Using Genetic Algorithm with Adaptive Mutation Mechanism for Neural Networks Design and Training

Y.R. Tsoy, V.G. Spitsyn
Computer Engineering Department, Tomsk Polytechnic University
84, Sovetskaya street, Tomsk, 634034, Russia

Abstract - **In this paper a developed evolutionary algorithm (NEvA) for simultaneous connections and weights of neural network training is described. A distinctive feature of the algorithm is flexible and effective evolutionary search and a balanced resulting neural network structure due to adaptive mutation operator. In NEvA neural network structure changes, caused by mutation operator, as well as mutation rate are defined independently for each individual. Two different problems are chosen to test the algorithm. The first one is a simple 2-bit parity problem, well known as XOR problem, and the second is a neurocontrol problem of 1 and 2 poles balancing. A comparison of obtained results with results of other algorithms is presented.**

I. Introduction

The genetic algorithm (GA) uses evolutionary concept to find a solution [1]. Due to mechanisms of heredity, mutability and selective pressure, GAs show great adaptive abilities. Artificial neural networks (ANN) are often used to solve classification, approximation and control problems [2].

The use of genetic algorithms for ANN training is called neuroevolution or neurogenesis, and such algorithms are called neuroevolutionary [3]. The task of simultaneous structure design and weights tuning of neural network has not been solved formally and thus represents a complex problem. In this paper we describe developed algorithm NEvA (NeuroEvolutionary Algorithm), analyze its performance with use of two different problems and compare results with that of for known algorithms.

II. Algorithm Description

The developed algorithm models evolution of the neural networks population. Genotype of each organism contains information about connections and their weights of the network corresponding to this organism. An example is given on Fig.1. To encode information about connection the indices of start and finish neurons and the connection weight are stored in the genotype. This way of encoding corresponds to the direct method of encoding of information about neural network [4]. The weight of connection is encoded with 17 bits and its value is in [-65,536; 65,535] range with precision of 0,001.

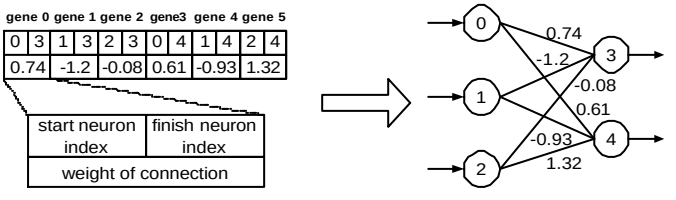

Fig.1. Genetic representation of ANN

For mating only those organisms were selected that had errors less than the average one. In result of two parents crossing two offspring are produced. These offspring inherit common neurons and connections for both parents and weights of such connections are recombined with use of 2-point binary crossover. Different connections are "gambled" between offspring. A mating example is shown in Fig.2. Solid lines denote common neurons and connections, and dash lines denote different elements of parental networks.

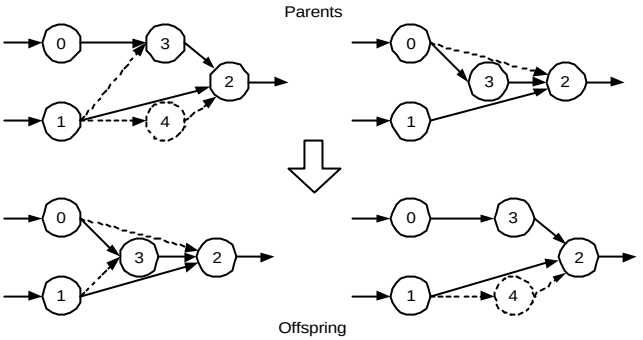

Fig.2. Mating example

In initial population all organisms represent networks without hidden neurons having all the input neurons connected to all the output neurons. Network weights are initialized randomly in range [-0,5; 0,5]. During the process of evolution networks "grow" and become more complex. This approach is similar to that of in NEAT algorithm [5], and seems to be more reasonable, than evolution of population of initially random-structured networks, because evolution with growth of complexity of solutions allows to evaluate neural networks topology more thoroughly and consistently. Neurons without offsets with log-sigmoid activation function were used. 1 elite organism passed to the next generation without any changes, i.e. mutation is never applied to the best individual in the current generation.

III. Adaptive Mutation Operator

Mutation mechanism allows to add and to delete neurons and connections and also to change connection weight for the random value. A new neuron is added with random input and output connections. When the neuron is deleted all the constrained connections are also deleted. Selection of type of mutation performed with respect to the neurons number and number of connections with use of the following heuristics:

$$f_C = \frac{N_C}{2^{FB-1}\left[N_N(N_N-1)-N_I(N_I-1)-(1-FB)N_O(N_O-1)\right]} \quad (1)$$

$$f_N = \frac{N_I+N_O}{N_N}, \quad (2)$$

here N_C – number of connections in network, N_I, N_O, N_N – accordingly number of the input, output and hidden neurons, FB – a flag, denoting whether recurrent connections arrival enabled (FB=1) of not (FB=0). It should be noted that connections from hidden neurons to output ones can appear in any case. Thus f_C coefficient is necessary to estimate "linkage" of neurons in network, i.e. the more number of possible connections, the more the coefficient's value is. Use of the second

coefficient f_N is based on heuristic assumption that the more total number of input and output signals is, the more complex network structure is necessary to solve given task.

To choose mutation type we use coefficients calculated with (2) and (3) and then squared. We denote them as F_C and F_N. Schematically the algorithm of selection of the applicable mutation change is shown in Fig.3. Here *Rnd* – uniformly distributed random value in range (0; 1), N_N – number of the hidden neurons in network.

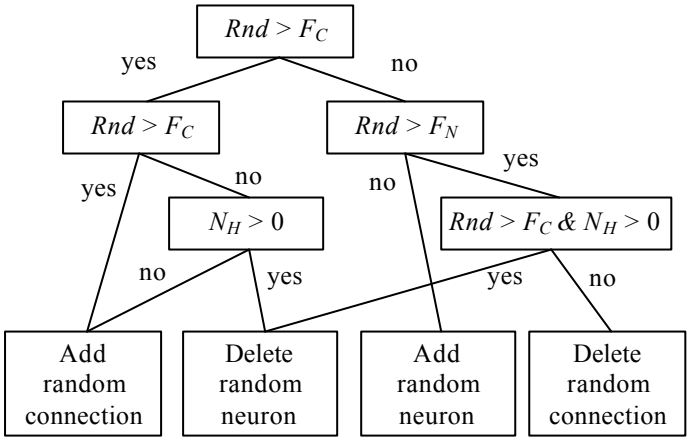

Fig.3. Mutation type selection scheme

To simplify, the adaptive mutation algorithm can be divided into two branches (by the first conditional transition):

1) Branch of f_C decrease (relative grow of the neural network complexity).

2) Branch of f_C increase (relative recession of the neural network complexity).

Since neuron deletion can lead to decrease as well as increase of f_C (it depends on the number of connections associated with this neuron) then this type of mutation exists in both branches. Thus the main factor of network structure regulation is its "linkage" degree.

Such method of mutation type selection from one hand doesn't limit explicitly number of hidden neurons from the top and, from the other hand, prevents too fast growth of neurons quantity since addition of each new neuron happens with less probability. Connection weight mutation occurs always 1 time per organism and changes the weight value for random number from range [-0,5; 0,5].

Let's look at an example of mutation type selection with respect to the introduced scheme (Fig.3). Let the network on Fig.4 is undergoing mutation. It is necessary to define which type of mutation change will be applied to this network.

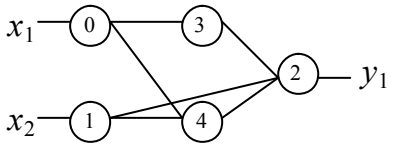

Fig. 4. Network for the example of mutation type selection.

We will denote probabilities of events "add random connection", "delete random connection", "add random neuron" and "delete random neuron" as p_{+conn} , p_{-conn} , p_{+neur} and p_{-neur} respectively. To estimate values of these probabilities F_C and F_N coefficients should be calculated first. So:

$$f_C = 6 /(0{,}5*(5*4 - 2*1 - 1*0)) = 0{,}667,$$
$$f_N = 3/5 = 0{,}6,$$
$$F_C = f_C^2 = 0{,}445,$$
$$F_N = F_C * f_N^2 = 0{,}160.$$

Multiplication of f_N^2 and F_C is necessary in order to change neurons number with respect to the current network topology, since addition or deletion of neuron demands an information about expediency of such a change. This information can be obtained indirectly via f_C value.

The case when there are no hidden neurons in network will be denoted via $\alpha = 1$, and the case when there are hidden neurons corresponds to $\alpha = 0$. Then according to the scheme on Fig.3 the following equations take place:

$$p_{+conn} = (1 - F_C)(1 - F_C + \alpha F_C),$$
$$p_{-conn} = F_C(1 - F_N)(F_C + \alpha),$$
$$p_{+neur} = F_C F_N,$$
$$p_{-neur} = F_C(1 - F_C)(2 - F_N)(1 - \alpha).$$

Taking into consideration that $\alpha = 0$ we can calculate probabilities of selection of certain mutation transformation:

$$p_{+conn} = 0{,}308,$$
$$p_{-conn} = 0{,}166,$$
$$p_{+neur} = 0{,}072,$$
$$p_{-neur} = 0{,}454.$$

Thus the probability that a random hidden neuron will be deleted is higher than the other probabilities although the probability to add a connection is also high. As it can be seen from the example the scheme for selection of mutation type is effective against growth of the neural network.

Not only a mutation type, but also a mutation operator probability is defined adaptively. In classical description [1] mutation operator for binary genotypes is applied with certain probability P_m and this event is "gambled" as many times as the number of alleles in the genotype, in other words accordingly to the number of bits in chromosome. This is the reason to choose mutation probability relatively small, because in the opposite case, when mutation probability exceeds some value (known as "mutation error threshold" [6]), evolution of the population "behaves" like a random search. Thus mutation probability equal to *1/L*, here *L* is a number of the smallest mutable parts of chromosome (single bit for binary representation and single connection encoded in genotype for NEvA), seems reasonable enough, since it assumes in average 1 mutation per organism. But for the later stages of evolution it's better to decrease mutation probability in order to lower losses of "good" individuals. Taking into consideration these ideas it was decided to "gamble" mutation event N_I*N_O times, here N_I and N_O – accordingly number of the input and the output neurons in network. So then, since we use a concept of "growing"

neural networks, it is possible to say that mutation effect decreases as evolution goes. E.g. L value increases and probability of 1 mutation event, which equals to $\frac{N_I \cdot N_O}{L}$, becomes smaller. Thus, since each genotype can represent network with different complexity, mutation rate is defined individually for every organism in the population.

IV. Tasks description

To test the introduced algorithm the following two problems that are acknowledged as classical ones for neuroevolutionary algorithms are chosen:

1) XOR problem. This is a rather simple problem where a a neural network that implements binary EXCLUSIVE OR operation should be built. So the network should have 2 inputs and 1 output and should produce output value close to "0" if there are (0, 0) or (1, 1) input signals and also output "1" in case of (1, 0) or (0, 1) input vectors. In experiments network error is calculated as halved average square difference between network response and required output value. Given neural network is regarded to be a solution if its error is less than some error threshold. For this research the threshold value of 0,001 was used. Algorithm performance is estimated with use of average number of object function calculations, i.e. how many times a training data was introduced.

2) Pole balancing problem, also known as "Inverted pendulum" [7]. This problem concerns neurocontrol domain. The task is considered to be hard for "standard" methods of neural networks training (for example, gradient based learning algorithms) because of constantly changing state of the control object. The goal is to try to balance 1 or 2 poles on the cart, moving this cart in the horizontal direction straightforward or backwards. There are several variants of this task involving discrete or continuous force, pushing the cart, and full or incomplete information about poles movement. There is also a variant when the cart is moved in 2D environment [8]. In this paper we will test NEvA algorithm for the following versions of pole balancing problem: (1) 1 pole with discrete force; (2) 2 poles with continuous force. Mathematical model for pole balancing problem is described in [9]. In experiments number of trials before the solution network will have been found is calculated.

V. Results and discussion

We first introduce results for XOR problem. The performance of NEvA algorithm is compared with that of canonical genetic algorithm (CGA) from [10], NEAT [5, 12] and two variants of back-propagation learning algorithm: simple (BP) and with momentum (BPM). We also made a comparison of NEvA with and without adaptive mutation probability (P_m). In the last case probability of 1 mutation per genotype was set to 0,2. Results for XOR problem, averaged over 50 runs, are shown in Table I. Average number of hidden neurons and number of connections denoted as N_H and N_C respectively. Results for the NEAT algorithm denoted as NEAT1 [5] and NEAT2 [12] are obtained for another error thresholds: 90% of maximum fitness (error threshold is approximately 0,005) for NEAT2, and in case of NEAT1 error threshold was not reported, although it seems to be larger than 0,005.

Although both NEvA and NEAT lose to gradient methods, we should note that the task was much more complex for neuroevolutionary algorithms, since not only weights but also network structures were tuned. And also NEvA outperforms CGA as many as 2 times and shows better performance than NEAT2, although NEAT networks are much simpler. A hypothesis can be proposed that since NEvA uses small population then phenotypic diversity inside one generation is large and exploitation of neural topologies is done very inconsistently. That is, only a few networks with the same topology are present at one generation and taking into consideration enormous number of possible combinations of weights it's clear that level of deception is high and thorough investigation of each structure is hard to do. So then networks in next generation are likely to have another structures, that differ from nets in previous generation, and this makes search with small population in some sense chaotic. And, on the contrary, NEAT uses speciation and together with rather large population size this technique allows to explore each "niche" in search space thoroughly enough. NEvA with adaptive setting of mutation probability is slightly better than algorithm without such ability, and networks in average are 'better" than networks obtained by NEvA algorithm without adaptive P_m.

TABLE I
RESULTS FOR XOR PROBLEM

	Average number of object function evaluations	N_H	N_C	Population size
NEvA	5904,64	4,62	15,3	13
NEvA (no adaptive P_m)	6026,26	4,8	16,22	13
NEAT1	4755	2,35	7,48	150
NEAT2	6612	3,12	11,72	150
CGA	13165,63	4	17	50
BP	5338,28	4	17	-
BPM	828,32	4	17	-

For single pole balancing benchmark we compared NEvA with GENITOR, SANE and ESP [9] algorithms. Number of trials, before successful network is found, was estimated. Results averaged over 50 runs are presented in Table II.

TABLE II
RESULTS FOR 1-POLE PROBLEM

Algorithm	Number of trials			N_H	N_C
	Average	Best	Worst		
GENITOR	1846	272	7052	5	35
SANE	535	70	1910	8	40
ESP	285	11	1326	5	30
NEvA (no adaptive P_m)	451	37	3872	0,58	5,22
NEvA	349	67	1027	1,22	6,52

By number of trials NEvA loses only to ESP algorithm and performs better than all the other algorithms. Neural networks, obtained by NEvA, are much better in all cases. It's noticeable that results for NEvA with adaptive mutation probability are better than that of for variant with fixed mutation rate, whereas

networks are "worse" since number of connections and hidden neurons is larger for NEvA results. Note that results of NEvA with adaptive mutation rate have better characteristics than NEvA with fixed P_m since the difference between the best and the worst results is much less. This means that NEvA with fully adaptive mutation operator showed more stable results for 1-pole balancing problem.

Results for 2-pole problem and information about obtained neural networks are shown in Table III. Results for evolutionary programming is taken from [11], for SANE and ESP from [9], for algorithm NEAT from [5]. Results were averaged over 50 runs, and results of NEAT algorithm were averaged over more than 150 runs [5]. There is no information in [5] about average number of neurons and connections for NEAT, but the range of these parameters is known, so their values seem to be relatively small.

TABLE III
RESULTS FOR 2-POLE PROBLEM

Algorithm	Average number of trials	Population size	N_H	N_C
Evolutionary programming	307200	2048	N/A	N/A
SANE	12600	200	N/A	N/A
ESP	3800	200	5	35
NEAT	3578	150	N/A (0–4)	N/A (6–15)
NEvA (no adaptive P_m)	3777	59	1,4	7,24
NEvA	2177	40	0,74	7,7

The performance of NEvA without adaptive P_m is comparable with that of ESP and NEAT. The results for NEvA with fully adaptive mutation mechanism surpass the results for all the other algorithms. Comparison of two variants of NEvA shows that for 2-pole balancing problem adaptation of mutation probability allows to leave less "adaptive" variant behind. Results for network structures are quite interesting. Number of hidden neurons is less for NEvA with adaptive P_m whereas number of connections is larger. Note that networks of NEvA with adaptive P_m seem to be better then networks obtained by NEAT, although NEAT uses speciation which allows to get compact solutions. It's hard to give a definite answer for this circumstance because results for XOR task are different but, probably, the following feature of NEvA can explain some things.

One of the peculiarities of introduced algorithm is that connections and neurons can be deleted in result of mutation. The side effect here is a probable isolation of some less-informative inputs in obtained solution. For example in pole balancing problem the information about cart speed and acceleration is not obligatory to balance poles successfully. So the neural network inputs correspondent to such signals can be deleted. For 2-pole balancing problem we obtained the following results (over 50 runs) of inputs deletion:

1) NEvA without adaptive P_m: 1 input for cart movement characteristics was deleted 7 times, 1 time both inputs about cart movement have been deleted;

2) NEvA with fully adaptive mutation operator: 8 times algorithm deleted 1 input, and in no run 2 inputs were deleted.

This effect is not guaranteed but can be quite a bonus of course if the population evaluation procedure is correct and/or the training data is sufficient. Thus there is always an opportunity to make a "rollback" in evolution of growing neural structures to look for solution amidst lesser complex network topologies. Likely, this circumstance makes rather significant impact on average number of hidden neurons and number of connections. For XOR task there was no insignificant inputs so the initial topology (the one without hidden neurons) could not be simplified.

There is a lot of work to do to improve the introduced algorithm. Our next goal is to try to implement adaptive population sizing in order to make the NEvA algorithm as autonomous as possible. All other parameters (crossover operator, selection and so on) seem to work well when their characteristics are fixed. Besides since the parameters of genetic algorithm have very complex interactions it's very hard to try to tune them all "on-line". We've already done some preliminary experiments with population sizing for genetic algorithm. The first results are quite interesting. It seems that GA starting with rather large population of size *N1*, which decreases steadily to size *N2*, performs better than GA with small initial population of size *N2* that increases up to size *N1*. In other words it seems that the first generations of evolution (when population sizes are close to either *N1* or *N2*) are more valuable than the later ones, possibly, because a probability to find a search space area with high fitness is larger at the beginning.

Other significant "milestones" are to:

1) Apply NEvA for tasks, where solutions with recurrent connections are necessary. Such tasks are, for example, 1 and 2 poles balancing without information about speed of cart and poles, so that solution network should have some memory to manage to keep poles on the cart.

2) Investigate generalizing capabilities of the networks obtained by NEvA algorithm. The generalization property is crucial for classification problems enabling successful classification of the "unknown" input data.

3) Test algorithm on tasks where large number of input and output neurons for resulting network is required, to examine adaptive mutation operator behavior more thoroughly. As an alternative it's possible to suggest a task where "long" evolution (more than 200 generations) is required. This is required to test adaptive mutation technique on complex networks.

VI. IMPLEMENTATION

The introduced algorithm is implemented to comply with the architecture of the software environment "GA Workshop" briefly described in [13]. The notion of the "GA Workshop" is to separate GA from the problem solved and the processing of run data with use of design patterns [14]. General scheme of this environment is presented on Fig. 5.

Thus it is possible to implement new tasks without, in most cases, any changes in GA source code. And vice versa: one can bring new features into GA implementation without any modifications in the other blocks of the environments. Although

"GA Workshop" is supposed to be used for research purposes its architecture allows easy implementation of many applications to solve real-world tasks.

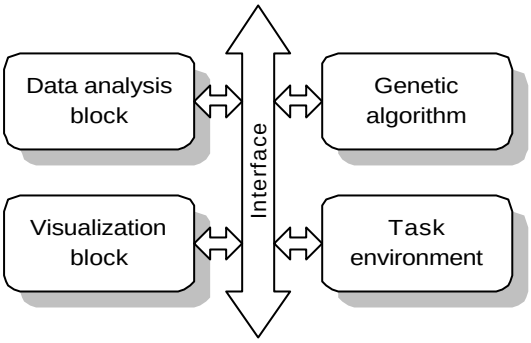

Fig. 5. "GA Workshop" architecture

At the time "GA Workshop" is under construction, although it is already possible to use it for researches. We have already used this environment for the investigation of GA with simple dynamic sizing scheme, experiments with NEvA algorithm, the study of quasi-species evolutionary model by Nobel Prize winner M. Eugen and for some other problems. For now there are:

1) Three different variants of GA, which include "standard" GA, compensatory GA [15] and NEvA algorithm introduced in this paper.

2) There are 5 different crossover operators for the "standard" and compensatory GA and 4 different crossover operators for NEvA.

3) Different selection strategies are also implemented. By now there are 4 widely used selection strategies in "GA Workshop".

4) There are 12 benchmark problems, including 7 numerical optimization tasks, 2 binary combinatorial tasks and 3 tasks for NEvA algorithm.

5) Parameters of GA run (population size, genetic operators type and probabilities, different stop criteria) as well as parameters for the all before mentioned components can be set in numerous combinations allowing thorough examination of the performance of GA with different characteristics.

The following data is available for analysis of results of multiple independent runs of GA:

1) Data that describes each generation of GA. The averaged fitness distributions for each generation of GA are available written in either frequencies table or in histogram with predefined number of categories.

2) Data that describes GA behavior. This includes the dynamics of averaged mean, the best and the worst fitness value and also the dynamics of averaged square deviation of fitness in population. Also the information about time per run in milliseconds is output to estimate GA performance.

3) Data that describes obtained solutions. There is the number of object function calculations until solution is found and the time costs in milliseconds. All solutions and some data describing additional properties of the solutions if there are any are output into separate file for further analysis and use.

VII. Conclusion

Results of experiments showed that NEvA performance is comparable with results of other algorithms for the reviewed problems (XOR problem and full information pole balancing). Due to adaptive selection of the type of mutation there is no need in direct global limiting variables or deterministic rules for network structure evolution, because addition or deletion of the new elements performed depending on individual peculiarity of each phenotype. Adaptive mutation rate caused increase of performance in comparison with NEvA with fixed P_m (up to 40% for 2-pole balancing task), although resulting networks were slightly worse for 1-pole balancing problems. Use of mutation that deletes connections can help to get rid of insignificant network inputs thus decreasing a search space complexity.

References

[1] J. Holland, *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[2] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, *Parallel Distributed Processing, Explorations in the microstructure of cognition*. Vol 1, Foundations. Cambridge, MA: MIT Press. 1986.

[3] Whitley D, "Genetic algorithms and neural networks," in *Genetic Algorithms in engineering and computer science*. Winter, Periaux, Galan and Cuesta, Eds. John Wiley, 1995. pp. 203-216.

[4] K. Balakrishan and V. Honavar "Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature," Technical Report CS TR #95-01, Iowa State University, 1995. http://www.cs.iastate.edu/~gannadm/Bib/TR95-01.ps.

[5] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 2, pp. 99-127, 2002.

[6] M. Nowak and P. Schuster, "Error thresholds of replication in finite populations: Mutation frequencies and the onset of Muller's ratchet," *Theoretical Biology*, vol. 137, pp. 375-395, 1989.

[7] Wieland A., "Evolving neural network controllers for unstable systems," *Proceedings of the International Joint Conference on Neural Networks, USA*, 1991.

[8] F. Gomez and R. Miikkulainen. "2-D Pole Balancing Benchmark with Recurrent Evolutionary Networks," *Proceedings of the International Conference on Artificial Neural Networks (ICANN–98), Sweden*, 1998.

[9] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, vol. 5, pp. 317–342, 1997.

[10] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65-85, 1994.

[11] N. Saravanan and D. B. Fogel, "Evolving neural control systems," *IEEE Expert*, pp. 23-27, 1995.

[12] D. James and P. Tucker, "A Comparative Analysis of Simplification and Complexification in the Evolution of Neural Network Topologies," *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2004), USA*, 2004.

[13] Y. Tsoy and V. Spitsyn, "Use of Design Patterns for Design of the Software Environment for Researhces in Genetic Algorithms," *Proceedings of the 8-th Korea-Russia International Symposium on Science and Technology KORUS-2004. Tomsk*, 2004.

[14] E. Gamma, R. Helm, R. Johnson and J. Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, Massachusetts: Addison-Wesley, 1995.

[15] Y. Tsoy and V. Spitsyn, "A Compensatory Genetic Algorithm," *Proceedings of the 10-th Jubilee international conference "Modern technique and technologies", Tomsk*, 2004.