Mutation Control in Neat for Customizable Neural Networks

Galchenkov Oleg,
Ph.D., Senior Lecturer
Odesa National Polytechnic University, Ukraine, Odesa
o.n.galchenkov@gmail.com

Nevrev Alexander,
Ph.D., Senior Lecturer
Odesa National Polytechnic University, Ukraine, Odesa
a.i.nevrev@gmail.com

A modification of the NEAT algorithm is proposed through the use of a mechanism for changing the level of mutations. At the same time, both the weight coefficients and the number of neurons in the hidden layers and the connections between them change. For the task of modelling the XOR operator, a significant acceleration of the training of a neural network was obtained.

Keywords: *neural network, genetic algorithm, fitness function, weighting factors, mutation, selection*

In traditional approaches to the use of neural networks, the network topology is usually selected and fixed before it is trained [1]. And the training itself consists of adjusting weights to ensure the extremum of the quality function. To select the neural network topology that is most suitable for a particular task, they try to train neural networks with various topology options and ultimately choose the one that shows the best values of the quality function or requires the same amount of computation for the same values of the quality function. Obviously, such a path is extremely costly in the computational sense and does not provide the best version of the topology due to the very large number of possible options. Therefore, it seems promising to develop and study algorithms that simultaneously configure both weighting factors and the topology of the neural network itself.

Various versions of training algorithms for neural networks with the simultaneous adjustment of both weights and network topology began to appear in the 90s of the last century [2-5] and continue to be developed to date [6-9].

The main issues encountered in the development of such algorithms are the following [6]:

- how to most effectively present variants of the neural network topology to be able to select the most useful elements and filter out the rest,

- how can a topological innovation be protected on the interval of the number of training iterations necessary to adjust its weight coefficients and manifest in the value of the quality function,

- how to ensure minimal neural network topology without using special functions that measure the complexity of the current configuration.

Among the algorithms aimed at solving these issues, one of the basic ones is the NEAT (NeuroEvolution of Augmenting Topologies) genetic algorithm [6]. All new modifications represent an improvement on this algorithm. As a rule, all new ideas can be used for this algorithm at the same time. Therefore, in this work, when developing a new modification of the NEAT algorithm, it seems appropriate to use it as a base one as well.

The entire configuration of the neural network in the form of a linear representation of neurons and the connections between them is used as the genome in the NEAT algorithm. The initial conditions for the genome are set in the form of a minimal configuration of the neural network and its size increases as it learns. To control the expandability of the network, each element of the genome (neurons and the connections between them) is assigned an innovative number that allows you to take into account the moment this element appears in the genome and, accordingly, the number of iterations of the learning algorithm during which this element is part of the given genome. The presence of an innovative number allows us to differentiate elements of genomes (genes) depending on the history of their appearance in the genome. Genes are called matching if they have the same innovation numbers. If the numbers are different, then the genes are called disjoint if the number of one lies in the range of the number of innovations in the genome of the other gene, or redundant if the number of one is outside this range. When a new genome is formed from matching genes, one of them is randomly taken from any parent, and redundant and disjoint genes are taken from the parent with a large value of the quality function (fitness function). Thus, the NEAT algorithm produces a crossover without using the function of assessing the complexity of the current topology and is aimed at ensuring the predominance of genes with a positive effect over random genes.

Since the introduction of new structural elements in the short term usually worsens the importance of the fitness function, and a positive effect is achieved after appropriate weighting, the NEAT algorithm uses innovative numbers to form niches for genomes with structural innovations and having similar topologies. Moreover, over a given number of iterations, genomes compete with each other only within niches, and not with the entire set of genomes. This allows you to protect new topological elements while adjusting the weight coefficients of these elements. The network structure gradually increases as structural mutations occur, leading to a better value of the quality function.

The aim of this work is the further development of the NEAT algorithm to ensure a higher learning speed and achieve the resulting structure of a neural network with a small number of neurons and the connections between them.

The NEAT algorithm is based on the ideology of the genetic approach [10] and, accordingly, involves the following steps [6]:

1) Generation of an initial population of individuals;

2) The calculation for each individual of its suitability with the help of a fitness function (quality function);

3) Removal of weak individuals;

4) Conducting a crossover of the best individuals in order to produce a new population;

5) Carrying out mutations of descendants due to changes in weighting factors, adding/removing connections between neurons, adding/removing neurons;

6) Repeat steps 2-5 until the desired value of the quality function is reached.

Elite selection is used to remove weak individuals. He assumes that there remains a fixed percentage of the best individuals, and the rest are discarded.

The NEAT algorithm assumes three types of mutations.

1. Mutations of weights (Fig. 1)

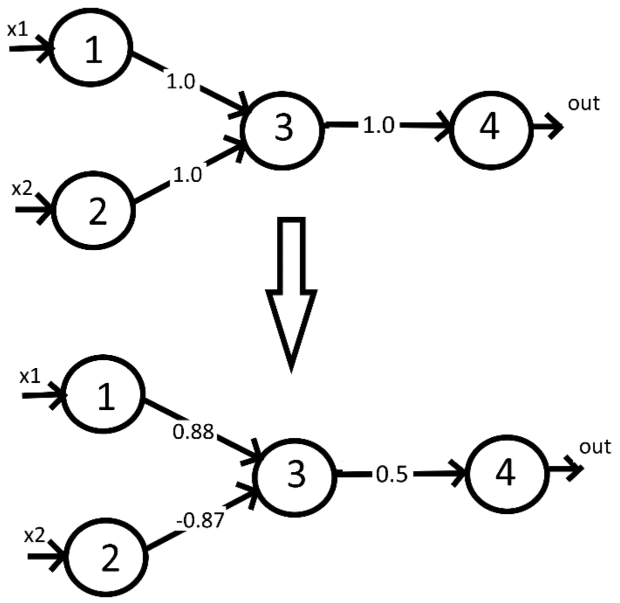

Fig. 1. Change in weights in existing relationships.

2. Removing/adding connections between neurons (Fig. 2).

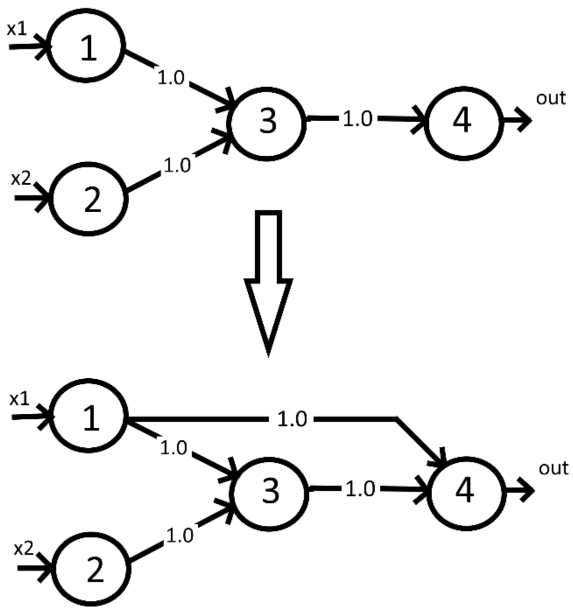

Fig. 2. Adding a connection between existing neurons.

3. Addition / removal of neurons (Fig. 3)

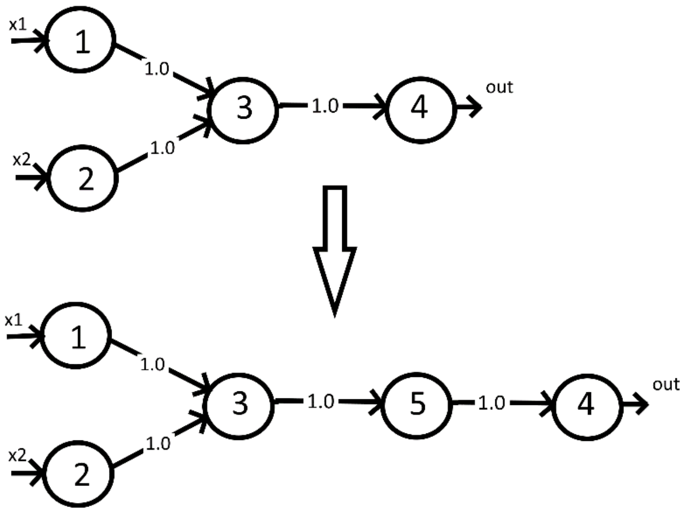

Fig. 3. Adding a neuron to the network

The variety of mutations used determines the complex nature of the movement of the learning algorithm on the surface of the quality function. The essence of the proposed modification of the NEAT algorithm is to add a mechanism for changing the intensity of mutations. It can be configured using the following four factors:

1. mutation_intensity - initial (and, subsequently, the current value of the intensity of mutations);

2. mutation_intensity_max - a maximum allowable value of intensity;

3. mutation_intensity_min - the minimum acceptable value of intensity, which is used when resetting the current mutation, when the network exceeded the previous value of the fitness function;

4. mutation_intensity_step - step of the mutation intensity, by which the current intensity increases if the network during mutations remains at the same value of the fitness function.

The number of mutations that will be carried out on one individual will vary from 1 to the value of mutation_intesity, that is, the current intensity of mutations. If the network is stuck in the learning process, the mutation intensity will slowly increase, thereby increasing the number of mutations per generation in order to accelerate the growth of the network. As soon as the network reaches a structure of higher complexity, and the value of the fitness function begins to increase again, the intensity of mutations is reset to the minimum value, the speed of mutations decreases so that the network does not create random extra connections and neurons.

We will verify the performance of the modified NEAT algorithm by the example of modelling the logical operation of "exclusive OR" (XOR) [2]. The truth table of the XOR operator is shown in table 1.

Table 1

X1	X2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

The function was used as a fitness function.

$$f = f_{00} + f_{01} + f_{10} + f_{11}$$

where $f_{ij} = \min\left(\frac{1}{\left(ans_{ij} - out_{ij}\right)^2}, 50 \right)$

ans_{ij} - the response of the neural network to the combination of ij at the inputs,

out_{ij} is the correct answer for the XOR operator when combining ij to the entrance.

Figure 4 shows the change in the structure of the neural network as it is trained by the modified NEAT algorithm and the corresponding values of the fitness function (fitness parameter).

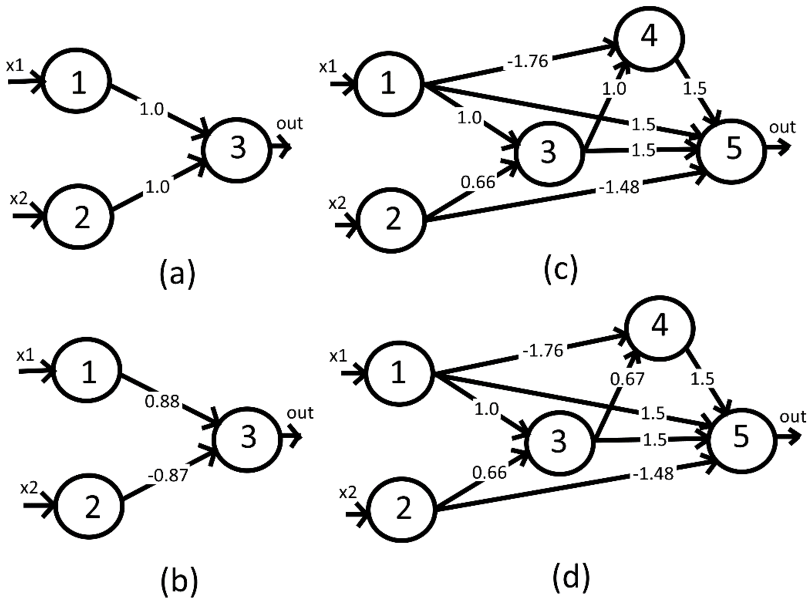

Fig. 4 - The structure of the neural network: a - the starting structure, b - for fitness = 150, c - fitness = 172, d - fitness = 200.

From a comparison of the configurations of the neural network at different stages of training, it can be seen that weights are first set up, and when the increase in the value of the fitness function slows down significantly, new neurons and connections are added. After that, setting the weights again becomes a priority.

To obtain averaged characteristics, the neural network learning process was run 10 times for the NEAT algorithm and the modified NEAT algorithm. The corresponding numbers of populations on which full network training was achieved in each of the implementations are shown in Figs. 5 and 6.

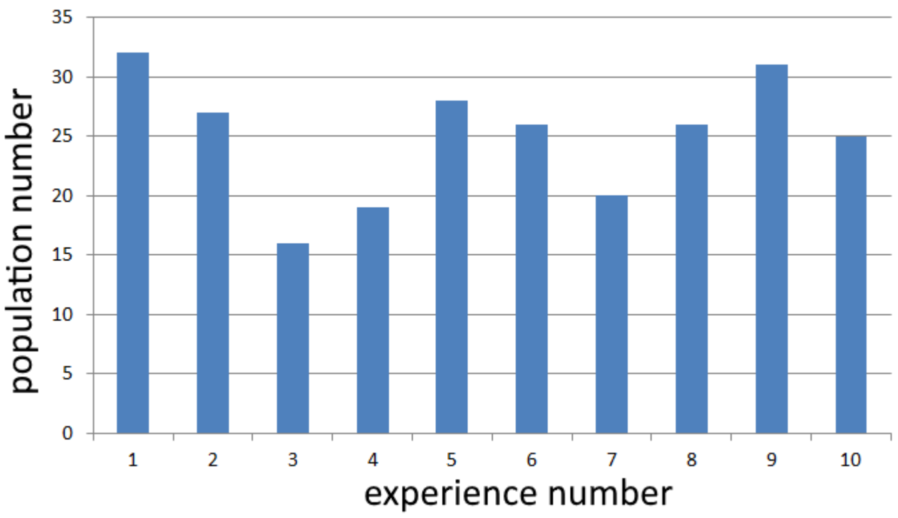

Fig. 5 - The number of the population on which the network is fully trained in modelling the XOR operator for each of the implementations when learning by the NEAT algorithm.

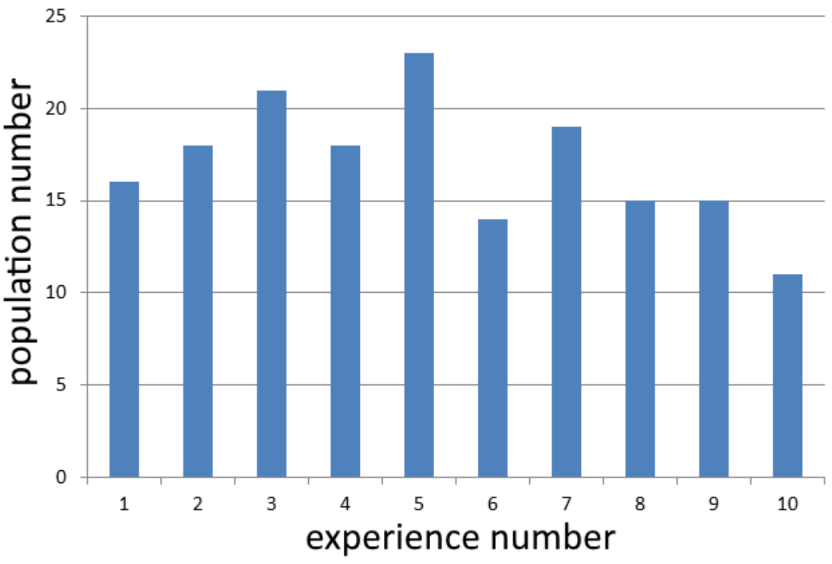

Fig. 6 - The number of the population on which the network has been fully trained in modelling the XOR operator for each of the implementations when training with the modified NEAT algorithm.

A comparison of the figures shows that the modified algorithm achieves complete network learning for a smaller number of populations. Since the work of learning algorithms begins with random initial conditions and uses random number generators when performing crossover and mutation operations, the NEAT algorithm in some experiments also required a small number of populations. Figure 7 shows the learning curves averaged over 10 implementations for the NEAT algorithm and the modified NEAT algorithm when the neural network simulates the operation of the XOR operator.

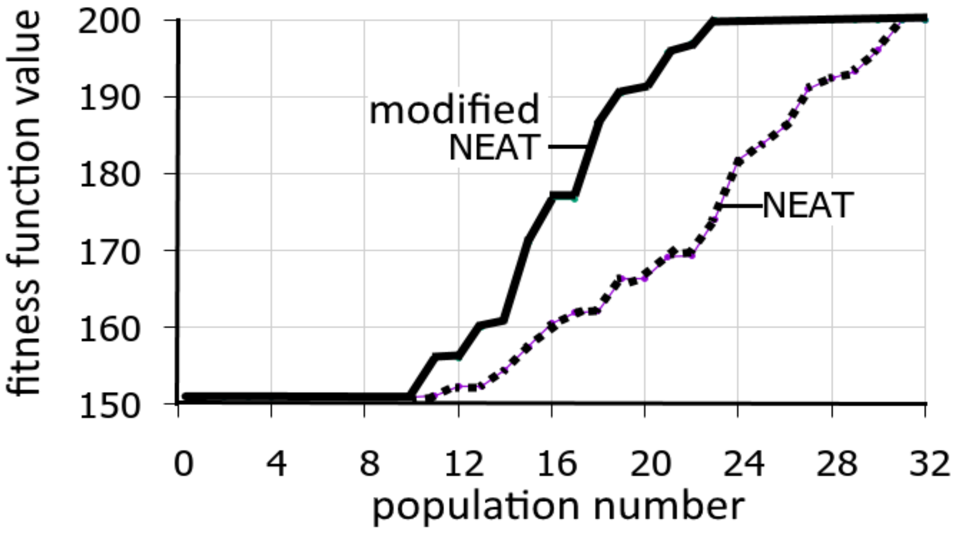

Fig. 7 - Comparison of the learning speed of the neural network when modelling the XOR operator with the NEAT algorithm and the modified NEAT algorithm

A comparison of the learning curves shown in Figure 7 shows a significant advantage of the modified version of the algorithm.

Usually, in practice, two processes are distinguished - training and the use of a neural network. We can train many times on a special powerful computer. But, as a rule, it will be necessary to use it on another calculator, which is either less powerful or loaded with other tasks. Therefore, in practice, you need to make a certain amount of training and take the minimum configuration to use. Based on this, it is possible to compare the average number of experiments with their generation of populations in each (more precisely, the total number of generations) that must be performed in order to obtain the minimum configuration. Figure 8 shows the average number of generations needed to find a solution depending on the intensity of mutations, and Figure 9 shows the average number of generations needed to find the maximum of the quality function and the minimum configuration depending on the intensity of mutations.

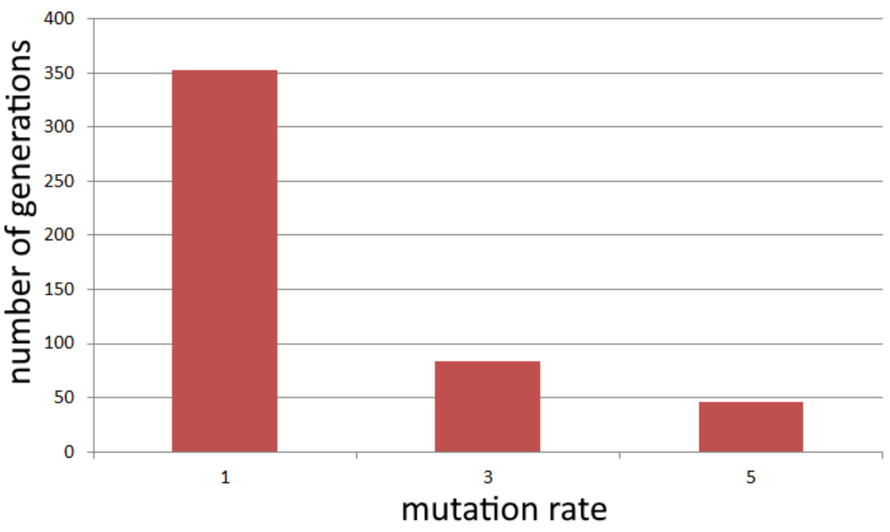

Fig. 8 - The average number of generations required to find a solution depending on the intensity of mutations

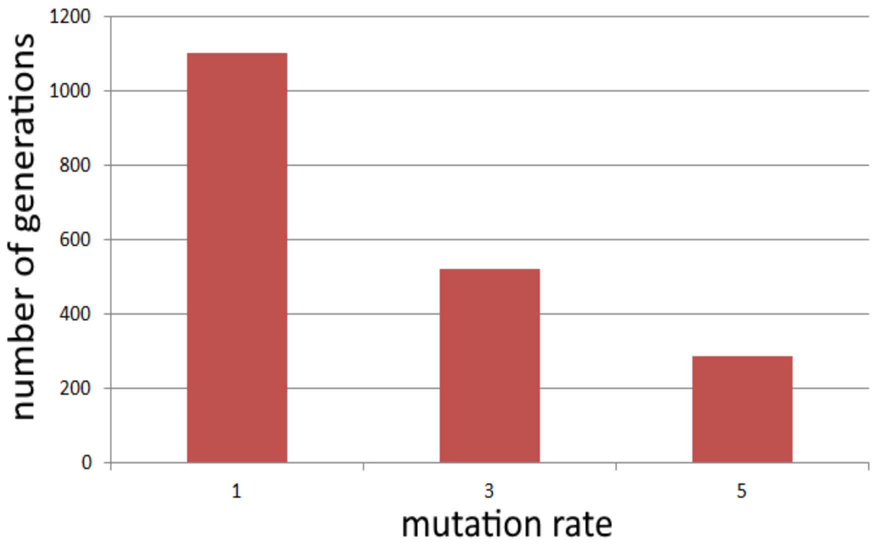

Fig. 9 - The average number of generations required to find a solution and the minimum configuration depending on the intensity of mutations

Figures 8 and 9 show that with an increase in the number of mutations from 1 to 5, the average number of generations of new populations that need to be done to find the maximum of the quality function without paying attention to the size of the resulting neural network decreases by more than 7 times. Naturally, to achieve the minimum configuration of a neural network, it is necessary to produce an average of 3-4 times more generations. Figure 9 shows that with an increase in the number of mutations from 1 to 5, in order to find the minimum configuration, an average of 4 times less is the generation of populations.

Conclusions

Additional use of the mechanism for controlling the intensity of mutations in the NEAT algorithm can significantly increase the learning speed of the neural network in the task of modelling the XOR operator.

This, in turn, leads to a decrease in the total amount of computation required to fully configure the neural network and achieve a minimum configuration. However, due to the very large search space when solving complex problems by the NEAT algorithm, further development of this algorithm is necessary for the direction of increasing the learning speed and parallelizing operations for implementation on multiprocessor computers.

References

1. Goodfellow I., Bengio Y., Courville A. Deep Learning, MIT Press, 2016, http://www:deeplearningbook:org.
2. Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1993). An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5:54–65.
3. Braun, H. and Weisbrod, J. (1993). Evolving feedforward neural networks. In Albrecht, R. F., Reeves, C. R., and Steele, N. C., editors, Proceedings of ANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms, pages 25–32, Springer-Verlag, Innsbruck.
4. Dasgupta, D. and McGregor, D. (1992). Designing application-specific neural networks using the structured genetic algorithm. In Whitley, D. and Schaffer, J. D., editors, Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks, pages 87–96, IEEE Press, Piscataway, New Jersey.
5. Opitz, D. W. and Shavlik, J. W. (1997). Connectionist theory refinement: Genetically searching the space of network topologies. Journal of Artificial Intelligence Research, 6:177–209.
6. Stanley K., Miikkulainen R., Evolving Neural Networks through Augmenting Topologies – 2002 by the Massachusetts Institute of Technology .-Evolutionary Computation 10(2): 99-127, http://nn.cs.utexas .edu/downloads/papers/stanley.ec02.pdf
7. Miconi, T. (2016).Neural networks with differentiable structure. Preprint at https://arxiv.org/abs/1606.06216
8. Salimans, T., Ho, J., Chen, X., Sidor, S. & Openai, I. S. (2017). Evolution strategies as a scalable alternative to reinforcement learning. Preprint at https://arxiv.org/abs/1703.03864

9. Bhushan, S. Hybrid approach to energy efficient clustering for heterogeneous wireless sensor network using biogeography based optimization and k-means / Bhushan S., Antoshchuk S., Teslenko P., Kuzmenko V., Wojcik W., Mamyrbaev O., Zhunissova U., // Przeglad Elektrotechniczny. 2019. Book: 95 Issue 4 PP. 138-141 DOI: 10.15199/48.2019.04.24

10. Mitsuo Gen, Runwei Cheng. Genetic Algorithms and Engineering Optimization .- John Wiley & Sons, Inc.- 2000.-511P.