Solving Applications by Use of Genetic Algorithms

MARIUS-CONSTANTIN POPESCU 1 LILIANA PERESCU-POPESCU 2

1Faculty of Electromechanical and Environmental Engineering. University of Craiova
2E. Cuza College of Craiova
ROMANIA
mrpopescu@em.ucv.ro

Abstract: 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: Cromozoni, Genetic operators, Objective function.

1 Introduction

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. Giving is a problem poorly optimized is always possible to find an efficient algorithm whose solution is almost optimal. 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. 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 themselves. 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.

2 Genetic Operators

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

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 weinary coding.

Crossing point with a cleft. R be the length of chromosomes. A notch point is an integer \( k \in \{1, 2, \ldots, r-1\} \). 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:

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

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:

```
  x
  y
```

descendants will be:

```
  x'
  y'
```

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:

```
  x
  y
```

will give two descendants of the type:

```
  x'
  y'
```

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 applied 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 probabilist operator (ie does not apply safely). We consider an \( n \) population of individuals (chromosomes), each having length \( r \). Each bit has the same probability \( pm \) 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 \( q<pm \), then the respective position mutation runs changing position 0 in 1 or 1 In 0. Otherwise, the position does not change.

3 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. Usually 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 applies 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 \((\mu+\lambda)\): \(\mu\) parents produce \(\lambda\) descendants. New population (temporary) of \((\mu+\lambda)\) individuals is reduced again - through a selection process - to \(\mu\) individuals. On the other hand, in the strategy \((\mu, \lambda)\), \(\mu\) individuals produce \(\lambda\) descendants \((\lambda>\mu)\) and through the selection process a new population of \(\mu\) individuals is chosen only from the crowd of \(\lambda\) descendants. Thus, the life of each individual is limited to one generation.

### 3.1 Evolutionary Programming

Original evolutionary programming techniques have been developed by Lawrence Fogel. 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 \(a_1, a_2, \ldots, a_n\) an algorithm will determine the next symbol (1 year), based on known symbols \(a_1, a_2, \ldots, a_n\), year.

The idea behind evolutionary programming is to develop an algorithm. As in evolutionary strategies, in evolutionary programming techniques 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.

### 3.2 Genetic Programming

Another interesting approach was discovered relatively recently by John Koza. 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 written 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 relation to 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 applying 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 a mass a set \(M\) which has the sum of the number closer to \(S\).

Determination of mass 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-polynomial 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 $O(nS)$. 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 mass $M$ $n$ elements. So, a solution of the problem is a mass. We encode a mass by a string of length $n$ which contains only values 0 and 1. If an item will have value $k$, then mass will include the $M_k$ (the $k$-th element of $M$ crowd), and if position $k$ is 0, then the item does not belong mass.

This representation of a specific type mass set of Turbo Pascal. The calculation of the fitness (quality) of a solution is simple. Calculate sum of mass 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 mass 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 $pm=0.1$). In another example of the problem, it is proposed to minimize one of the five features proposed by Ken DeJong in 1975, $F_1$ (area):

$$f(x) = \sin(x^2 - 1) + \cos x + \sin x, x \in [-2\pi;0].$$  \hspace{1cm} (3)

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.

Fitness function value: 2.410984667385811E-4
Optimization terminated: average change in the fitness value less then options.

Also, during the evaluation can be seen in real time as parameters vary elected (Fig. 3).
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.

In another situation will be considered continuous function:

$$f(x) = \sin(x^2 - 1) + \cos x + \sin x, x \in [-2\pi;0].$$  \hspace{1cm} (4)

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:

GA running
GA terminated
Fitness function value : -2.3191745361054976
Optimization terminated: average change in the fitness value less then options.
and is very accurate.

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. 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. Function is considered:

\[ F(x,y) = x^2 + y^2 + 22 \cos(x) \sin(y), \ x, y \in [-\pi, \pi]. \]  \hspace{1cm} (5)

What is more local minimum and only one global:

\[ f(0, -1.4396) = -19.7385 \]

illustrated by the graph in Fig. 6.

![Fig. 6: Graph objective function.](image)

4 Conclusions
In this article were presented the main directions of genetic algorithms. 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