution. If our objective is to minimize the criterion, then we say that a solution is better than another, if the fitness of the two is greater [4], [5].

2 Structure of a Genetic Algorithm

In 1990 Koza proposed such a evolution system, genetic programming, to search for the best computer program to solve a particular problem. The program structure development is shown in Fig.1a.

Steps to be taken in such an algorithm can be described as Fig.1b [7]. The idea of genetic algorithms is to represent possible solutions of the problem in the form of chromosomes, and work at each step with a fixed number of chromosomes that

form a population. In the above algorithm, P is a population of chromosomes representing the problem solution found in step respectively, and P' is an intermediate population, generated from P by specific genetic methods (cross between chromosome mutations and spontaneous). Thus, attempts to improve the population of chromosomes within the time available, the meaning of closeness as much as the optimal solution.

```
procedure evolutionsry algorithm
t←0
Creation P(t)
Asseement P(t)
   While not subjet to termination
        t←t+1
       selection P(t) from P(t+1)
        change P(t)
       assessment P(t)
   end while
end procedure
```

a)

```
P=InitPopulation();
While not exhausted during the
execution
{
  Rate(P);
    P'=SelectParents(P);
    Recombine(P');
    Mutations(P');
    P=P';
}
Show(P);
```

b)

Fig.1:Structure of a genetic algorithm after [8] and [15]

Generation 0 is chosen completely randomly, and the remaining operations and use them to generate random numbers. Consequently, the result of execution of such algorithm will also depend on chance, and, moreover, will be run at each other.

To better explain how the algorithm works, we choose a concrete problem, namely, "Determination of a maximum function $f(x)$ on interval $[a, b]$".
This problem has the advantage that allows us to evaluate whether the algorithm easily leads us to the solution or not, although there is a significant example for using genetic algorithms [10]. Clearly, for the best results, you should consider as many values for the variable x in the interval $[a, b]$. I noted with no number of such values. All values that we choose will be quantified in the form of chromosomes. Chromosome is a sequence of k binary positions, each position being a gene. Therefore we have no chromosome with k genes each.
First step is to choose randomly the NR chromosome, generating one sequence of random gene k (values of 0 or 1). To convert a chromosome into a real variable in $[a, b]$, a division of the field in the 2^k intervals and assigned to chromosome $a \leftrightarrow 00...000$ and $b \leftrightarrow 111...111$ chromosome, the rest being distributed proportionally.

To obtain the solution to first consider the chromosome as NR: $c_1,c_2 \ldots,c_{NR}$. To assess the population of chromosomes, will calculate the following values:

- First, objective function $v_i = f(c_i)$, thereby convert each chromosome into a real value, namely the function whose maximum ill want point in the chromosome but c_i;
- Calculate the amount the objective function

$$S = \sum_{i=1}^{NR} v_i \; ; \tag{1}$$

- For each chromosome we calculate the probability of selection

$$p_i = \frac{v_i}{S}; i \in \{1,2...NR\}; \tag{2}$$

- For each chromosome is calculated cumulative probability of selection

$$q_j = \sum_{i=1}^{j} p_i \; ; \tag{3}$$

the observation series $q_1,q_2 \ldots q_{NR}$ will be increasing, the last value being 1.
As c_{i+1} but contains a value for which to obtain a higher value for the objective function, both with difference between q_{i+1} and q_i will be higher. Thus, row cumulative probability selection is a division of the interval [0.1].
To create an intermediate population of chromosomes, select NR uniform random numbers in the interval (0, 1]. If a number is located in the $(q_i, q_{i+1}]$, but then chromosome c_{i+1} is selected. It can be seen that the probability that a chromosome is selected to be much higher as the $(q_i, q_{i+1}]$ is greater. I showed in the previous paragraph as the length of this interval is much higher as the objective function for the chromosome is larger. Consequently, there is a greater probability that a chromosome "best" to be selected, but does not warrant its selection. In addition, a chromosome can be selected several times in the intermediate population [14].

The next step is mating between chromosomal populations of intermediate. Here is a problem, namely how to choose chromosome pairing. First, if you choose none, there is the possibility to obtain a solution better than the present, and if you choose all too much risk destroying the entire population of chromosomes, so that after pairing time resulting population may be better or worst.

In the below read from a computer keyboard, and for each chromosome generate a random number in the interval (0, 1]. If the number is smaller than p_c, that will be subjected to chromosome pairing time. An acceptable value for pc is 0.1 (10% of chromosome pairing time will be subjected).

Technical crosses is the following:

- Cross first chromosome selected for mating with the second, third fourth etc (if selected for mating to an odd number of chromosomes, is the last drop);
- The crossing is in exchange between the two genes at chromosome, where t is chosen randomly in $(0, k)$.

After crossing two chromosomes are obtained us:

- The first new chromosome will contain the first gene of the first t chromosome old and last k-t genes of the second chromosome old;
- The second new chromosome will contain the first gene t of the second chromosome old and the last k - t genes of the first chromosome.

Finally, the intermediate population of chromosome is subjected to simple mutations. For this, we read from the keyboard probability of occurrence of mutations simple p_s, which should have a small value (close to 0). For each gene of each chromosome is randomly choose a number between (0,1], and particularly if the number is less than p_s, gene content change of 0 or 1 in reverse. Following these operations, to obtain a new population of chromosomes and returned to the stage of assessing the population.

The algorithm runs in limited time available, which is read as a parameter from the keyboard. Note huge similarity between genetic algorithms and everyday life.

Although chromosomal values have higher chances to reach a new population, there is the possibility that some of them to lose. Important chromosome is not so, but the population of chromosomes. It must evolve.

Following implementation of a program for different functions, the following results are obtained using parameters NR=5000, k=30, p_c=10%, p_s=1%, t=1 seconds (time is not relevant, it strongly depends on the speed system was available).

Tab. 1: The results by applying the algorithm.

Function	Inter - val	(x_{max},y_{max}) obtained		(x_{max},y_{max}) analytical		Err-or
$\lvert x^3 - x\rvert$	(0,1)	0.5773	0.3849	0.5774	0.3849	$<10^{-4}$
$\lvert\sin x\rvert$	(0,1.6)	1.57018	1	1.5707	1	$<10^{-4}$
$\frac{\lvert\ln x\rvert}{x}$	(1,3)	2.71814	0.367879	2.71828	0.367879	$<10^{-4}$

We illustrate the working of genetic algorithms using a simple problem: designing a box of cans. We consider a cubic box of canned food, with only two parameters: the diameter d and height h (obviously, can be considered and other parameters such as thickness, material properties, shape, but just enough are the two parameters to illustrate with genetic algorithms). To believe that this box of canned food should have a volume of at least 300 ml and the project objective is to minimize the cost of materials used in the manufacture of cans.

We formulate our problem as follows:

$$f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right), \tag{4}$$

to minimize the function where c is preserved material cost per cm^2, and the expression in brackets is the area preserved. Function f is called and the criterion function (or objective function). Have met and provided that the box is at least 300 ml and we will make it so:

$$g(d,h) = \frac{\pi d^2}{4} \geq 300. \tag{5}$$

The parameters d and h can vary between certain limits. The first step in using a genetic algorithm is to establish a codification of the problem. Binary encoding is the most common techniques of coding, it is easy to handle and gives robustness problem. Binary representation can encode almost any situation, and operators do not include knowledge of the field problem. It's why a genetic algorithm can be applied to very different problems. If binary encoding, each value is mean by a string of specified length which contains the values 0 and 1. In some situations it is necessary to use encoding "natural" problem, instead of binary representation. An example would be the natural coding actual coding, which uses real numbers for representation. To use genetic algorithms to find optimal values for parameters d and h, which satisfies the condition in the form and function g to minimize the function f,

we will first need to represent the binary strings in the parameters (we use therefore a binary encoding of the problem).
Genetic algorithms require not only values the whole of a given interval, in general, we can choose any real value or by changing the length binary string.

3 Genetic Operators

We further describe using genetic operators, usually in a genetic algorithm.

Selection. An important role in a genetic algorithm is occupied by the selection operator. The operator decides wich of the population individuals will participate in forming the next population. The purpose of selection is to provide more reproductive opportunities to the most performant of the individuals in a given population. Through selection we aim to maximize individual performance.we will briefly present the most important selection mechanisms in the following

a) Proportional selection. In proportional selection case, the likelihood of selecting an individual depends on the performance thereof. Suppose you have a lot of chromosomes $x_1,x_2,\ldots,x_n$. For each chromosome we calculate x_i performance to $f(x_i)$. Should be provided that $f(x_i)\geq 0$. The performance sum for all chromosomes of the population will be the total performance and we will note it with F.

b) Selection based ordering. This selection is to calculate (for each generation) the fitness function values and to arrange the individuals in a descending order of these values. It will assign each i individual a selection probability pi that depends on its rank in series. Probabilities now depend only on the chromosome position. The most promising individual has probability is 1.

c) Selection through contest. Selection through competition or selection lists are based on direct comparison of two chromosomes and selecting the best performing. The operations involved are:
• are chosen at random two chromosomes;
• calculating performance chromosome selected;
• best performing chromosome is selected (copied in the population over which interim apply genetic operators).

Other mechanisms for selecting another type of selection is elitist selection. In this case, every generation we keep the most promising or the most promising individuals. Another idea would be that every generation, to be replaced only a small part of the population.

The reproduction operator. Operator reproductive role is to maintain the promising solutions of the population and to eliminate the less promising, keeping constant the population size. This is done as follows:
• identifying promising solutions of the population;
• to create multiple copies of promising solutions;
• be deleted less promising solutions of the population so that multiple copies of promising solutions can be placed in the population. There are several ways to do this. The most common methods are proportional selection, the tournament selection and selection by order.it is easily seen that the promising solutions have more than one copy in the intermediate population.

The crossing operator. The meeting is applied on individuals in the population between. In our example, will be applied to the binary representation of the six elements that we have people in between. The cross acts in the following way: they are two randomly chosen individuals from intermediate population (which is also called and cross the pool) and some portions of the two individuals are interchangeability.

The operator mimics natural interchromosome crossing. It is used by operators of cross type (2,2), ie, two parents give birth to two descendants. Crosses made an exchange of information between the two parents. Descendants produced by crossing will have characteristics of both parents. Given the importance of crossing were proposed several models of interbreeding. We enumerate here some of those used when binary coding.

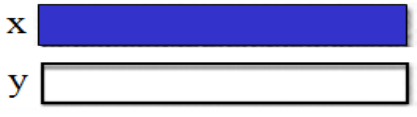
Crossing point with a cleft. R be the length of chromosomes. A notch point is an integer $k\in\{1,2,...,r\text{-}1\}$. The number k indicates the position of the chromosome sequence where chromosomal breaks that are produced segments to recombine with other segments from other chromosomes. We consider two chromosomes:

$$x=x_1x_2\ldots x_k\,x_{k+1}\ldots x_r \quad \text{and} \quad y=y_1y_2\ldots y_ky_{k+1}\ldots y_r.$$

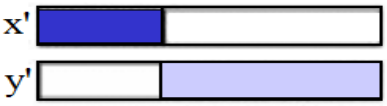
Following recombinations change chromosomes between the two sequences in the right notch point k chromosomes will be:

$$x'=x_1x_2\ldots x_k\,y_{k+1}\ldots y_r \quad \text{and} \quad y'=y_1y_2\ldots y_k\,x_{k+1}\ldots x_r.$$

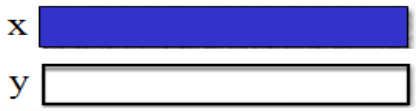
For example, if you have a possible representation of the two chromosomes:



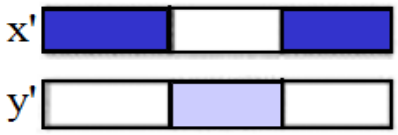
descendants will be:



Cross with more notch points. If more notch points, the segments are obtained by combining the rule again. We consider two crossing notch points. This type of crossing is done according to schedule below. Of chromosomes:



will give two descendants of the type:



In the case of three notch points, descendants will form:

Returning to our example, we consider the crossing with a single gash point. For example, the crossing of two solutions represented by the box that has the fitness 23, h =8 d =10 and the box with the fitness 26, h=14 and d=6, will give two descendants who will have the fitness 22, h=10 and d=6, respectively the fitness 38, h=12 and d=10 model below:

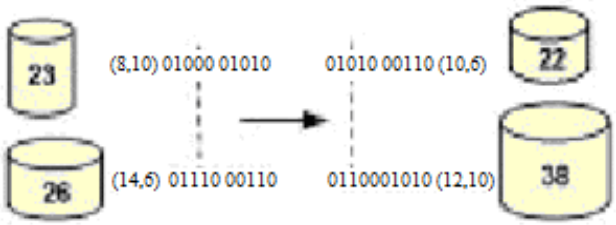


Fig. 2: Explanation of the application with a single crossing point cut.

It should be noted that the crossing not randomly generates descendants. Although it is unlikely that the cross between two solutions of the population to generate "sons" solutions most promising than the parent solutions, however, it shortly becomes clear that the chance to create more promising solutions is higher than in random search. From crosses with a single notch point of a pair of binary strings, it can only create two different pair strings who will have in its composition combining bits from both parents; son solutions being created are, probably, strings at least as promising. Therefore, not every meeting can create solutions as promising, but will not be less promising than their parents. If a less promising solution was obtained, then it will not appear when the next reproduction operator will be applyed and thus it will have a short life. If a more promising solution is created, then it is likely that she has more copies when the following reproductive operator implementation. To keep such a string selection promising During the reproduction operator application, not all strings of the population are used to cross. The crossing operator is primarily responsible for the search aspect of genetic algorithms, while the mutation operator is used for other purposes. The mutation is the second operator in the genetic order of importance and its use. The effect of this operator is the change of a single position from chromosome. By mutation other individuals are introduced in the population who could not be obtained through other mechanisms.

The mutation operator is acting on bytes whatever of their position in chromosome. every bit of the chromosome may suffer a mutation. In a chromosome may exist, in conclusion, more positions that undergo mutation. The Mutation is a probabilistic operator (ie does not apply safely). We consider an n population of individuals (chromosomes), each having length r. Each bit has the same probability p_m to suffer the mutation. There are several variants of the mutation operator. One of them would be the mutation into the strong form. In this case it proceeds as follows: it generates a random number q in the [0, 1) interval. If $q<p_m$, then the respective position mutation runs changing position 0 in 1 or 1 In 0. Otherwise, the position does not change. Returning to our example, if we apply the mutation operator to an obtained solution in the process of cross-breeding,to the solution that has fitness 22, we get a solution that will have fitness 16.

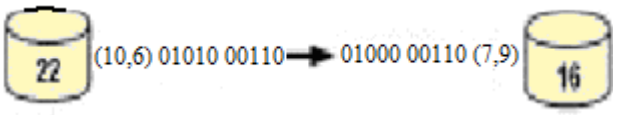


Fig. 3: Explanation of the application of the operator to move.

The solution obtained is more promising than the original solution. In consequence, the reproduction operator selects the most promising rows, cross operator combines two strings substring from the promising form to the most promising substring and mutation operator changing strings locally, also to improve the solution.

4 Evolutionary Strategies

Evolutionary strategies have been developed as a method for solving optimization problems parameters. First evolutionary strategy was based on a population consisting of a single guy. It is also used a single operator in the process of evolution: mutation. This is in line with the biological concept that small changes occur more frequently than big changes. Ussualy this strategy that a parent gives birth though mutation to a single descendant is known as evolutionary strategy 1+1. The way that this algorithm practically applys is simple: a solution is generated randomly on the search domain and mutations are made to it. The best of parent and descendants is chosen. The mutation operator is applied repeatedly until a solution is reached.

Another type of strategy is the strategy (μ+λ): μ parents produce λ descendants. New population (temporary) of (μ+λ) individuals is reduced again - through a selection process - to μ individuals. On the other hand, in the strategy (μ, λ), μ individuals produce λ descendants (λ>μ) and through the selection process a new population of μ individuals is choosen only from the crowd of λ descendants. Thus, the life of each individual is limited to one generation.

4.1 Evolutionary Programming

Original evolutionary programming techniques have been developed by Lawrence Fogel [6]. He sought a development of artificial intelligence in the sense of developing the ability to predict changes in an Environment.

Environment was described as a sequence of symbols and evolving algorithm supposed to obtain a new product, namely a new symbol. The symbol will maximize the final function who will measure the accuracy of predictions.

For example, we can consider a series of events marked a_1, a_2, ..., a_n, an algorithm will determine the next symbol (1 year), based on known symbols a_1, a_2 ,..., year.

The idea behind evolutionary programming is to develop an algorithm. As in evolutionary strategies, in evolutionary programming technique descendants are created first and then the individuals are selected for the next generation. Each parent produces a single descendant, so intermediate population size doubles (as in evolutionary strategy (*n, n*), where n is the size of the population). The descendant is created by a random mutation of the parent (it is possible to apply more than one mutation to an individual). A number of individuals (the most promising) equal to the size of the population are retained for the new generation. In the original version this process is repeated to obtain a new symbol which is available. Once obtained a new symbol, it is added to the list of symbols known and the whole process is repeated. Recently, evolutionary programming techniques have been used for solving numerical problems of optimization and many other purposes.

4.2 Genetic Programming

Another interesting approach was discovered relatively recently by John Koza [8]. Koza suggests that the desired program will evolve himself during a process of evolution. In other words, instead of solving a problem and instead of building a progressive program to solve the problem, we try to find a source code to solve. Koza developed a new methodology which provides a way to make this search. By example,we want to obtain a program Pascal or C++ to solve the problem of the Hamiltonian road or exit from a maze. So, we are not interested to get a solution to a set of some data, but rather, we are interested to get a source to generate a correct solution for any given entry. In other words, we are interested to get as result a similar program to which that we could have writen if we knew to solve the problem.

In terms of evolutionary the approach to such problems is generating a lot (population) random source codes, which are then selected based on function and fitness evolved through specific genetic operators. Most importantly we must assign a function of quality (fitness function)to each generated program. The fitness function should reflect the performance of the program of which it is attached. Usually the attaching of a fitness function is made running the program and measuring the solution quality in relationti with the solution which is known to be optimal.

A program will have a higher quality if its generated solution will be similar to the correct solution.It is not bad if an optimum solution Is not known previously, because we want to achieve solutions with a fitness as a high as it can be (or as small as it can be).

The evolution of the source program is done through specific genetic operators. For example, a recombination operator can mean the merging of sequences from a source code with sequences from another source code. A mutation operator could mean the insertion of new instructions in the source code, deleting of instructions, processing instructions. Obviously, after aplying those genetic operators a source code is generated that contains syntax errors. Also, useless source code sequences are generated. In what follows this will solve a

problem using genetic algorithms. It is considered M a lot of n and a number S. To determine of lot set M which has the sum of the number closer to S. Determination of lot amount of time a problem is *NP*-complete. This means that it is not known whether or not there is an algorithm of polynomial complexity to solve this problem. Until now, the algorithms used have exponential complexity, and some cases have pseudo-polinomial complexity. For example, we can reasonably solve this problem if the input data satisfy the following conditions: they are no more than 100 natural numbers, the amount not exceeding 500 numbers (more precisely, the number of numbers and their sum must not exceed the maximum allowable size for the allocation a matrix (we assume that it is statically allocated). If these conditions should be fulfilled, we could easily solve this problem using dynamic programming, using an algorithm of complexity $O(n \bullet S)$. However, if the numbers would not be whole but real, or their sum would be greater than 500, or differences between them would be so great. Then the algorithm by dynamic programming can not be used. I have listed here only cases, but can be imagined and other difficulties. For these reasons we will solve this problem using a genetic algorithm. We need to find a representation of the solution and also a function of fitness. How we represent our solution is given even stated the problem: it requires a lot of M whit n elements. So, a solution of the problem is a lot. We encode a lot by a string of length n which contains only values 0 and 1. If an item will have value k is 1, then lot will include the M_k (the k-th element of M crowd), and if position k is 0, then the item does not belong of lot [13]. The calculation of the fitness (quality) of a solution (of lot) is simple. Calculate sum of lot and fitness will be the difference (in absolute value) of the amount obtained and the number of S. Under these conditions the fitness will be minimized, because we want to determine an amount for which of lot elements is as close time value of S. The proposed genetic algorithm for solving this problem has been described above. We will use the tournament selection to obtain intermediate population. Genetic operators used are specific binary coding (turning a single point of scission, with mutation probability p_m=0.1).

Allocation of fitness. I have stated previously that genetic algorithms work with strings of bits representing the parameters and not the parameters them selves [12]. After he created a new series (a new solution) by genetic operators should evaluate it. In most cases, the fitness is just the criterion function for the solution. If our goal is to minimize the criterion, then we say that a solution is better than another, if the fitness of the two is greater.

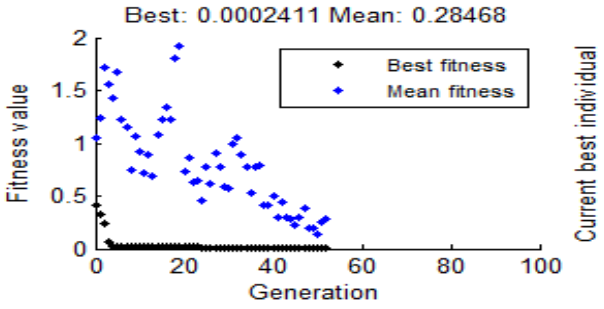
In another example of the problem, it is proposed to minimize one of the five features proposed by Ken DeJong in 1975, F1 (area):

$$f(x)=\sum_{i=1}^{3} x_i^3 \ , \ x_i \in [-5.12;5.12]. \qquad (6)$$

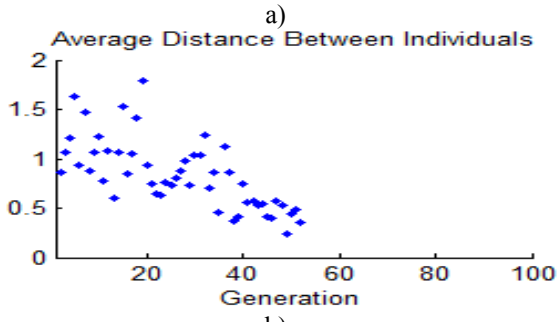
The global minimum in $f(0,0,0)=0$. After all calculations were performed for 100 generations, the results will be displayed [15]:

```
GA running
GA terminated
Fitness function value:
      2.4109846667385811E-4
Optimization terminated:
average change in the fitness value
```

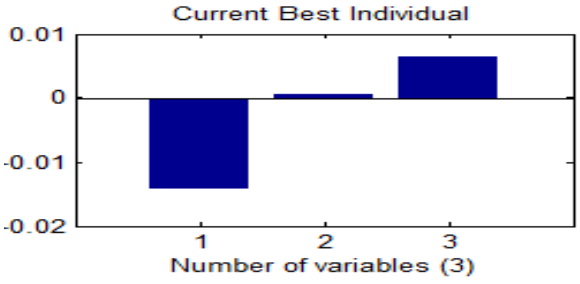
Also, during the evaluation can be seen in real time as parameters vary elected (Fig. 4). It is noted that as chromosomal approaching optimal, tend to behave like, because they are influenced by its predecessors. They say that they evolve to the optimum. To avoid congestion on the graphics will set a maximum value of 3.



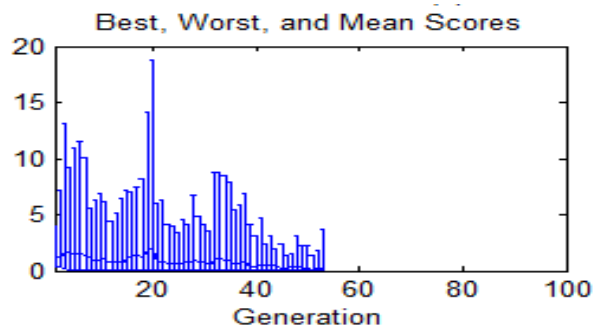
a)



b)



c)



d)

Fig. 4: Variation of parameters considered example.

In another situation will be considered continuous function:

$$f(x)=\sin\left(x^2-1\right)+\cos x+\sin x\,,\ x\in[-2\pi;0]. \quad (7)$$

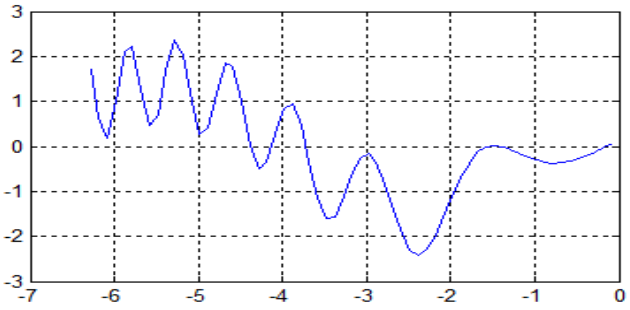
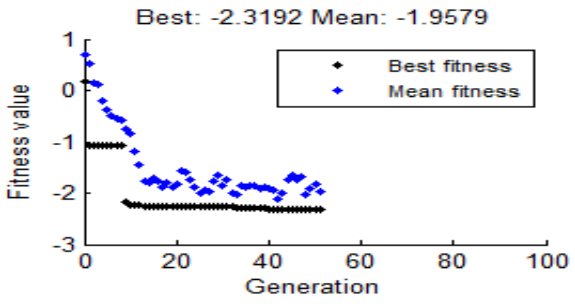


Fig. 5: Figure objective function.

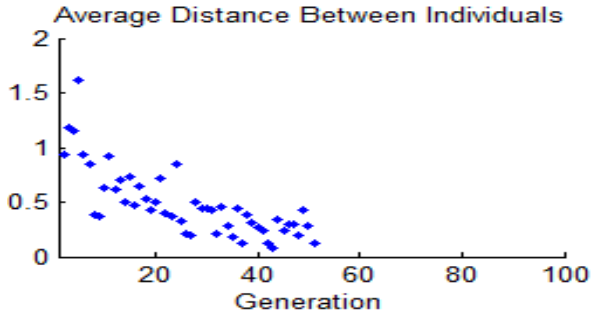
Function is the convex portions, and algorithms based on minimizing the search interval can not be applied on this definition, they offer a local optimum according to the boot. Choose the number of variables to 1.
Since the method is random, a result was obtained in the form [15]:

```
GA running
GA terminated
Fitness function value :
-2.3191745361054976
Optimization terminated: average
change in the fitness value less
then options
```

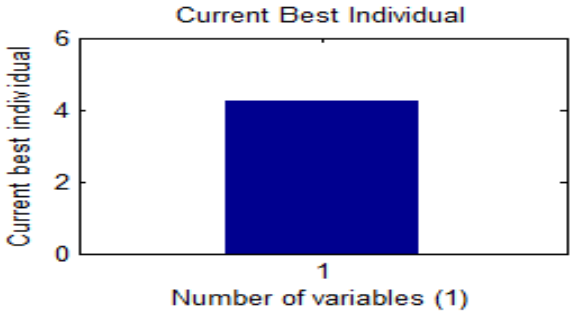
and is very accurate.



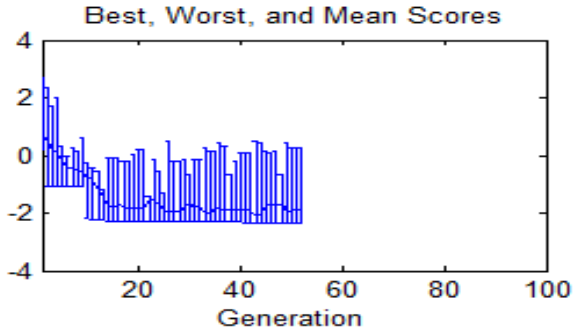
a)



b)



c)



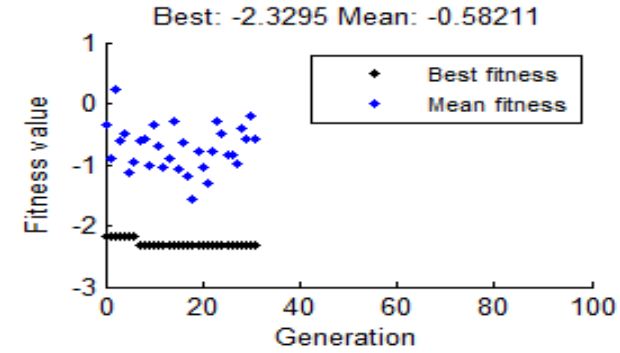
d)

Fig. 6: Variation of parameters considered example.

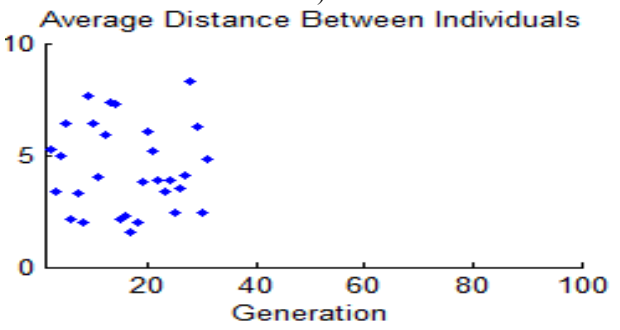
You specified that the boot was the default, i.e. [0.1]. The algorithm was able to "jump" and yet to find the global optimum with very high precision. If the change of uncertainty, as defined at the top to obtain a result as close to the global optimum. With such precision appropriate, we are interested to minimize the time needed to run the calculation algorithm.

```
GA running
GA terminated
Fitness function value:
-2.3294869995075116
```

The number of operations is strongly influenced by the number of chromosomes. Also be altered and the stop criterion. The solution was thus obtained in this case in only 38 generations, more quickly. Furthermore, it can proceed with a hybrid approach in solving the minimization of: running the genetic algorithm to obtain an approximate (around global optimum) and then apply an algorithm Deterministic.



a)



b)

Fig. 7: Variation of parameters considered example.

Function is considered:

$$f(x,y)=x^2+y^2+22\cos(x)\sin(y), \quad x,y\in[-\pi;\pi]. \quad (8)$$

What is more local minimum and only one global?

$$f(0,-1.4396) = -19.7385$$

illustrated by the graph in Fig.8.

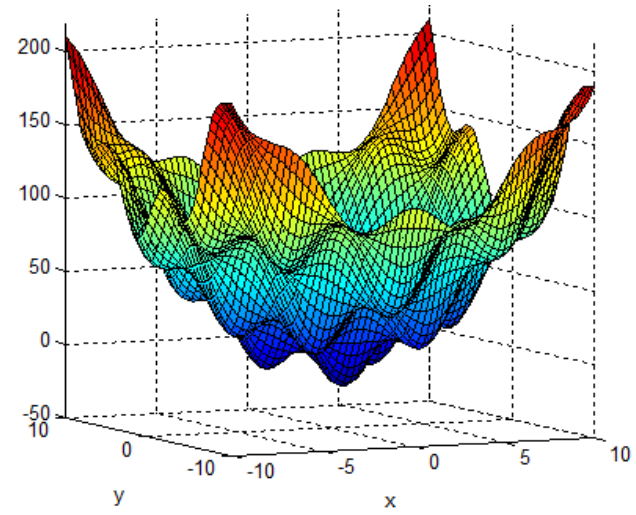


Fig. 8: Graph objective function.

In general, any abstract task to be performed, could be seen as solving a problem, which in turn, can be seen as a search in the potential solutions. As always, seek the best solution, we can look at this process as one of optimization. For small spaces, classical methods are comprehensive enough, for large spaces can be used special techniques of artificial intelligence.

The best known techniques of the evolutionary computation class are the genetic algorithms, evolutionary strategies, genetic programming and evolutionary programming. There are other hybrid systems incorporating various properties of the above paradigms, moreover, the structure of any evolutionary computation algorithm is largely the same [4].

5 Conclusions

Practical applications of these algorithms are numerous. They are used in more unexpected areas such as designing airplane wings or the design shape orbital stations. To solve a genetic problem, must take account of some recommendation.

To resolve a problem with genetic algorithms must be converted first into an optimization problem, ie to minimize or to maximize the value (the shortest hamiltonian chain, the largest component internally stable, etc.).

Genetic algorithms are Heuristic algorithms, ie the solution they found is not always best, but is in a neighborhood of the optimal solution. So if you have a choice between a polynomial algorithm that solves the problem and secure a genetic algorithm would be preferable to use the polynomial algorithm.

Genetic algorithms, typically have polynomial complexity. Therefore they are very often used to solve difficult problems (*NP*-complete). The results are very close to those obtained by certain algorithms, but have run thousands of hours.

If the issue is complex using a genetic algorithm and not an evolutionary strategy. Mutation is usually a weak search operator, so if it is used only, there is great opportunity to achieve local solutions and not global.

References

[1] Beasley D., Bull D.R., Martin R.R., *An Overview of Genetic Algorithms*, Part 1, Foundations, University Computing, Vol.15, No.4, pp.170-181, 1993.

[2] Boteanu N., Popescu M.C., *Optimal Control by Energetic Criterion of Driving Systems*, Proceedings of the 10th WSEAS International Conference on Mathematical and Computational Methods in Science and Engineering (MACMESE'08), pp.45-51, Bucharest, Romania, 7-9 november 2008.

[3] Bulucea C.A., Popescu M.C., Bulucea C.A., Manolea Gh., Patrascu A., *Interest and Difficulty in Continuous Analysis of Water Quality,* Proceedings of the 4th IASME/WSEAS International Conference on Energy & Environment, pp.220-225, Cambridge, 22-23 february 2009.

[4] Drighiciu M., Petrisor A, Popescu M.C., *A Petri Nets approach for hybrid systems modelling*, International Journal of Circuits, Systems and Signal Processing, Issue 2, Vol.3, pp.55-64, 2009.

[5] Dumitrescu D., *Algoritmi genetici şi strategii evolutive*: *Aplicaţii în inteligenţa artificială şi în domenii conexe*, Editura Albastră, Cluj-Napoca, 2000.

[6] Garey M.R., Johnson D.S., *Computers and Intractability: A Guide to NP-completeness*, W.H. Freeman and Company, New York, 1978.

[7] Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison - Wesley, Reading, MA,1989.

[8] Koza J.R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[9] Mastorakis N., Bulucea C.A., Manolea Gh., Popescu M.C., Perescu-Popescu L., *Model for Predictive Control of Temperature in Oil-filled Transformers*, Proceedings of the 11th WSEAS International Conference on Automatic Control, Modelling and Simulation, pp.157-165, Istanbul, May - June 2009.

[10] Oltean M., *Proiectarea şi implementarea algoritmilor,* Computer Libris Agora, Cluj-Napoca, 2000.

[11] Popescu, M.C., Balas, V.E., Popescu, L. *Heating Monitored and Optimal Control of Electric Drives,* 3rd International Workshop on Soft Computing Applications, Proceedings IEEE, Library of Congres 2009907136, pp.149-155, Szeged-Hungary-Arad-Romania, July - August 2009.

[12] Popescu M.C., Olaru O, Mastorakis N. *Equilibrium Dynamic Systems Integration,* Proceedings of the 10th WSEAS Int. Conf. on Automation & Information, pp.424-430, March 2009.

[13] Popescu M.C., Onisifor O., Mastorakis N., *Equilibrium Dynamic Systems Intelligence,* WSEAS Transactions on Information Science and Applications, Issue 5, Vol.6, pp.725-735, May 2009.

[14] Popescu M.C., Balas V., Olaru O., Mastorakis N., *The Backpropagation Algorithm Functions for the Multilayer Perceptron,* Proceedings of the 11th WSEAS International Conference on Sustainability in Science Engineering, pp.28-31, Timisoara, May 2009.

[15] Popescu M.C., Perescu-Popescu L., *Solving Applications by Use of Genetic Algorithms,* Proceedings of the 11th International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering, Published by WSEAS Press, pp.208-214, Vouliagmeni Beach, Greece, September 2009.