WCCI 2012 IEEE World Congress on Computational Intelligence
June, 10-15, 2012 - Brisbane, Australia IEEE CEC

Self-Configuring Genetic Programming Algorithm with Modified Uniform Crossover

Eugene Semenkin
Institute of computer science and telecommunication
Siberian State Aerospace University
Krasnoyarsk, Russia
eugenesemenkin@yandex.ru

Maria Semenkina
Institute of computer science and telecommunication
Siberian State Aerospace University
Krasnoyarsk, Russia
semenkina88@mail.ru

Abstract— **For genetic programming algorithms new variants of uniform crossover operators that introduce selective pressure on the recombination stage are proposed. Operators probabilistic rates based approach to GP self-configuration is suggested. Proposed modifications usefulness is demonstrated on benchmark test and real world problems.**

Genetic programming; uniform crossover; selective pressure recombination; self-configuration; symbolic regression; classification

I. Introduction

Evolutionary algorithms (EA), the best known representatives of which are genetic algorithms (GA) and genetic programming (GP), are information processing techniques based on the principles of natural evolution. Although EAs have been successful in solving many optimization and modeling problems, the performance of this technique depends on the selection of the EA settings and parameters tuning. Moreover, the process of settings determination and tuning parameters is known to be a time-consuming and a complicated task. Much research has attempted to deal with this problem. Some approaches attempted to determine appropriate settings by experimenting over a set of well-defined functions or through theoretical analysis. Other approaches, usually applying a term such as "self-adaptation", try to eliminate the setting determination process by adapting settings through the algorithm execution.

There exists much research devoted to "self-adapted" EA based on a range of ideas, but all of them aimed at reducing the role of human expert in algorithm designing.

Let us follow definitions given by Gabriela Ochoa and Marc Schoenauer, organizers of the workshop "Self-tuning, self-configuring and self-generating evolutionary algorithms" (Self* EAs) within PPSN XI [1]. According to this definition, "*… The following 3 general paths toward automated heuristic design will be distinguished: 1) tuning: the process of adjusting the control parameters of the algorithm; 2) configuring: the process of selecting and using existing algorithmic components (such as variation operators); 3) generating: the process of creating new heuristics from existing sub-components borrowed from other search methods…*". As the main ideas of the algorithms proposed here rely on automated "selecting and using existing algorithmic components", these algorithms might be called self-configuring. Within this, the probabilities with which will be the genetic operators used are subject to the tuning. This allows us to say that the algorithms are partially self-tuning. Our algorithms are also related to the third path, as we use heuristics based on introducing selective pressure during crossover in GA and "borrow" this idea for including in GP.

In order to specify our algorithms more precisely, one can say that according to the classification [2], broadened in [3], we use dynamic adaptation on the population level ([4]). The probabilities of genetic operators to be applied are modified "on the fly" as the algorithm executes. According to the classification given in [5] we use centralized control techniques (central learning rule) for parameters settings with some differences from the usual approaches. Operator rates (the probability to be chosen for generating off-spring) are adapted according to the relative success of the operator during the last generation independently of the previous results. That is why our algorithm avoids problem of high memory consumption typical for centralized control techniques [5]. In some cases it results in situation where no operators have success (off-spring fitness improvement) but nevertheless some of them (worked badly but better than others) will be rewarded. Operators' rates are not included in individual chromosomes and they are not subject to the evolutionary process. All operators can be used during one generation for producing off-springs one by one. The time complexity of the operator productivity is not increased with the number of generations used but it is increased according to the size of the population.

Having conducted numerical experiments we found that the proposed approach deserves special attention and further development and investigation.

The rest of the paper is organized in the following way. Section 2 explains briefly the idea of selective pressure on the stage of individuals crossover in GA and describes the previous results of the algorithm performance investigation, Section 3 introduces a uniform crossover operator for GP and shows algorithm performance investigation on the symbolic regression test problems, Section 4 describes the proposed method for EA self-configuring for GA and GP and testing results, Section 5 describes results of numerical experiments comparing the performance of the proposed approach with known algorithms, and in the Conclusion section we discuss the obtained results.

II. Uniform crossover operator with selective pressure for genetic algorithms

The uniform crossover operator is known as one of the most effective crossover operators in conventional genetic algorithm [6, 7]. Moreover, nearly from beginning, it was suggested to use a parameterized uniform crossover operator and it was proved that tuning this parameter (the probability for a parental gene to be included in the off-spring chromosome) one can essentially improve "The Virtues" of this operator [7] that allows it to emulate other crossover operators. Nevertheless, in the majority of cases using uniform crossover the mentioned possibility is not adopted and the probability for a parental gene to be included in an off-spring chromosome is given equal to 0.5 [8, 9].

That is why it seemed interesting to us to modify the uniform crossover operator with a purpose of improving its performance. Having a desire to avoid real number parameter tuning, we introduced selective pressure during process of recombination [10] making the probability of a parental gene being passed to off-spring depended on parent fitness values. Like the usual GA selection operators, fitness proportional, rank-based and tournament-based uniform crossover operators were added to the conventional operator which we now call the equiprobable uniform crossover. It is evident that tournament-based uniform crossover can be activated only in the case of multiparent recombination with 3 and more parents for one off-spring.

The performance of conventional GA with additional uniform crossover operators and multiparent recombination was evaluated on the usual test optimization problems (see Appendix) as well as on the problem of artificial neural network (ANN) automated design (the choice of the structure and weights tuning) and on the problem of tuning parameters for symbolic formula generated with GP algorithm [10]. After multiple runs and statistical processing of the results, the following observations were found (in terms of algorithm reliability). On the test problems, 7-parents recombination was the best one with 2-parents recombination being the second-best. The average best crossover operator is a rank-based uniform crossover, conventional equiprobable uniform crossover is the second best. Tournament-based uniform crossover is better than the fitness proportional uniform crossover and 1-point and 2-points crossovers. On the last two problems devoted to ANN and GP there was no statistically significant difference between 7- and 2-parents recombinations as well as between equiprobable and rank-based uniform crossovers.

Partial illustration of the test experiment results is given in the first six rows of the table 1 below. Table 1 contains average, minimal and maximal reliability of algorithms averaged on the first 14 test problems from the Appendix, each solved 1000 times. The reliability is the portion of the runs for a given algorithm that gives satisfactorily precise solution. In the table 1, row headers "1-point, 2-point, UE, UT, UP, UR" indicate type of crossover, respectively, 1-point, 2-point, uniform equiprobable, uniform tournament-based, uniform fitness proportional and uniform rank-based crossovers. "Average", "min" and "max" mean, correspondingly, average, minimal and maximal reliability of algorithms over 1000 runs. Numbers in the cells indicate the reliability averaged over 14 test functions. The numbers in the brackets show the range of the corresponding indicator. The first number in the brackets is the minimal value among 14 tests, the second number is the maximal value among 14 tests.

The meaning of the last row will be described later in the section 4.

TABLE I. Reliability of GAs over 14 test optimization problems

Crossover	Average	Min	Max
1-point	0.760 [0.507,0.915]	0.696 [0.411,0.856]	0.822 [0.591,0.978]
2-point	0.479 [0.132,0.821]	0.413 [0,0.754]	0.534 [0.167,0.871]
UE	0.819 [0.442,0.953]	0. 780 [0.589,0.887]	0.878 [0.669,0.999]
UT	0.627 [0.354,0.967]	0. 587 [0.299,0.917]	0.697 [0.38,**1**]
UP	0.647 [0.276,0.935]	0.622 [0.232,0.888]	0.718 [0.3,0.976]
UR	**0.833** [0.598,0.974]	0.771 [0.578,0.935]	0.888 [0.635,0.999]
SelfCGA	**0.928** [0.83, **1**]		

III. Uniform crossover operator with selective pressure for genetic programming algorithms

We use a tree representation in our genetic programming algorithms. To apply the crossover operator with selective pressure in GP, that was useful in GA, we first have to specify the uniform crossover operator for GP. Such an operator has been theoretically substantiated and introduced by Poli and Langdon in 1998 ([11, 12]). Later they also proposed a problem-specific modification of this operator [13, 14]. See, also, the recent description in [15].

According to [12], the GP uniform crossover process starts at the tree's root node and works its way down each tree along some path until finding function nodes of differing arity at the similar location. Furthermore it can swap every node up to this point with its counterpart in the other tree without altering the structure of either. Any node in one tree having a corresponding node at the same location in the other is said to be located within the common region. Those pairs of nodes within the common region that have the same arity are referred to as interior. The common region necessarily includes all interior nodes. Once the interior nodes have been identified, the parent trees are copied. Interior nodes are selected for crossover with some probability which is generally set to 0.5. Crossover involves exchanging the selected nodes between the trees, while those nodes not selected for crossover remain unaffected. Non-interior nodes within the common region can also be crossed, but in this case the nodes and their subtrees are swapped.

We organize the uniform crossover in slightly different way. After parents selection the uniform crossover operator produces one off-spring choosing nodes from parental trees with the given probability. Crossing over also has a place in the situation when parental nodes contain functions with different arity because all arguments compete with each other. E.g., if one node is "SIN – X" (sin(x)) and other one is "/ – Y – Z" (y/z) and the off-spring inherits the node "/" then resulting subtree can be one of "y/z", "x/z" or "y/x". If the off-spring inherits the node "SIN" then resulting subtrees can be "sin(x)", "sin(y)", "sin(z)". Other aspects of the uniform crossover operator are the same as described in [11]. The described modification brings more flexibility to the crossover process and allows the potential for a change in the algorithms behavior. Recall that uniform crossover allows multiparent recombination.

Having the appropriate uniform crossover operator we can introduce now a selective pressure in the same way as we did for the genetic algorithm in Section 2. The off-spring can inherit everyone of its nodes from one of parents not only equiprobably but also with different probabilities determined by parent fitness values in one of the ways mentioned above: fitness proportionally, according to ranks or through tournament.

We have implemented the described approach and conducted numerical experiments to evaluate the usefulness of developed operator. As a commonly accepted benchmark for GP algorithms is still an "open issue" [16], we used the symbolic regression problem with 17 test functions given in the Appendix for preliminary evaluation.

Experiments settings are 100 individual, 300 generations and 100 algorithm runs for each test function.

TABLE II. PERFORMANCE OF GPS OVER 17 TEST SYMBOLIC REGRESSION PROBLEMS

	Average	**Min**	**Max**	**Average generations variance**
GP	0.43 [0.13, 0.77]	0.12 [0, 0.41]	0.60 [0.27, 0.91]	[33, 289]
MGP	0.53 [0.31, 0.88]	0.31 [0.11, 0.56]	0.75 [0.43, 1]	[27, 243]
SelfCGP	0.69 [0.42, 1]			[49, 201]

The experiment results are given in the first two rows of Table 2 for conventional GP and GP with new modified uniform crossover operator (MGP). The first three columns concern the reliability, i.e. proportion of 100 runs when approximation with sufficient precision was found. This reliability was averaged first over all parameter settings given averaged reliabilities for every test problem. Their minimum and maximum are given in brackets in the first column. The averaged value over 17 problems is above these brackets. The next two columns contain information on the reliability of the worst and the best settings averaged over 17 problems (upper number) and its range (numbers in brackets).

The last column of table 2 contains information on the necessary computational efforts that is necessary for finding a sufficiently exact approximation averaged over the 100 runs of each of algorithm settings combination. The variance is given over 17 test functions.

TABLE III. SOLUTIONS QUALITY OF GPS OVER 17 TEST SYMBOLIC REGRESSION PROBLEMS

	% precise solutions	**% cond. precise solutions**	**% approx. solutions**
GP	50	16	34
MGP	58	20	22
SelfCGP	58	16	26

The columns in Table 3 contain information on the quality of the obtained approximations. The first shows the percentage of precise solutions symbolically identical to the test function. The second one shows the percentage of conditionally precise solutions that needed some elementary transformations and numbers rounding to be symbolically identical to the test function. The third column shows the percentage of the obtained solutions which cannot be transformed into a symbolically identical form.

From table 2 and table 3 we can see that GP with modified uniform crossover operators is slightly better than conventional GP. This demonstrates that proposed operators may be useful. Additional observations are almost the same as for GA. The best number of parents is 7, second best is 2 and the difference between both is not so big. The best crossover operator is rank-based uniform crossover, second best is equiprobable uniform crossover operator.

Although proposed new operators give higher performance than conventional operators, at the same time it increases the number of algorithm setting variants that complicates algorithms adjusting for end user, particular a non-expert in GP. That is why it is necessary to suggest a way eliminating extra efforts needed for adjustment.

IV. OPERATORS' RATES BASED SELF-CONFIGURATION OF ALGORITHMS

As mentioned above, we apply operators' probabilistic rates dynamic adaptation on the level of population with centralized control techniques. To avoid the issues of precision caused while using real parameters, we used setting variants, namely types of selection, crossover, population control and level of mutation (medium, low, high). Each of these has its own probability distribution, e.g., there are 5 settings of selection – fitness proportional, rank-based, and tournament-based with three tournament sizes. During initialization all probabilities are equal to 0.2 and they will be changed according to a special rule through the algorithm's execution in such a way that a sum of probabilities should be equal to 1 and no probability could be less than predetermined minimum balance. The "idle crossover" is included in the list of crossover operators to make crossover probabilities less than 1 as is used in conventional algorithms to model a "childless couple".

When the algorithm creates the next off-spring from the current population it first has to configure settings, i.e. to form the list of operators with using the probability operator distributions. The algorithm then selects parents with the chosen selection operator, produces an off-spring with the chosen crossover operator, mutates off-spring with the chosen mutation probability and puts off-spring into the intermediate population. When the intermediate population is filled, the fitness evaluation is computed and the operator rates (probabilities to be chosen) are updated according to operator productivities. Then the next parent population is formed with the chosen survival selection operator. The algorithm stops after a given number of generations or if the termination criterion (e.g., given error minimum) is met.

The productivity of an operator is the ratio of the average off-spring fitness obtained with this operator and the average fitness of the overall off-spring population. Successful operators having productivity more than 1 increase their rates obtaining portions from unsuccessful operators. There is no necessity to consume extra computer memory to remember past events and updates are more dynamic (that can be both a plus and a minus).

The described approach can be used both in GA and GP as well as in other EA techniques.

Results of self-configuring GA (SelfCGA) performance evaluation over 14 test optimization problems are given in the last row of table 1 given in Section 2 above.

As this algorithm has no adjusted settings there is only one column for it. The worse reliability (for the hardest problem) averaged over 1000 runs was equal to 0.830. The best reliability was equal to 1.00. SelfCGA reliability averaged over 14 test function is better than the averaged best reliability of other algorithms. It demonstrates the usefulness of the proposed modifications. The main advantage of SelfCGA is that there is no need in algorithm adjustment and yet there is no loss in performance. This makes this algorithm useful for many applications where there is no possibility to choose settings (e.g., ANN weight coefficients tuning or GP generated symbolic formulations parameters adjustment) as well as in cases when end users are not experts in evolutionary optimization, but nevertheless intend to apply GA for solving optimization problems in hand.

Results of self-configuring GP (SelfCGP) performance evaluation over 17 test problems are given in the last rows of tables 2 and 3. As this algorithm has no adjusted settings there is one only column for it. The worse reliability (for the most hard problem) averaged over 100 runs was equal to 0.42. The best reliability was equal to 1.00. SelfCGP reliability averaged over 17 test function is better than averaged best reliability of conventional GP and slightly less than best reliability of modified GP. Computational efforts are also better than alternative algorithms. It gives us a possibility to recommend SelfCGP for solving symbolic regression problems as better alternative to conventional GP. Main advantage of SelfCGP is no need of algorithmic details adjustment without any losses in the performance that makes this algorithm useful for many applications where end users being no experts in evolutionary modeling nevertheless intend to apply GP for solving these problems.

In this section, we observed usefulness of proposed modifications of GA and GP. All three variants of GPs used SelfCGA for tuning parameters of obtained formulations that saved us of the doubts whether made we mistake with proper adjustment of tuning algorithm. However, we fulfill all our experiments with the test problems and some of them seemed to be hard but nevertheless more like to a toy. Now we have to check whether proposed algorithm can compete with other existing approaches in solving really hard problems.

V. PERFORMANCE COMPARISON OF SELF-CONFIGURING GENETIC PROGRAMMING AND ALTERNATIVE CLASSIFICATION METHODS

We have solved above some test symbolic regression problems with the proposed modified GP algorithms and demonstrated competitive results with conventional GP. In this section we intend to solve two hard classification problems and compare our results with alternative approaches. Classification problems will be solved by construction of separating surface by means of symbolic regression formulation.

The first data set, called the German Credit Data Set, includes customer credit scoring data with 20 features, such as age, gender, marital status, credit history records, job, account, loan purpose, other personal information, etc. There are 700 records judged to be creditworthy and 300 records judged to be non-creditworthy. The second data set includes Australian credit scoring data with 307 examples of the creditworthy customers and 383 examples for the non-creditworthy customers. It contains 14 attributes, where six are continuous attributes and eight are categorical attributes. Both data sets are made public from the UCI Repository of Machine Learning Databases [17], and are often used to compare the accuracy with various classification models.

For our experiments we used 100 individuals and maximum 500 generations over 20 runs. For each run the data sets were randomly divided into train and validation subset with proportion 70% - 30%. Results were averaged. Both our algorithms, GP with modified uniform crossover (MGP) and self-configuring GP (SelfCGP), used for classification the symbolic regression solutions with SelfCGA as the tool for numerical coefficients adjustment.

Results for alternative approaches have been taken from scientific literature. In [18] the performance evaluation results for these two data sets are given for authors' two-stage genetic programming algorithm (2SGP) as well as for the following approaches taken from other papers: conventional genetic programming (GP), multilayered perceptron (MLP), classification and regression tree (CART), C4.5 decision trees, k nearest neighbors (k-NN), linear regression (LR). Additional material for comparison we have taken from [19] where are evaluation data for authors' automatically designed fuzzy rule based classifier as well as for other approaches found in literature: Bayesian approach, boosting, bagging, random subspace method (RSM), cooperative coevolution ensemble learning (CCEL).

The results of comparison (proportion of correctly classified objects in validation set) are given in table 4.

Numbers highlighted in bold in table 4 are the best results in our comparison. The winner is 2SGP from [18], the algorithm specially designed for credit scoring problems solving. SelfCGP is the second best one for both problems proposed in this paper. Another algorithm, MGP, has 5th place of 15 for Australian credit and 3rd for German credit.

TABLE IV. THE COMPARISON OF THE CLASSIFICATION ALGORITHMS (PROPORTION OF CORRECTLY CLASSIFIED INSTANCES)

	Australian credit	**German credit**
SCGP	0.9022	0.7950
MGP	0.8985	0.7875
2SGP	**0.9027**	**0.8015**
GP	0.8889	0.7834
Fuzzy classifier	0,8910	0,7940
C4.5	0.8986	0.7773
LR	0.8696	0.7837
Bayesian approach	0,8470	0,6790
Boosting	0,7600	0,7000
Bagging	0,8470	0,6840
RSM	0,8520	0,6770
CCEL	0,8660	0,7460
k-NN	0.7150	0.7151
CART	0.8744	0.7565
MLP	0.8986	0.7618

We understand the limitation of this comparison, e.g. there is no information on indicators' variation or on computational effort for the compared methods. Some results are the best for the given method, others are averaged. Moreover, it is clear that 2SGP and fuzzy classifier are much more useful for decision makers as they give not only a computational expression but also human expert understandable production rules. Our intention was to make it clear whether our approach could give results competitive to alternative techniques and without suggesting the best tool for banking credit scoring. This is why we did not adapt our algorithms to these problems making no modifications for category data and given computational resources to 500 generations (rather than 1000 as in [18]), etc.

We conclude that the self-configuring genetic programming algorithm hybridized with self-configuring genetic algorithm proposed in this paper can produce competitive results, it has the usual drawbacks of any general-purpose technique losing to the problem specific algorithms on corresponding problems, but has the advantage requiring no algorithmic details adjustment.

VI. CONCLUSION

In this paper, special kind of uniform crossover operator that introduces selective pressure on the recombination stage for GA and GP are proposed. They are fitness proportional, tournament-based and rank-based uniform crossover operators similar to well-known selection schemes of evolutionary computations. It is demonstrated on the benchmarks that the use of these operators gives positive impact on GA and GP performance. In particular, rank-based uniform crossover is the best among all types of crossover operators.

Then we presented a modified approach to probability based operator rate assignments for the automated configuration of algorithms that assumes the probabilistic choice of operators for every next off-spring and updating the rates after every generation. This modification also turned to be useful both for GA and GP as was demonstrated with test problems.

And at last, we checked our modifications hybridizing SelfCGA and SelfCGP for solving two hard classification problems that demonstrated the usefulness and perspectiveness of the proposed modifications. We developed the self-configuring GP algorithm that can be used for solving different problems by means of different tools. It seems to be not so hard work to adopt our SelfCGP for automated design and tuning of the artificial neural networks, fuzzy logic based systems, decision trees and other computational intelligence tools including problem-specific ones.

ACKNOWLEDGMENT

The research is partially supported through Governmental contracts № 16.740.11.0742 and № 11.519.11.4002.

REFERENCES

[1] Schaefer R., Cotta C., Kołodziej J., Rudolph G. (Eds.) Parallel Problem Solving from Nature – PPSN XI 11th International Conference, Kraków, Poland, September 11-15, 2010.

[2] Angeline P.J. Adaptive and self-adaptive evolutionary computations. In: Palaniswami M. and Attikiouzel Y. (Eds.) Computational Intelligence: A Dynamic Systems Perspective, pages 152–163. IEEE Press, 1995.

[3] Eiben A.E., Hinterding R., and Michalewicz Z. Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 3(2):124–141, 1999.

[4] Meyer-Nieberg S., Beyer H.-G. Self-Adaptation in Evolutionary Algorithms. In F. Lobo, C. Lima, and Z. Michalewicz (Eds.) Parameter Setting in Evolutionary Algorithm, pp. 47-75, 2007.

[5] Gomez J. Self Adaptation of Operator Rates in Evolutionary Algorithms. In Deb K. et al. (Eds.) GECCO 2004, LNCS 3102, pp. 1162–1173, 2004.

[6] Syswerda G. Uniform crossover in genetic algorithms. In J. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, 1989.

[7] De Jong, K.A., Spears W. On the Virtues of Parameterized Uniform Crossover. In: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, July, 1991.

[8] Haupt R.L., Haupt S.E. Practical genetic algorithms. John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[9] Eiben A.E., Smith J.E. Introduction to evolutionary computing. Springer-Verlag, Berlin, Heidelberg, 2003.

[10] Semenkin E.S., Semenkina M.E. Application of GA with modified uniform recombination operator for automated implementation of intellectual information technologies. In: Vestnik. Scientific Journal of Siberian State Aerospace University named after academician M. F. Reshetnev. – 2007. – Issue 3 (16). – Pp. 27-32. (In Russian, abstract in English).

[11] Poli R., Langdon W.B. On the Ability to Search the Space of Programs of Standard, One-Point and Uniform Crossover in Genetic Programming. – Technical Report CSRP-98-7. - The University of Birmingham (UK), 1998. – 21 p.

[12] Poli R., Langdon W.B. On the search properties of different crossover operators in genetic programming. In J. R. Koza, et al., editors, Genetic Programming 1998:Proceedings of the Third Annual Conference, pages

293–301, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998.

[13] Poli R., Page J. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. In Genetic Programming and Evolvable Machines, 1 (1/2):37-56, 2000.

[14] Poli R., Page J., Langdon W.B. Smooth uniform crossover, sub-machine code GP and demes: A recipe for solving high-order boolean parity problems. In W. Banzhaf, et al., editors, Proceedings of the Genetic and Evolutionary Computation Conference, volume 2, pages 1162-1169, Orlando, Florida, USA, 1999.

[15] Poli R., Langdon W.B., McPhee N.F. A Field Guide to Genetic Programming. - Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

[16] O'Neill M., Vanneschi L., Gustafson S., Banzhaf W. Open issues in genetic programming. In: Genetic Programming and Evolvable Machines (2010) 11:339–363.

[17] Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[18] Huang J.-J., Tzeng G.-H., Ong Ch.-Sh. Two-stage genetic programming (2SGP) for the credit scoring model // Applied Mathematics and Computation, 174 (2006): 1039–1053.

[19] Sergienko R., Semenkin E., Bukhtoyarov V. Michigan and Pittsburgh Methods Combining for Fuzzy Classifier Generating with Coevolutionary Algorithm for Strategy Adaptation. In: 2011 IEEE Congress on Evolutionary Computation (CEC'2011), June 5-8, 2011, New Orleans, LA.

Appendix. Test optimization problems

	Functions	**Interval**
1	$I(x)=0.05(x-1)^2+(3-2.9\cdot e^{-2.77257x^2})(1-\cos(x(4-50\cdot e^{-2.77257x^2})))$	$x\in[-1;1]$
2	$I(x)=1-0.5\cos(1.5(10x-0.3))\cos(31.4x)+0.5\cos(\sqrt{5}\cdot 10x)\cos(35x)$	$x\in[-1;1]$
3	$I(x,y)=0.1x^2+0.1y^2-4\cdot\cos(0.8\cdot x)-4\cdot\cos(0.8\cdot y)+8$	$x,y\in[-16;16]$
4	$I(x,y)=(0.1\cdot 1.5\cdot y)^2+(0.1\cdot 0.8\cdot x)^2-4\cdot\cos(0.8\cdot 1.5\cdot y)-4\cdot\cos(0.8\cdot 0.8\cdot x)+8$	$x,y\in[-16;16]$
5	$I(x,y)=100\cdot(y-x^2)^2+(1-x)^2$	$x_1,x_2\in[-2;2]$
6	$I(x,y)=\dfrac{-10}{0.005\cdot(x^2+y^2)-\cos(x)\cdot\cos(y/\sqrt{2})+2}+10$	$x_1,x_2\in[-16;16]$
7	$I(x,y)=\dfrac{-100}{100(x^2-y)+(1-x)^2+1}+100$	$x_1,x_2\in[-5;5]$
8	$I(x,y)=\dfrac{1-\sin^2(\sqrt{x^2+y^2})}{1+0.001\cdot(x^2+y^2)}$	$x_1,x_2\in[-10;10]$
9	$I(x_1,x_2)=0.5(x_1^2+x_2^2)\left[2*0.8+0.8\cos(1.5x_1)\cos(3.14x_2)+0.8\cos(\sqrt{5}x_1)\cos(3.5x_2)\right]$	$x_1,x_2\in[-2.5;2.5]$
10	$I(x_1,x_2)=0.5(x_1^2+x_2^2)\left[2*0.8+0.8\cos(1.5x_1)\cos(3.14x_2)+0.8\cos(\sqrt{5}x_1)\cos(3.5x_2)\right]$	$x_1,x_2\in[-5;5]$
11	$I(x_1,x_2)=x_1^2\lvert\sin 2x_1\rvert+x_2^2\lvert\sin 2x_2\rvert-\dfrac{1}{(5x_1^2+5x_2^2+0.2)}+5$	$x_1,x_2\in[-4;4]$
12	$I(x_1,x_2)=0.5(x_1^2+x_1x_2+x_2^2)[1+0.5\cos(1.5x_1)\cos(3.2x_1x_2)\cos(3.14x_2)+0.5\cos(2.2x$	$x_1,x_2\in[0;4]$
13	$I(x_1,x_2)=-z(x_1)z(x_2)$, $z(x)=-\dfrac{1}{(x-1)^2+0.2}-\dfrac{1}{2(x-2)^2+0.15}-\dfrac{1}{3(x-3)^2+0.3}$	$x_1,x_2\in[0;4]$
14	$I(x_1,x_2)=z(x_1)+z(x_2)$, $z(x)=-\dfrac{1}{(x-1)^2+0.2}-\dfrac{1}{2(x-2)^2+0.15}-\dfrac{1}{3(x-3)^2+0.3}$	$x_1,x_2\in[0;4]$
15	$I(x_1,x_2)=(x_1-2)^2+(x_2-1)^2$	$x_1,x_2\in[-5;5]$
16	$I(x_1)=\sin(x_1)x_1^2$	$x_1\in[-5;5]$
17	$I(x_1)=\sin(x_1)+x_1$	$x_1\in[-5;5]$