Procedia Computer Science

Volume 51, 2015, Pages 316–325

ICCS 2015 International Conference On Computational Science

Genetic Algorithm using Theory of Chaos

Petra Snaselova and Frantisek Zboril

Faculty of Information Technology
Brno University of Technology
Brno, Czech republic
isnaselova@fit.vutbr.cz, zboril@fit.vutbr.cz

Abstract

This paper is focused on genetic algorithm with chaotic crossover operator. We have performed some experiments to study possible use of chaos in simulated evolution. A novel genetic algorithm with chaotic optimization operation is proposed to optimization of multimodal functions. As the basis of a new crossing operator a simple equation involving chaos is used, concrete the logistic function. The logistic function is a simple one-parameter function of the second order that shows a chaotic behavior for some values of the parameter. Generally, solution of the logistic function has three areas of its behavior: convergent, periodic and chaotic. We have supposed that the convergent behavior leads to exploitation and the chaotic behavior aids to exploration. The periodic behavior is probably neutral and thus it is a negligible one. Results of our experiments confirm these expectations. A proposed genetic algorithm with chaotic crossover operator leads to more efficient computation in comparison with the traditional genetic algorithm.

Keywords: optimization, genetic algorithm, chaos

1 Introduction

Chaos is the science of surprises and unpredictability. It teaches us to expect the unexpected. While the most traditional sciences deal with supposedly predictable phenomena like are gravity, electricity, or chemical reactions, chaos theory deals with nonlinear problems whose behaviors are impossible to predict or control, like turbulent fluid flow, global weather patterns, healthy heart rhythms, DNA coding sequences, and so on. However, complex dynamic systems with complicated feedback paths may often be understood in a sense of the chaos theory. Therefore, it is natural to consider that evolutionary processes can also be understood in a sense of the chaos. To explore this idea we incorporated a simple logistic function into standard genetic algorithm.

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.248

2 Multimodal Function

A large number of real-world problems can be considered as multimodal function optimization problems. Objective functions may have several global optima, that is, each objective function may have the same values in several different points in the explored state space. Moreover, it may have some local optima points in which objective function values lay nearby a global optimum. This paper is focused on multimodal problems with one global extreme only and more local extremes, because for solving problems with multiple global extremes further modification of proposed algorithm is needed.

3 Genetic Algorithm

Several methods have recently been proposed for solving the multimodal optimization problems. They can be divided into two main categories: deterministic and stochastic (metaheuristic) methods. Deterministic methods, e.g. gradient descent method or quasi Newton method, when they solve complex multimodal optimization problems, may easily trapped into some local optimum as a result of deficiency in exploiting local information. They strongly depend on a priori information about the objective function that can yield to fewer reliable results.

Stochastic algorithms combine randomness and rules mimicking several phenomena. These phenomena include physical processes (e.g. simulated annealing proposed by Kirkpatrick [11]), evolutionary processes (e.g. evolutionary algorithm proposed by Koza [8], de Jong [7], and Fogel [3], and genetic algorithms (GAs) proposed by Goldberg [4] and Holland [5]), immunological systems (e.g. artificial immune systems proposed by de Castro [2]), electromagnetism-like (proposed by Birbil [1]) and gravitational search algorithm (proposed by Rashedi [10]).

Genetic algorithms are inspired by Darwin's theory of biological evaluation as an optimization algorithm. The main idea of this technique is derived from natural evolution, so there are biological operators such as crossover, mutation and selection. Genetic algorithm has three randomly generated phases: *Initial population* of chromosomes, *crossover* operator and *mutation* operator [6]. Each chromosome represents some "solution" of the problem and its quality is done by the value of so called *object/fitness* function. Genetic algorithm starts by generating of some random solutions which are marked as an *initial population*. In the second step, random *crossovers* lead to produce new offspring and in the third step, with random value of *mutation* a few of genes in chromosome are changed or replaced. The new generation of candidate solutions is then used in the next iteration of the algorithm.

Traditional GAs usually well work for single optimum problems but they fail if they have to find multiple solutions.

4 Chaos

Very roughly, chaos is defined as a chaotic behavior of a non-linear dynamic system that is very sensitive on initial conditions and that is described by deterministic algorithm. Nowadays, practical use of the chaos significantly increased. Many applications in different fields have employed chaos theory and exploited its benefits.

Implementation, experiments and analysis of optimization processes based on chaos presented in this paper show that the chaos features are important and they can improve the efficiency of the genetic algorithm.

Chaotic function which we use in modified genetic algorithm is very well known logistic function (1).

$$z_{n+1} = \lambda z_n (1 - z_n) \tag{1}$$

This function takes a value z_n, which can be any number between 0 and 1, and changes it into a new value z_{n+1}. The new value of z is then back into the formula, and so on. Such process is called an iterative process. λ parameter in equation (1) is a number between 0 and 4. It is kept constant while the iteration process is run. The value of this parameter determines behavior of the variable z: it can be convergent, periodic or chaotic. Values of λ, that are smaller than 3, lead to convergent solution of equation (1), i.e. behavior of z variable converges to some constant value. When λ is between 3 and 3.56, then periodic behavior occurs. Firstly two different values of variable z alternate (for value of λ near to 3), when value of the λ increases, then four, eight, sixteen etc. values of z variable alternate. The solutions of equation (1) for λ between 3.56 and 4 become fully chaotic: neither convergent nor periodic, but variable with no discernible pattern.

4.1 The Logistic Function as a Genetic Operator

The chromosome is slightly different from traditional GA chromosome. One chromosome of the proposed algorithm is composed of three parts *solution*, *value of* λ and *mask of uniform crossover*. First, we will explain how the modified GA works, and then we will show it on a practical example.

Part of the chromosome containing the *solution* encodes the solution of the optimization function which is solved. The number of bits of solution is set by the required accuracy of the solution (the number of decimal ciphers). The solution encodes k values representing the coordinates $x_1, x_2, \ldots x_k$, therefore requires k-times the number of bits. The λ *value* in chromosome represents the λ from equation (1) and the number of bits required for this value is calculated similarly as numbers of bits of solution, but the number of bits is needed only once. *Uniform mask crossover* contains mask for crossing and its number of bits is set to the same number of bits as the solution. It is necessary to encode the mask into the Gray code.

The Gray code, also known as reflected binary code, is a binary numeral system where two successive values differ in only one bit (binary digit). The problem with natural binary codes is that together with change of one bit the number is not change only about number one, but about more count of number. Relation between Gray code and natural binary code is shown in figure 1.

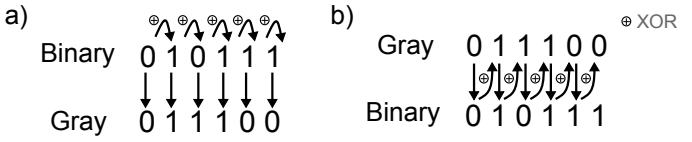

Figure 1: Gray Coding: (a) Converting from Natural Binary to Gray, (b) Converting from Gray to Natural Binary.

For example 3D function can be solved by chromosome, where the solution is encoded at 22 bits (value of x_1 at 11 bits and value of x_2 also at 11 bits), λ at 6 bits, and mask is encoded at 22 bits.

In the proposed algorithm uniform crossover is used. Now we will describe the emergence of the first offspring, the second offspring is created in the same way, but by swapping the first and the second parents. Each gene in the solution of the first offspring is created by copying the corresponding gene from the solution part of first or second parent chosen according to binary crossover mask. Where there is 1 in the crossover mask, the gene is copied from the first parent, and where there is 0 in the mask the gene is copied from the second parent.

Bits on position of λ value are inherited from the first parent, bits of masks are created using the modified mask of the first parent.

Mask is modified by the logistic function (1), using the λ *value* stored in the chromosome and mask itself as the input. The value of x_z is obtained by interpretation of the mask bits as real value, scaled into the range from 0 to 1. New mask is set as a x_{z+1} to the same bit interpretation of equation (1).

This modified crossover operation leads to self-directed evolutionary search the space for solutions via theory of chaos.

Both λ and the solution bits are also subjected to ordinary *bit-flipping* mutation. Figure 2 shows a flowchart of the modified algorithm, including the new crossover operator. In table 1 is a summary of used representation operators.

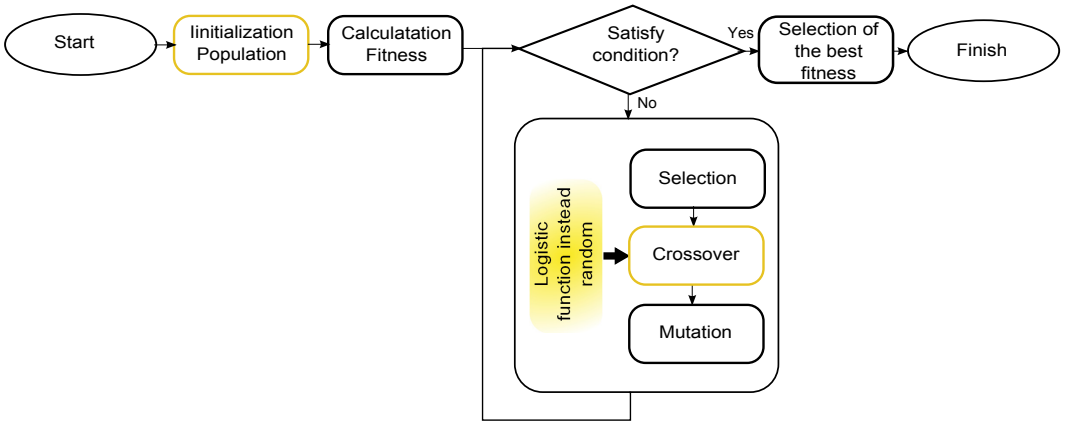

Figure 2: Comparison of basic genetic algorithm flowchart with chaotic genetic algorithm flowchart. Operation, in which the modified algorithm differs from the basic algorithm, is highlighted.

4.2 The Use of New Crossover Operator in a Practical Example

Now, we will demonstrate the use of the new crossover operator in a practical example, the summary of calculations and transformations from this example are shown in table 2. An accuracy of one decimal will be sufficient for this illustrative example. λ value lies in the interval from 0 to 4. Suppose, the number of required bits for λ value is 6 (the distance between two neighboring points is 0,0625 in this presentation). Interval of the searched solution is given together with an definition of optimization task [9]. Now is supposed a test area restricted to square $x_1 \in \langle -10, 10\rangle$ and $x_2 \in \langle -10, 10\rangle$. The number of bits needed to encode the mask is identical as the number of bits needed for the solution, both are encoded in 16 bits. The mask is decoded into interval $(0, 1)$.

Chromosome	Solution (coordinates)	λ	Mask (z_n)
Size	kn	m	kn
Range of phenotype	According to the task	(0.4)	(0.1)
Crossover type	Uniform with mask	Copy from 1. parent	Modified by logistical functions
Mutation typem	Flip bit	Flip bit	-

Table 1: Table describes the contents and sizes of the genes in a chromosome, the range of alleles and the genetic operations used on that gene. The solution represents the value of x_1, x_2, x_3 ..., x_k defined in the domain.

Suppose a randomly generated chromosome 0101010101101010 | 111001 | 1011001110010110. The first part of the chromosome is a solution, the values of x_1 and x_2. The second part of the chromosome is the λ value and the last part of the chromosome represents the mask of uniform crossover. It is important that the mask is encoded in Gray code. This chromosome will be marked as the first parent, randomly generated chromosome 1110000111001101 | 101101 | 1011001001100001 as the second parent.

The first step in the proposed crossover is the crossing with mask of a solution chromosome. Thus, after this step, we have a solution part of offspring chromosome 0101000101001011. This step is shown in table 2a. For the second step it is necessary to decode the λ value and mask. Decoding the mask from the binary vector v from Gray code into the interval $(0, 1)$ in system of real numbers is done as follows (summary in table 2b).

1. The binary vector v in Gray code is converted to binary vector v' in binary code.
2. The binary vector v' is converted to the decimal system. (k represents the number of bits of the converted value in binary code, it means the number 8. The mask is encoded in 8 bits.)[1]

$$m' = \sum_{i=1}^{k} v'^{i} \cdot 2^{k-i} \tag{2}$$

3. Then the corresponding real number is found by the application of a linear transformation as follows. Here, a symbol m is the calculated mask value, a is the lower limit of the interval of m, b is the upper limit of the interval.

$$m = a + \frac{b-a}{2^k - 1} \cdot m' \tag{3}$$

New mask value is obtained by substituting the calculated mask value m for the unknown variable x_n in the Eq. (1). The newly acquired mask value is then converted back to binary vector stored in the Gray code and together with the λ *value* is copied into the offspring. λ value is decoded in a similar manner and it is substituting for the parameter λ in the Eq. (1) during the calculation of the new mask.

[1]The mask consists of a mask for x_1 and a mask for x_2. The vector v thus contains only half of the all mask binary string, it means 8 bits.

The mask in Gray code for x_1 10110011 and for x_2 10010110 is written as binary numbers 11011101 and 11100100 and as 221 and 228 in decimal numbers. After the transformation into the interval $(0, 1)$ the mask is equal to the value 0.8667 and 0.8941. This mask value after the modifying by the Eq. (1) is for x_1 and x_2 0.3374 and 0.2765, after the reverse transformation from real number to decimal system is 86 and 70, in binary code 1010110 and 1000110 and in Gray code 1111101 and 1100101. The summary of mask crossing is shown in table 2b. By crossing the first and second parent we will get offspring: 0101000101001010 | 111001 | 11111011100101.

Solution	x_1	x_2
1. parent solution	01010101	01101010
2. parent solution	11100001	11001100
Mask of 1. parent	10110011	10010110
New offspring	01010001	01001010

(a) New offspring

Mask	x_1	x_2
Mask in Gray code (G. c.)	10110011	10010110
Mask in binary code (b. c.)	11011101	11100100
Mask in decimal	221	228
Mask in interval (0.1)	0.8667	0.8941
Mask modified by (1)	0.3374	0.2765
Modified mask in decimal	86	70
Modified mask in b. c.	1010110	1000110
Modified mask in G. c.	1111101	1100101

(b) Mask crossing

Table 2: Summary of the example. In (a) there is a example of crossing with mask of a solution chromosome. From solutions of 1. and 2. parent is created new offspring. In (b) is summary of the outputs of the proposed crossing method of a mask.

4.3 Expected Effect of New Genetic Operator

Consider the behavior of the logistic function in dependence of λ value. Suppose the λ value is below 3. After iteratively applying the logistic function from some initial value x_0 results converge to some value x_m. Speed of convergence decreases and the value of x_m increases with λ value approaching value 3. For the λ value which is larger than 3, the course of the function is an oscillating. This will tend to produce crossover masks that preserve the higher order bits of the mask, but vary the lower order bits. For this reason, it is desirable to encode the chromosomes using a Gray code, because here two successive values differ in one bit (binary digit) only. Near convergence, the mask will become fixed. Thus an individual with convergent λ will tend to produce offspring with progressively more rigid crossover masks. Individuals with non convergent λ will tend to have a high degree of variability in the crossover masks of their descendants, with the variability increasing as λ approaches the maximum value of 4.

The proposed algorithm has a large influence on the quality of behavior. Low values of λ should favor exploitation of good masks and large values of λ should favor exploration of the space of possible masks and consequently also solutions. Therefore one should expect greater exploration with new operator than without. Similarly, one should observe increased exploration when λ is in the chaotic domain rather than it is in the convergent one.

5 Genetic Algorithm Using Chaos and Benchmarks

We implemented the genetic algorithm using chaos based on simple logistic function Eq. (1) in Java and tested it using some famous benchmarks such as *Easom function* (this function

has one global significant extreme), *Goldstein-Price function*, *Rotated hyper-ellipsoid* (these two function are tasks with one slight global extreme), *Rastrigin function*, *Michalewicz function*, *Schwefel function* and *Shubert function* (functions with many local and one global extreme) [9]. Based on the results of tests was found that the novel algorithm takes effect only when we solve problems with multiple local extrema or problems with one global significant extreme. The results in the remaining groups of tasks were almost the same as the results of the classic genetic algorithm. Two functions that were chosen for representation in this publication are defined in [9] (*Easom function* and *Michalewicz function*) and have many good features for optimization problems such as kind of acceptable optimums.

6 Results of Solving Multimodal Problems

Now we show the results of some test that we have done with the proposed and the classical algorithm. Parameters of GA (such as probability of crossover, etc.) that were used during experimentation are as follows.

- Probability of crossover was 0.85,
- probability of mutation was 0.01,
- elitism was used in both algorithm,
- basic GA uses two pointed crossover and flip bit mutation,
- GA with logistic function uses new genetic crossover operator and flip bit mutation,
- calculation was performed for many generations and was ended by generation convergence,
- each run was repeated ten times and the results were averaged,
- 4 decimal places of accuracy were used.

To determine which scaling and selection algorithms give the best performance, we compared several types of selection algorithms, namely linear and exponential arrangement, probability roulette, tournament and probability tournament. Tournament selection became the most successful type of selection. It is versatile and can be used for all tasks. So we used tournament selection during further testing.

Figure 3 in 1. column shows the average of results of basic GA and proposed GA using chaos in a) *functions with one significant extreme* b) *functions with one slight extreme* c) *functions with many local extremes*. The results of experiments are the average of all task results belonging to the three individual type of functions mentioned above. We also compare the efficiency of the algorithms from the perspective of convergence rate depending on the size of the population. As it is shown in the figure 3 in 1. column, the results of GA using chaos are much more efficient than basic GA in cases c) *functions with many local extremes* and a little in a) *functions with one significant extreme*. The differences of results of b) *functions with one slight extreme* are small.

During further testing, we focused on the task with many local extremes, because they were significantly improved. Especially we perform experiments on *Michalewicz function*.

The figure 3 in 2. column shows the evolution of populations in dependence on the random numbers distribution during the initial generation of individuals. In the figure 3 a) individuals are generated according to the *uniform* distribution, the b) according to the *normal (Gaussian)*

distribution, and the c) according to λ *parameter logistic equation Eq.* (1), where the same number of individuals is generated in each interval $(0, 3)$, $(3, 3.56)$ and $(3.56, 4)$. As can be seen from the figure 3, the different distributions have no effect on the result.

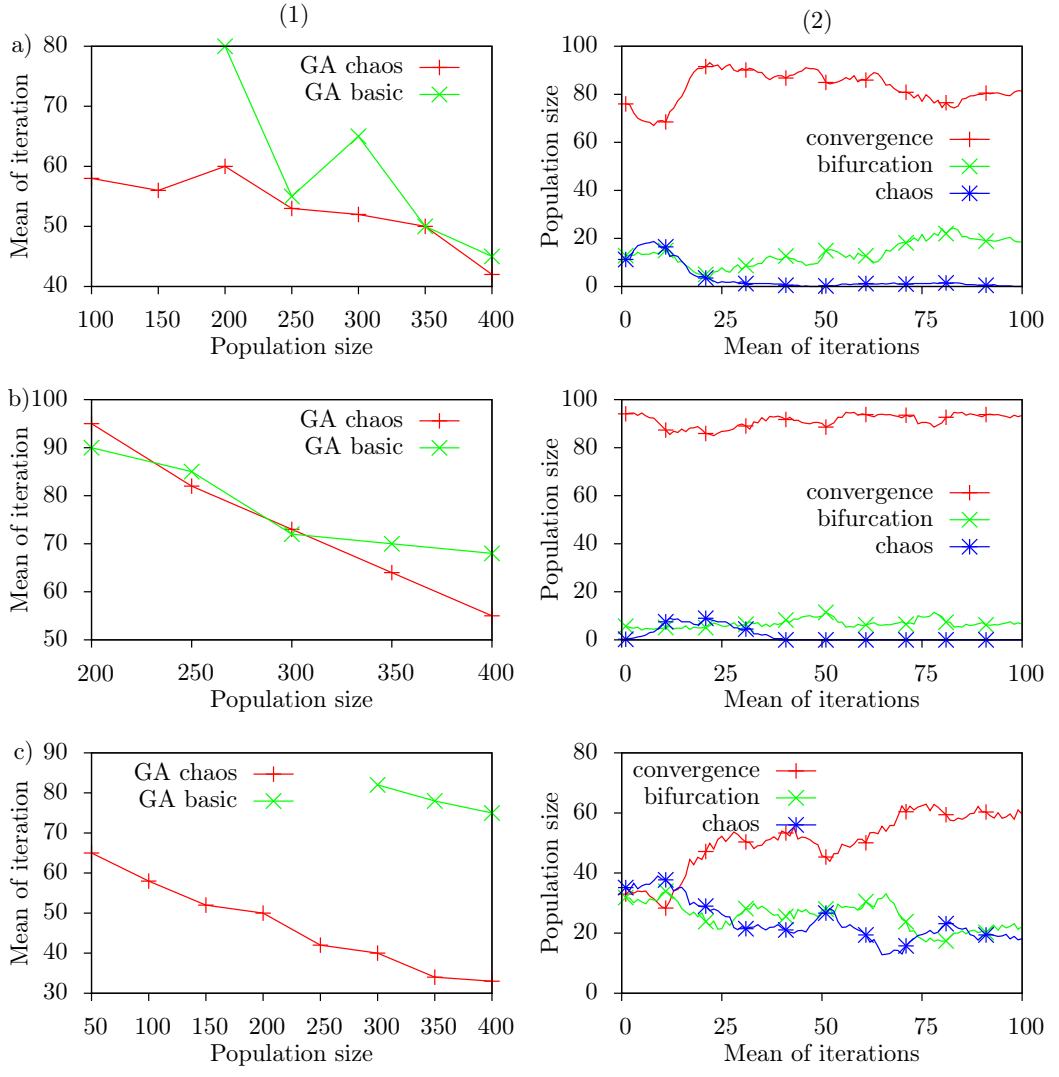

Figure 3: In (1) the figures show the decreasing of time complexity when population size is increased on: a) functions with one slight extreme, b) functions with one significant extreme, c) funtion with many local extremes. Only solutions that converges to the correct results are drawn. In (2) figures shows the number of specific λ values in each category (λ converging, bifurcations and chaos) in dependence on the number of iterations of the run. The initial population in the picture a) was generated by a uniform distribution, in b) by a normal (Gaussian) distribution and in c) in dependence on the boundary interval values of the parameter λ of Eq. (1), so that to each category of parameter λ belonged the same number individuals.

To get a better handle on the effect of new crossing operator, we forced several distributions of λ value on the initial population: including *convergent* values, *bifurcation* λ values, and

chaotic values. Different distributions are shown in the table 3. All three series of tests were performed with the population of 200 individuals and 100 of them survived to next generation. All three series of tests were performed with two mutation probabilities 0.01 and 0.001. Note that the lowest number of iterations and the lowest deviation for these tests occurred for one of the mix distribution cases. Apparently, a mixture of initial λ values in the population improves the effectiveness of proposal algorithm. There is a weak suggestion that the proposal algorithm with a mixed initial population is beneficial for obtaining low deviation per low number of iterations.

Even if a mixture of λ values in the initial population is beneficial, one category of λ values might tend to produce the highest scoring individual more frequently than the others do. As figure 3 in 2. column presents, it might have been expected that whatever value of chaotic λ have during a calculation tend to die off by the final generation. Individuals which possess a chaotic λ value have a positive effect for the beginning of searching space, individuals with a convergent λ value are beneficial at the end of calculation. Benefit of bifurcation λ values is neutral.

Type of GA	Initial population	Crossing operator	Probability of mutation	Deviation	Iterations
Classic GA	classic	2 point crossing	0.001	6.0492E-06	62
Classic GA	classic	2 point crossing	0.01	6.0492E-06	51
Proposal GA	mix	λ crossing	0.001	6.0492E-06	34
Proposal GA	mix	λ crossing	0.01	6.0492E-06	36
Proposal GA	convergent	λ crossing	0.001	1.2166E-03	100
Proposal GA	convergent	λ crossing	0.01	1.9918E-04	99
Proposal GA	bifurcation	λ crossing	0.001	1.9918E-04	94
Proposal GA	bifurcation	λ crossing	0.01	2.2777E-04	66
Proposal GA	chaotic	λ crossing	0.001	5.5243E-05	72
Proposal GA	chaotic	λ crossing	0.01	5.5243E-05	53

Table 3: Comparison of different GA. Mix initial population randomly select λ from all domains. Convergent, bifurcation and chaotic initial population selects from convergent, bifurcation and chaotic domains. Classic initial population randomly generates population without using chaos.

7 Conclusion

In this paper, we studied using of chaos in the optimization problems. A number of interesting trends were detected by a data.

In summary, the results show that the introduction of chaos into genetic algorithms can be useful.

New genetic operator with a mixture of λ categories improved the rate of calculation, but on deviation in solution it had no effect. It is only important that the initial population is represented by the chaotic and by the convergent values. The chaotic values presumably aid exploration, and the convergent values aid exploitation. There must be benefit in mixing of chaotic and non-chaotic individuals. The mixing of genetic material from exploring (chaotic) individuals and exploitative (convergent) individuals is the greatest advantage. Bifurcation of individuals are not useful neither for exploration, nor exploitation, but merely cyclic repetition of previously used genetic material.

We also found that the different methods of distribution of initializing an initial population does not affect the result.

Acknowledgment

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the Reliability and Security in IT project (FIT-S-14-2486).

References

[1] Ş İlker Birbil and Shu-Chering Fang. An electromagnetism-like mechanism for global optimization. *Journal of global optimization*, 25(3):263–282, 2003.

[2] Leandro Nunes De Castro and Fernando José Von Zuben. Artificial immune systems: Part i–basic theory and applications. *Universidade Estadual de Campinas, Dezembro de, Tech. Rep*, 210, 1999.

[3] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, Chichester, WS, UK, 1966.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[6] J. Hynek. *Genetické algoritmy a genetické programování*. Pruvodce (Grada). Grada, 2008.

[7] Rafal Kicinger, Tomasz Arciszewski, and Kenneth De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23):1943–1978, 2005.

[8] John R Koza. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Stanford University, Department of Computer Science, 1990.

[9] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

[10] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. Gsa: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

[11] C. D. Gelatt S. Kirkpatrick and M. P. Vecchi. Optimization by simulated annealing. *American Association for the Advancement of Science*, 220(4598):671–680, May 1983.