Genetic Algorithm for Fuzzy System Optimization: Generational or Lamarckian?

Zoltán Zsolt Ortutay, István Kádár

Budapest University of Technology and Economics
Department of Electric Power Engineering, 1521 Budapest
e-mail: ortutay.zoltan@gmail.com

Abstract: In the paper the simple genetic algorithm and the Lamarckian genetic algorithm compared. The results persent the benefit and drawback of Lamarckian variant.

Keywords: fuzzy control, optimization with genetic algorithm, lamarckian variant of genetic algorithm

1 Introduction

Genetic algorithms as methods of global optimization often applied in the design of fuzzy logic controllers e.g. for tuning of fuzzy membership functions. As the main drawback is the great time demand numerous modifications of organic-type algorithms attempt to speed up the optimization process [1]. On the other hand, appreciating by the reported papers the Lamarckian-type algorithms are not in use for engineering applications.

The task chosen in the individual project work subject of the electrical engineering study is to find out whether Lamarckian-type algorithms are suitable for optimization of fuzzy membership functions, may they be real alternatives or not.

The final aim is to implement a physical model of a small DC machine driven fuzzy controlled balancing system with optimized fuzzy controller.

In this paper the results of the first step is presented.

2 The Implementation of Genetic Algorithm

We proposed a fast working C++ code for testing, which algorithm is better for optimize our system. We choose this object oriented programming language,

because the running code is much more faster then other cases. We created our own header file, which made possible to simplify our source code.The basic idea was, that we may deduce the efficiency of the optimisation of the fuzzy control system from a particular step number of a plainer optimisation task. If we can appreciate the running time of the fitness function, which depends on the running time of the controlled system, then we can appreciate the optimisation 's time also. That is because calculating the fitness function is the slowest step of the algorithm. The appreciated time of fuzzy controlled system 's optimisation can be calculated as T_{fuzzy} = (Numbers of steps)*(Runnig time of Fitness Function)

3 Simple Generational Genetic Algorithm

Genetic algorithms are implemented as computer simulations in which a population of abstract representations of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. In presented case every solution is represented as an array of bits, in form matrices $n \times m$, where *n* is the number of individuals, and *m* is the length of the strings. Each row of the matrix is a solution of the task.

3.1 Main Steps of Basic Generational Genetic Algorithm:

1 Initialization of start population

2 Selection

3 Reproduction (Crossover, and mutation)

4 Repeat from the second step

3.1.1 Initialization of Start Population

Traditionally, the population is generated randomly, covering the entire range of possible solutions (the *search space*). In our algorithm stings are generated sting bit by bit randomly. Then a number with continuous uniform distribution between 0 and 1 generated. If this number was greater then 0,5 then the current bit of the string (called chromosome) set to 1, else to 0. This method repeated for every chromosome of the string.

$$array_{new}(n) = array_{old}(random_{array}(n)) \quad (1)$$

3.1.2 Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions measured by a fitness function are typically more likely to be selected. The roulette-wheel selection algorithm was used, which implemented as follows:

1 The fitness function is applied for each individual, providing fitness values, which are normalized. Normalization means dividing the fitness value of each individual by the sum of fitness values, so that the sum of fitness values of whole population equals 1. In that way we get an array of relative fitness values of individuals.

2 Another array created, where the elements of array were integer numbers generated randomly between one and the population size, representing the ordinal number of different solutions. The generation of random number was modified by the relative fitness values.

3 The last step of the algorithm was creating the array of selected solutions. The *n-th* element of the new array was equal to *k-th* element of the original array of solutions, where *k* was the value of the *n-th* element of the randomly generated array.

Generating Random Number According to Relative Fitness

In C++ such as many other programming language there is an easy way to generate continuous uniform distribution on the interval [0,1]. The uniform distribution describes a variable where the probability of occurrence of any value in range defined by the minimum and maximum values is equal. First we need to initialize the random number generator, traditionally the starting point is set according to the computer system time. Then a random number is generated between zero and one.

Using the step function this random number applied to calculate another number, which distribution depends on the relative fitness values. The step function consist of finitely many pieces and provides piecewise constant values. The number of intervals is equal to the number of individuals. Take an example of two individuals. The relative fitness value of the first is 0,25, and the relative fitness value of the second is 0,75. In this case number 2 (the order number of second individual) is three times higher then number 1 (the order number of first individual). The appropriate step function according to [2] is shown in Fig. 1:

$$step(x)\begin{cases} 0 \; if \;\; x < 0 \\ 1 \; if \;\; 0 \le x < 0{,}25 \\ 2 \; if \;\; 0{,}25 < x \end{cases} \tag{2}$$

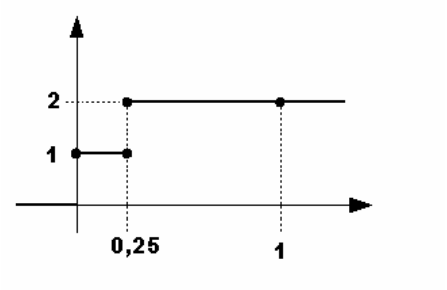

Figure 1
The step function from the example

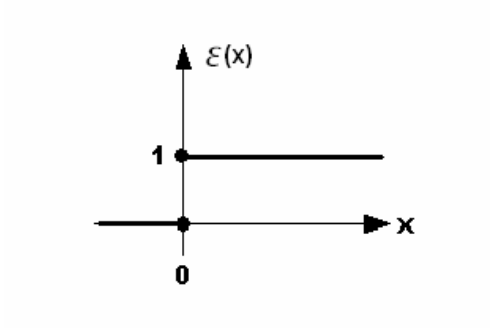

Figure 2
Step function according to (3)

If x is a uniform distribution random number on the interval [0,1] this step function provides 1 or 2, with the distribution we wanted. This function can be defined as the sums of step functions with offset. In this example:

$$\varepsilon(x)+\varepsilon(x-0{,}25) \tag{3}$$

where

$$\varepsilon(x)\begin{cases}0\ if\ \ x<0\\1\ if\ \ x\geq 0\end{cases} \tag{4}$$

Generally we can create arbitrary distribution using sum of step functions. The flowchart of the previous calculations can be seen in Fig. 3.

In the calculations double precision floating point numbers are needed, because of the division in definition of x. This algorithm is the reason,what makes important to normalize the relative fitness values. In case the sum of relative fitness values don't equals one,then the delay value wouldn't work on interval [0;1].

Values of the calculations form an array, which define the new (selected) population. The number of individuals in the new population is equal to the original number.

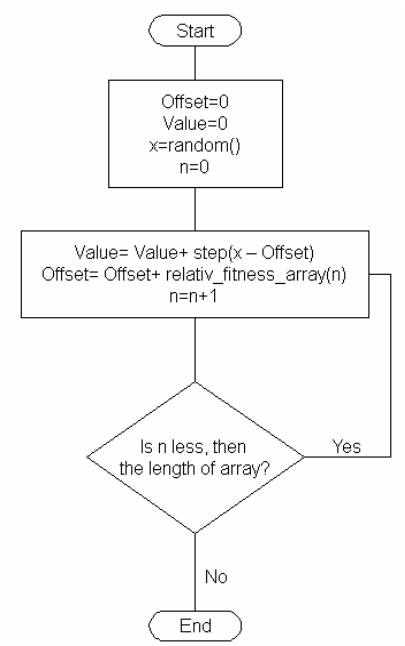

Figure 3
Generating random number according to relative fitness

3.1.3 Reproduction

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation. For each new solution a pair of "parents" is selected for breeding from the pool of previous solutions. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". These processes ultimately result in the new generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best individuals from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

A, Crossover

In genetic algorithms, crossover is an operator used to modify one or more chromosome from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. In this project both one-point, and two-point crossovers were tested, and the two-point method gave better results: The good solutions were more stables and the computing was faster.

Two-point crossover was implemented by repeating one-point crossover for each pairs of solution. Variable named number_of_crossover equals two, and new_population is a two dimensional array which represents all individuals In C++ the two-point crossover can be implemented as follows:

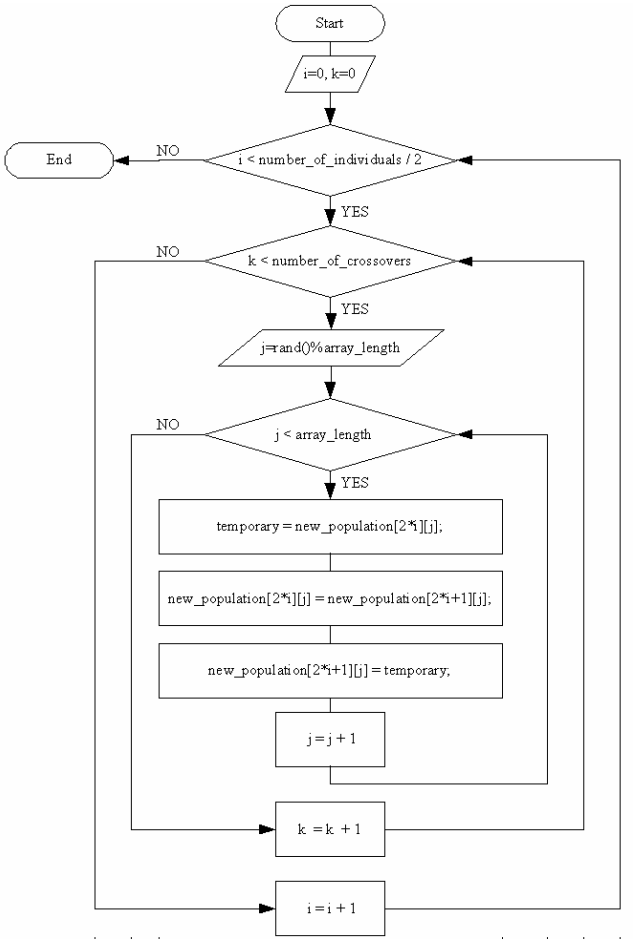

Figure 4
Flowchart of two-point crossover

B, Mutation

In usual genetic algorithms mutation is a operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation. The implementation of the mutation operator involves generating a random variable rm (called mutation rate) for each bit in array. This random variable tells whether or not a particular bit will be modified.

In the evaluated method the mutation is different. First we generate a random number (q1) of the bit in the array, which would be change. Another random number (q2) from interval [0,1] is compared to rm. Only if q2 > rm will the mutation performed applying a XOR function on the bit in question. The flowchart of that is the following:

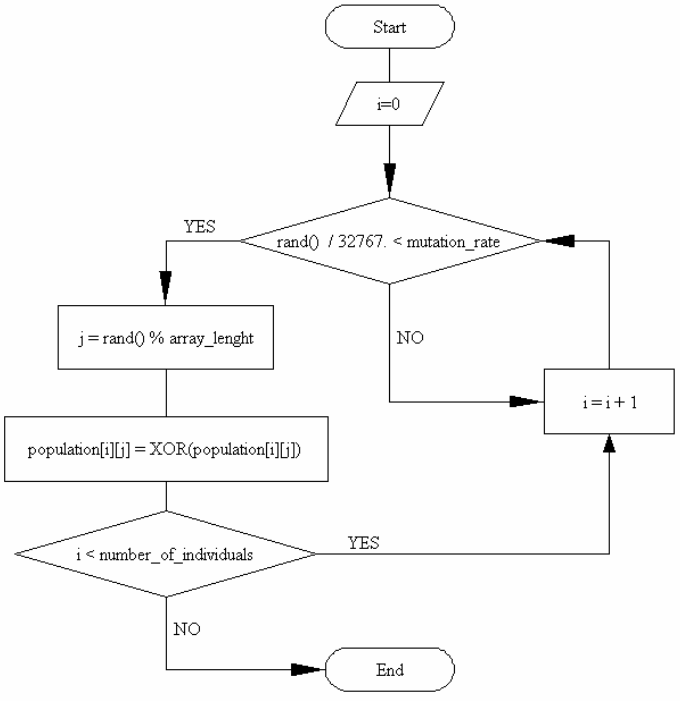

Figure 5
The flowchart of mutation

3.1.4 About the Lamarckian Idea

Lamarckian Gentic Algorithm is a genetic algorithm that employs Lamarckian inheritance, rather than Darwinian [2] – [6]. Jean-Baptiste Pierre Antoine de Monet, Chevalier de Lamarck (1744-1829) was a French soldier, naturalist, academic and an early proponent of the idea that evolution occurred and proceeded in accordance with natural laws. “Lamarck stressed two main themes in his biological work. The first was that the environment gives rise to changes in animals. He cited examples of blindness in moles, the presence of teeth in mammals and the absence of teeth in birds as evidence of this principle. The second principle was that life was structured in an orderly manner and that many different parts of all bodies make it possible for the organic movements of animals.” [7]

3.1.4.1 Implementation of Lamarckian Evolution

In our implementation of this idea there are two differences according to simple genetic algorithm. We eliminated crossovers from reproduction, and we used another mutation algorithm. In new mutation method a bit of the sequence , which represents a solution changes only if the new solution is better, then the original, and the mutation rate is higher.

3.1.5 Number of Iteration

The generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- a solution is found that satisfies minimum criteria
- a given number of generations reached
- allocated budget (computation time/money) reached
- the highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- manual inspection
- combinations of the terms above.

In our implementations we used the second method, and repeated generating new populations until, we reached a fix number (iteration number). We set this number to 10000.

4 Testing Algorithms

For testing the algorithms we chosen a non-linear function for calculating fitness values. We searched the maximums of these functions. Function we used is the following:

$$F(x) = \frac{x}{x^2 + 1} \tag{5}$$

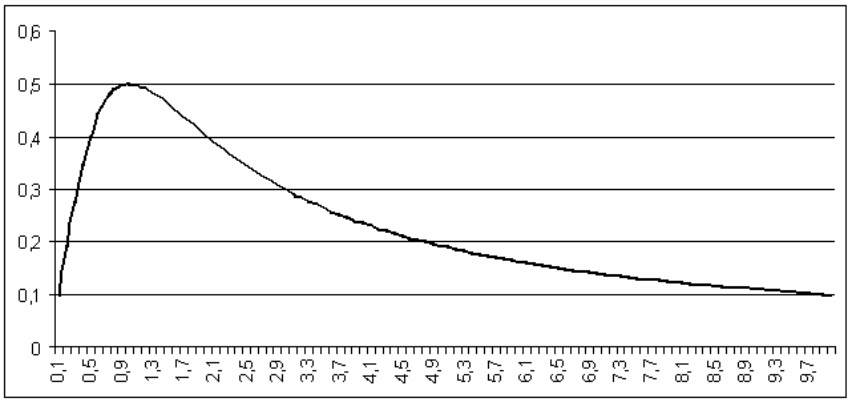

Figure 6
The non-linear funciton

5 Test Results

Both the simple and the Lamarckian algorithms used 32 individuals of 14 bit.

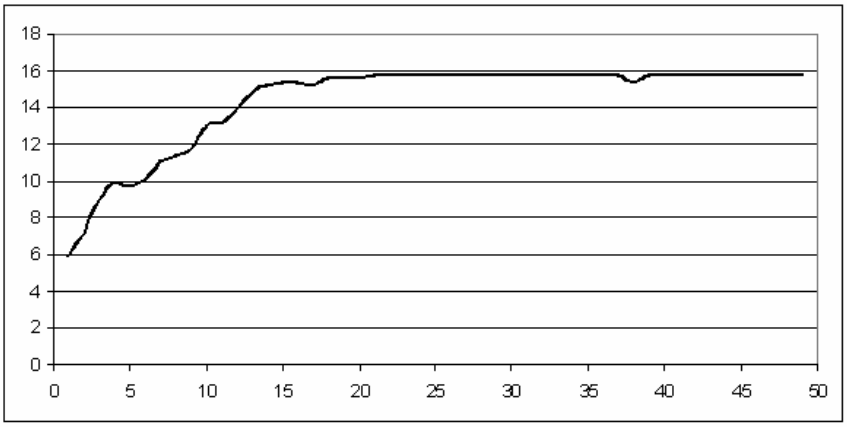

Figure 7
Sum of fitness values calculated, with Simple Genetic Algorithm during fifty iterations

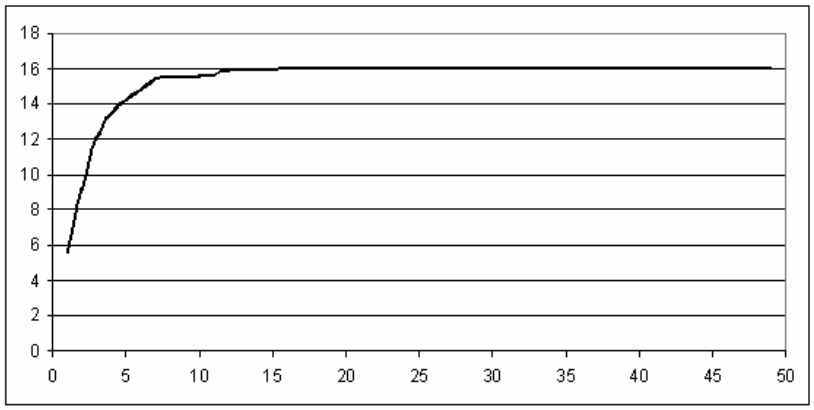

Figure 8
Sum of fitness values calculated, with Lamarckian Genetic Algorithm during fifty iterations.

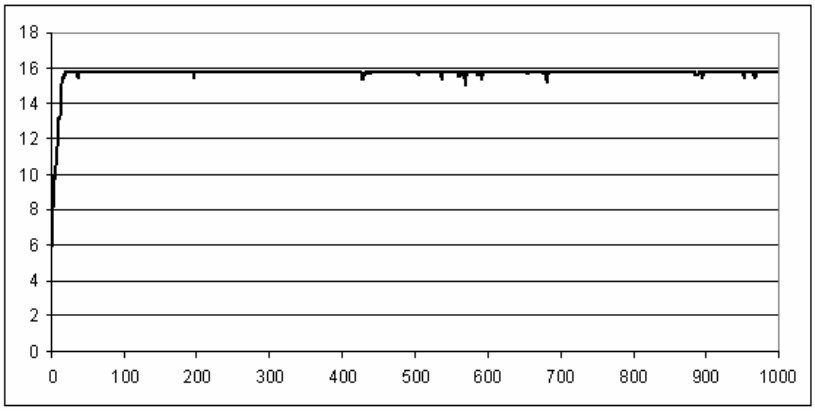

Figure 9
Sum of fitness values calculated, with Simple Genetic Algorithm during thousand iterations

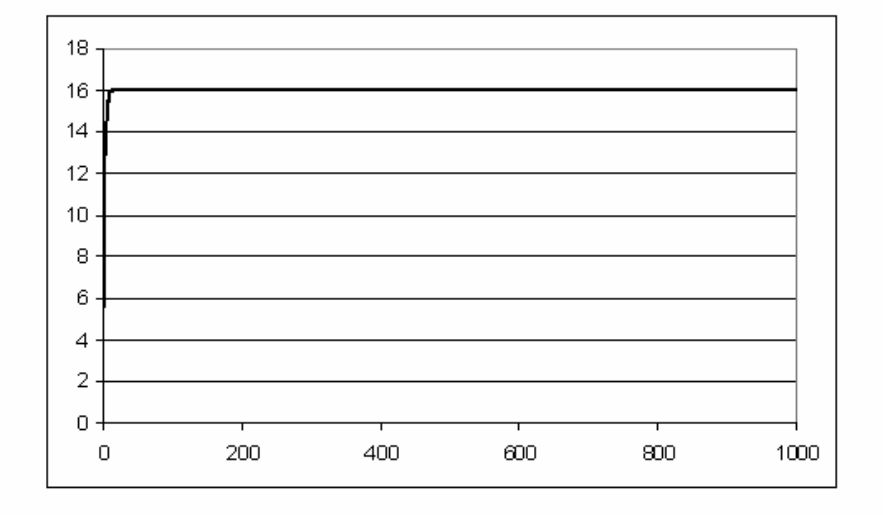

Figure 10
Sum of fitness values calculated, with Lamarckian Genetic Algorithm during thousand iterations

Conclusion

As seen in Figures seven to ten the step number of Lamarckian algorithm is less then the step number of simple genetic algorithm. Although this variant needs to calculate fitness values twice. For final appreciation further investigations needed.

Further Steps

The next step is design of the appropriate fuzzy controller, and adequate coding of membership functions and fuzzy sets.

References

[1] Gyula Retter: Combinations of Fuzzy, Neural, Genetic Systems. In Hungarian: Kombinált fuzzy, neurális, genetikus rendszerek (Kombinált lágy számítások). INVEST-MARKETING Bt., Budapest, 2007

[2] Alexandre Blansché, Pierre Gançarski and Jerzy J. Korczak: Genetic Algorithms for Feature Weighting: Evolution vs. Coevolution and Darwin vs. Lamarck, in Proceedings of Fourth Mexican International Conference on Artificial Intelligence, Monterrey, Mexico, November 14-18, 2005, pp. 682-691, Springer-Verlag Berlin Heidelberg, 2005

[3] Wuhong He, Haifeng Du, Licheng Jiao, Jing Li: Lamarckian Polyclonal Programming Algorithm for Global Numerical Optimization in Proceedings of First International Conference on Natural Computation, Changsha, China, August 27-29, 2005, pp. 931-940

[4] Wuhong He, Haifeng Du, Licheng Jiao, Jing Li: Lamarckian Clonal Selection Algorithm-based Function Optimization, in Proceedings of 8th International Work-Conference on Artificial and Natural Neural Networks, Barcelona, Spain, June 8-10, 2005, pp. 91-98

[5] Habib Rajabi Mashhadi, Hasan Modir Shanechi, Caro Lucas: A New Genetic Algorithm with Lamarckian Individual Learning for Generation Scheduling, in IEEE Transactions on Power Systems, Vol. 18, No. 3, August 2003, pp. 1181-1186

[6] Henry Fairfield Osborn: From the Greeks to Darwin: An Outline of the Development of the Evolution Idea, New York, The Macmillan Company, p. 160