AUTOMATED SYNTHESIS OF SELECTION OPERATORS IN GENETIC ALGORITHMS USING GENETIC PROGRAMMING[1]

Evgenii Sopov, Eugene Semenkin

Department of Systems Analysis and Operations Research
Siberian State University of Science and Technology
evgenysopov@gmail.com, eugenesemenkin@yandex.ru
Russian Federation

Abstract: Genetic algorithms must be fine-tuned in order to achieve the best results. In this study, we have proposed a new hyper-heuristic based on genetic programming for the automated synthesis of a selection operator in genetic algorithms. Black-Box Optimization Benchmarking is used as a training set and as a test set for estimating the generalization ability of a synthesized selection operator. The results of numerical experiments for benchmark problems and a real-world problem are presented and discussed. The experiments have shown that the proposed approach can be used for designing new selection operators that outperform standard selection operators on average.

Key words: genetic algorithms, genetic programming, constructive hyper-heuristic, selection operator.

1. INTRODUCTION

Metaheuristics, such as genetic algorithms (GAs), have proved their efficiency over a wide range of real-world search optimization problems. GAs realize the "blind" search strategy and do not require any specific information about the features of the search space and objectives. Nevertheless, the performance of a GA strongly depends on the algorithm's structure, the chosen types of genetic operators and their parameters. Such fine-tuning of the GA can be performed using some a priori information on a given problem, through a series of numerical experiments with different algorithm settings or in an automated way during the algorithm' run. The last option has become more popular in recent times and often outperforms traditional schemes of applying GAs. There exist many self-adaptive and self-configuring approaches in GAs [1].

The most recent studies in the field propose more complex approaches for the automated design of search metaheuristics, which are called hyper-heuristics (HHs). Genetic

[1] This research is supported by the Ministry of Education and Science of Russian Federation within State Assignment № 2.1676.2017/ПЧ and by Russian Foundation for Basic Research, Government of Krasnoyarsk Territory, Krasnoyarsk Region Science and Technology Support Fund to the research project №6-41-240822.

programming (GP) has been proposed as a method for automatically generating computer programs. Today GP is used in the field of machine-learning for a wide range of applications. GP can be also applied as a hyper-heuristic for generating search heuristics and metaheuristics (so-called GPHH) [2].

In the field of GAs, the problem of configuring and fine-tuning an algorithm is usually solved only once for a given instance of an optimization problem. At the same time, machine-learning approaches are aimed to find or to build an algorithm (a classifier, a regression model, etc.) based on a set of training instances, which can deal efficiently with new instances of a problem. This concept can be transferred to the GPHH approach. An objective of the GPHH is to design a metaheuristic that demonstrates high performance on average over a set of complex optimization problems. Such feature is necessary for "black-box" optimization, because the synthesized metaheuristic should not adapt to a specific problem, but cover a range of problems.

This study is devoted to improving the GA performance by generating and tuning new selection operators using the GPHH. We will focus on the standard binary GA for solving single-objective unconstrained "black-box" optimization problems. The well-known benchmark problems from the GECCO and IEEE CEC conferences (Black-Box Optimization Benchmarking (BBOB)) are used as a training set. The benchmark contains GA-hard problems and combines many features of optimization problems.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes the proposed approach. In Section 4, the results of numerical experiments are discussed. In the Conclusion the results and further research are discussed.

2. RELATED WORK

2.1. Hyper-heuristics

Hyper-heuristic approaches perform a search over the space of heuristics or metaheuristics when solving optimization problems. In a HH approach, different heuristics or heuristic components can be selected, generated or combined to solve a given problem in an efficient way. In [3], a general classification of HHs is given according to two dimensions: the nature of the heuristic search space and the source of feedback during learning. The first characteristic defines the following two classes:

- Heuristic selection: Methodologies for choosing or selecting existing heuristics.
- Heuristic generation: Methodologies for generating new heuristics from components of existing heuristics.

The first class is similar to the idea of the self-configuration in evolutionary algorithms. The selection can be performed with a set of predefined evolutionary and genetic operators (for example, Population-level Dynamic Probabilities in GP [4], Linear Genetic Programming for the Synthesis of Evolutionary Algorithms [5]) or from a set of predefined algorithms (for example, the Population-based Algorithm Portfolio [6], the Multiple Evolutionary Algorithm (MultiEA) [7] or the Self-configuring Multi-strategy Genetic Algorithm [8]).

In this study, we will focus on the second class of HHs. As we can see from many papers (a good survey of HHs is proposed in [2]), the majority of HHs have been proposed for solving combinatorial optimization problems. Nevertheless, there exist some examples of

HHs for general optimization problems, and the best results are achieved with HHs based on GP [9].

With respect to the source of feedback during learning there exist:

- Online learning HHs: Learn while solving a given instance of a problem.
- Offline learning HHs: Learn from a set of training instances a method that would generalize to unseen instances.
- No-learning HHs: Do not use feedback from the search process.

In many real-world applications, a chosen heuristic optimization method is applied to find the best solution with respect to a single instance of a certain optimization problem. In such applications, the generalization ability of the method applied does not matter, as the problem should be solved only once. In this case, we can use a no-learning HH if there is presented a priori information on the problem, or we can use an online learning HH to solve the problem in an automated, adaptive way.

In this study, we will use an offline learning approach to provide the generalization feature and to synthesize a GA with a new selection operator, which outperforms the standard GA on average.

The application of GP as a HH is a rather new direction in the field of automated algorithm design. GP builds candidate solutions to the problem from a set of primitives, which are represented by single operators, functions or whole heuristics and metaheuristics. One of the main advantages of GP is that it simultaneously provides the structural synthesis of a solution and the tuning of its parameters. The solution can be a human-readable symbolic expression (a formula) or a computational algorithm (an executable computer program).

2.2. The design of selection operations in EAs

Hyper-heuristic approaches perform In the majority of studies, GPHH is applied for the automated design of data mining tools and combinatorial optimization algorithms. There are only a small number of works on applying GPHH for the design of new operations in evolutionary and genetic algorithms. In [10], GPHH is used for the automated design of mutation operators for evolutionary programming and in [11] the same problem is solved for GAs. In [12], selection operators are synthesized using a register machine as a tool for automated programming.

A selection operator is an important component of any evolutionary or genetic algorithm. The selection operator is intended to improve the average fitness of a population by giving individuals with better fitness a higher probability to be copied into the next generation. From the point of view of search optimization, selection focuses the search process on promising regions in the search space, while recombination performs a random search within previously explored regions, and mutation discovers new points in the search space.

Any selection operator can be viewed as a probability distribution that assigns the chance of being chosen for further operations to every individual in a population. Thus, selection can be defined as a mapping (of a function) *s* to the [0, 1] interval. The domain of the *s* function comprises ranks for the ranking, tournament and truncation selection schemes, and comprises fitness values for the proportional selection (1). The *s* function should satisfy the normalization requirement (2).

$$s(i): rank_i \to [0,1], rank_i \in Z \quad \left(s(i): fitness_i \to [0,1],\ fitness_i \in R\right) \qquad (1)$$

$$\sum_{i \in Pop} s(i) = 1 \tag{2}$$

where *Pop* is a population and *i* is an individual in the population.

A comprehensive analysis of selection schemes in GAs is presented in [13], where selection operators are described and compared using estimations of average fitness, fitness variance, reproduction rate, loss of diversity, selection intensity and selection variance. Unfortunately, the proposed models are too complicated and can be applied only for simple optimization problems such as the ONEMAX function. Nevertheless, it can be seen that the characteristics of probability distributions presented by selection schemes are different and there is no guarantee that the best-chosen scheme from a set of traditional selection schemes will be optimal or even efficient enough for solving an arbitrary optimization problem.

We have visualized the probability distributions of commonly used selection operators (Fig. 1). We have ranked solutions for all types of selection in such a way that the first rank is assigned to the worst solution. The tournament selection is not presented, because it is asymptotically equal to the linear ranking. As we can see, the probability distributions are always monotonically increasing functions. The functions for the ranking and tournament are smooth. The function graph of the proportional selection strongly depends on a distribution of the fitness values. If the fitness distribution is uniform, the graph is close to the graph of the linear ranking, otherwise it is closer to the exponential ranking.

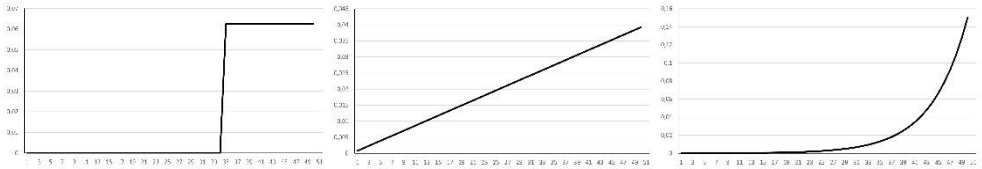

Fig. 1. The probability distributions of commonly used selection operators: a) truncation, b) linear ranking, c) exponential ranking (base of the exponent is c=0.8)

The traditional selection operators are inspired by nature and use straightforward and simple ways for calculating the probability of being selected. In this study will use GPHH to synthesize new selection operations, which maximize the average performance of a GA.

3. GPHH FOR AUTOMATED SYNTHESIS OF SELECTION OPERATORS IN GAs

In this study, we propose the following conception of applying GP as a HH for the automated design of selection operators. We will use a GP algorithm as a meta-procedure for solving a symbolic regression problem, in which tree solutions represent probability distributions. A raw solution is normalized, and after that is executed as a selection algorithm in a certain GA. For evaluating fitness of the GP solution, we estimate the average performance of the GA with the designed selection operator over a series of runs. The BBOB functions are used and they are divided into training and test sets to estimate the generalization ability of the solution. The proposed scheme of GHPP is presented in Fig. 2. We will discuss the components and stages of the approach in detail.

It is obvious that the proposed GPHH is a computationally costly procedure. Thus, we will introduce some modifications to the standard GP algorithm. As was shown in the previous section, the probability distributions of selection are rather simple, smooth and increasing functions. We will strongly control the bloat of the tree solutions by limiting their maximum depth [14]. In standard GP, a grow method on the initialization step uses equal probabilities for all elements of the functional set. We will use a higher probability for the addition to get more simple and smooth expressions. The same probabilities for elements of the functional set will be used for a point-mutation operation.

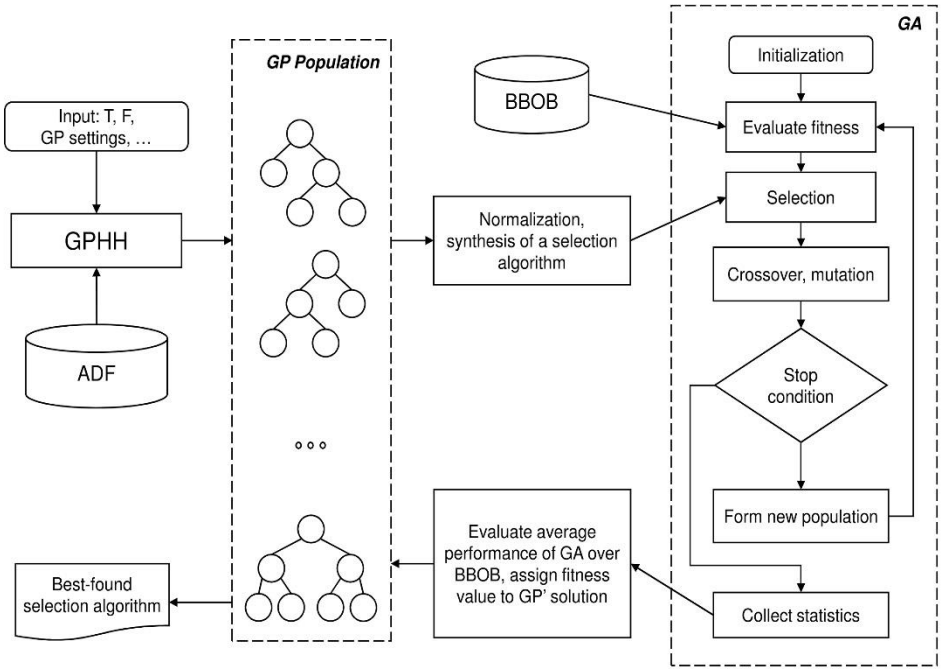

Fig. 2. The general scheme of the proposed GPHH for automated synthesis of selection operators in GAs

A terminal set in GP includes variables and constants for a given problem. The selection probability of a solution in the GA may depend on many factors. Usually all requirements of an optimization problem are presented in the fitness function. Thus, we can include in the terminal set only one input variable based on the fitness value. We do not use the values of the fitness function as values varies from population to population, and thus there will be the need for adjustment of terminal parameters in the GP solutions or we will have to scale the fitness on each iteration.. To avoid fitness scaling, we will rank individuals in the GA, and will use the ranks as the variable in the terminal set (in this work, the first rank corresponds to the worst individual). We will also include two automatically defined functions (ADFs) in the terminal set, which are the linear ranking and the exponential ranking with c=0.8.

Each tree solution in the GP is a function with arbitrary codomain (denoted as $f(i)$, where i is the rank of an individual after the ranking). We need to provide some transformation of

the function for applying it as a selection operator *s(i)*. We will bound the domain with rank values and will apply normalization (3).

$$s(i) = \frac{f(i) - f_{\min} + \Delta}{\sum_{i=1}^{PopSize} \left(f(i) - f_{\min} + \Delta \right)}$$
$$f_{\min} = \min_{i \in [1, PopSize]} f(i) \quad (3)$$

where *PopSize* – is the size of a population, *i* is an individual in the population, f_{min} shifts the function into the positive range, Δ is a parameter that guarantees a non-zero probability for the worst individual.

The GA used in this study is a standard GA with standard binary representation. The crossover operator is two-parent uniform recombination. The mutation operator is a bit inversion and has average intensity, which is calculated as (1/*chromosome_length*) (one mutation per string on average). The stop criterion is a limitation of fitness evaluations in terms of the number of generations with a fixed-sized population. Each new population is formed by copying offspring into it. The worst individual is replaced by the best-found solution (elitism). The fitness function for a single run of the GA is the same as the chosen objective from the BBOB set, and it is to be maximized.

We have chosen the BBOB, because this benchmark is widely used by other researchers. It is well-developed and contains a wide range of complex optimization problems with many different features. This benchmark has also been used at the Black-box Optimization Competition within the IEEE CEC and GECCO conferences since 2009. More detailed information and source codes can be found in [15].

In this study, only noiseless functions have been used. The set contains 24 benchmark problems, which are grouped in 5 subsets: separable functions (the subset contains 5 problems), functions with low or moderate conditioning (4 items), functions with high conditioning and unimodal (5 items), multi-modal functions with adequate global structure (5 items) and multi-modal functions with weak global structure (5 items).

To balance benchmark problems in the training and test sets, we have divided the problems in the following way: one random problem from each subset has been chosen into the test set, and all the other problems from the subsets have been added to the training set. As result, the training set contains 19 problems, and the test set contains 5 problems. The training set is used for evaluating the fitness function in the GP algorithm. Finally, we will examine the generalization ability of the best-found selection operator with the test set.

All benchmark functions are scalable with the dimension. In this work, we have set the dimension for all problems equal to *D*=2. The accuracy for the binary coding is ε=10E-3. The chromosome length is equal to 28 (14 bits per one real-valued variable).

In order to estimate the average performance of the GA over the training set, we have to evaluate a quality metric for every problem in equal scale. In the BBOB, the position of the global optimum for a problem is initialized at random in $[-5, 5]^D$. We can estimate the GA performance by calculating the distance between the best-found solution and the global optimum position. The Euclidian metric is used (4). As the search domain is given and we can calculate the maximum and minimum values for the metric, we will transform the metric in the following way: it is equal to 1 if the global optimum has been found and is equal to 0 if the distance is maximum. Thus, we can represent the performance measure for the GA as a percentage (5). Finally, we can assign the fitness of a solution in the GP as in (6).

$$Dist = \left\| x_{GA} - x_{opt} \right\| \tag{4}$$

$$PerformanceGA_{ij} = \frac{\max Dist - Dist_{ij}}{\max Dist}, \quad i = \overline{1,F},\ j = \overline{1,R} \tag{5}$$

$$fitnessGP(a_k) = \frac{1}{F}\sum_{i=1}^{F}\left(\frac{1}{R}\sum_{j=1}^{R} PerformanceGA_{ij}(a_k)\right) \tag{6}$$

where *Dist* is the distance between the best-found solution x_{GA} and the global optimum position x_{opt}, $PerformanceGA_{ij}$ is the GA performance estimated with the *i*-th training problem on the *j*-th algorithm run, *F* is the number of the training functions, *R* is the number of runs, $fitnessGP(a_k)$ is the fitness assigned in the GP to a solution a_k.

We will also estimate the performance of the standard GA with proportional selection, linear ranking and exponential ranking with c=0.8 and their average. The average value can be viewed as the average performance of the GA with a randomly chosen selection operator. Such an estimate is very useful for black-box optimization problems, because we have no information about problem features and, consequently, about what type of selection to use. The Mann-Whitney-Wilcoxon Test at the significance level α=0.05 is used for defining the statistical significance of difference in the results.

4. EXPERIMENTAL SETUP AND RESULTS

4.1. The BBOB problems

All algorithms and benchmark problems have been implemented in the R language. The BBOB source codes in R are available in [15]. The GP algorithm is based on the R genetic programming framework (RGP) from the CRAN repository [16]. The GA was realized in R by authors.

The experimental settings are: the BBOB problems in the training set: *{f1-2, f4-8, f10-11, f13-14, f16-19, f21-24}*; the BBOB problems in the test set: *{f3, f9, f12, f15, f20}*; the search domain for all problems: [-5, 5]x[-5, 5], encoding accuracy: ε=10E-3.

Settings for the GP algorithm are: population size: N_{GP}=50; the grow method: full with maximum depth of trees equal to 6; the functional set (probability for initialization): {+ (0.5), - (0.2) ,* (0.1), % (0.1), *sin* (0.05), *exp* (0.05)}; the terminal set (probability for initialization): {i (0.5), constants (0.3), ADF1 (0.1), ADF2 (0.1)}; constants: random uniform in [0,1]; ADF1 (the linear ranking); ADF2 (the exponential ranking with c=0.8); crossover type (probability): one-point (0.95); mutation type (probability): one-point (0.01); max number of generations: 1000; fitness function: see (6).

Settings for the GA algorithm are: population size: N_{GA}=50; chromosome length: n=28; initialization: random uniform in the binary search space; selection type: based on GP solutions; crossover type (probability): two-parent random uniform (1.0); mutation type (probability): bits inversion (1/n); fitness function: objectives from the training set; max number of generations: 50; number of independent runs: 15.

The expression of the best-found solution (*fitnessGP*=0.83) is presented in (7). The expression after normalization is presented in (8) and its graph is shown in Figure 3a.

$$f(i) = 10 \cdot \sin(0.1 \cdot i - 5) + i \quad (7)$$

$$s(i) = \frac{10 \cdot \sin(0.1 \cdot i - 5) + i + 9.82453}{707.405} \quad (8)$$

The performance of the GA with the standard selection operators and the GPHH best-found selection operator (7) on the test set is shown in Table 1. The table contains results over 15 independent runs of the GA in the form of "mean/STD".

We have also estimated the statistical significance of differences in the results (Table 2). We use the following notations: the sign "=" is used when the difference is not significant, the sign ">" is used when the GPHH best-found selection provides significantly better results, and the sign "<" when the GPHH best-found selection provides significantly worse results.

As we can see, the synthesized selection operator performs better with 3 of the 5 problems. The linear ranking performs better on f15 and f20, but the difference is significant only with the f15 problem. The proposed selection operator always outperforms the proportional selection and it is better or equal to the exponential ranking.

We will try to analyze why the proposed selection operator performs well. The form of the curve proposed in the Figure 3a is close to the exponential ranking distribution of the selection probabilities. The exponential ranking provides stronger selective pressure than the linear ranking, but does not lead to the "super-individual" problem as the proportional selection does. The proposed selection operator has also a subinterval of equal probabilities (approximately for ranks from 8 to 30). This may cause to the diversity maintaining in the population. However, as the probabilities raise for higher (better) ranks, the GA still able to converge.

Table 1. The experimental results for the test problems.

The test set	***Proportional selection***	***Linear ranking***	***Exponential ranking (c=0.8)***	***Average***	***GPHH best-found selection***
f3	*50.14/4.56*	*66.97/8.86*	*73.47/6.02*	*63.53/6.48*	***78.86/6.56***
f9	*76.71/7.64*	*81.64/6.84*	*82.93/6.24*	*80.24/6.91*	***90.55/5.11***
f12	*78.48/11.43*	*88.13/6.53*	*89.02/7.2*	*85.21/8.39*	***91.43/3.64***
f15	*64.43/11.10*	***85.17/5.37***	*79.48/8.20*	*76.36/8.22*	*82.26/7.86*
f20	*61.86/13.74*	***83.01/8.26***	*76.51/14.50*	*73.79/12.17*	*81.44/11.81*

Table 2. The results of the Mann-Whitney-Wilcoxon tests.

The test set	***Selection***	***Proportional selection***	***Linear ranking***	***Exponential ranking (c=0.8)***
f3	*GPHH best-found selection*	>	>	>
f9		>	>	>
f12		>	=	>
f15		>	<	=
f20		>	=	=

The synthesized selection operator outperforms the average of three standard selection operators on the whole range of test problems. Thus, we can recommend it for black-box optimization problems.

4.2. Real-world problems

We have estimated the performance of the proposed approach with a real-world problem of training neural network (NN) classifiers. The problem of efficient applying NNs is still a challenge in the field of machine learning [17]. The combination of EAs and NNs is known as one of perspective approaches [18]. The automated synthesis of an EA used for design or/and training NN can improve the total efficiency of the evolutionary neural network.

In this study, GAs are used to find a starting point for the back-propagation algorithm. We have chosen 11 datasets from the UCI repository [19], which contain two- and multiclass problems.

First 8 problems are used as a training set for the GPHH, the remaining 3 are used as a test set. For multiclass problems, a series of binary classification problems is solved using the "one against all" scheme. All missing values in data are interpolated using average values. The implemented neural network is based on the "neuralnet" package from the CRAN repository [20]. The standard accuracy is used as the training and test error metric. Each classification problem is solved 20 times.

The general settings for the GPHH and GAs are as in the previous section except of chromosome lengths and population sized in GAs. These two parameters are manually tuned and they depend on the chosen neural network structure for each dataset.

The expression of the best-found solution is presented in (9). The expression after normalization is presented in (10) and its graph is shown in Figure 3b.

$$f(i) = 0.001 \cdot i^3 - 20 \cdot \sin(6.8 - 0.1 \cdot i) \tag{9}$$

$$s(i) = \frac{0.001 \cdot i^3 - 20 \cdot \sin(6.8 - 0.1 \cdot i) + 8.095998}{2294.765} \tag{9}$$

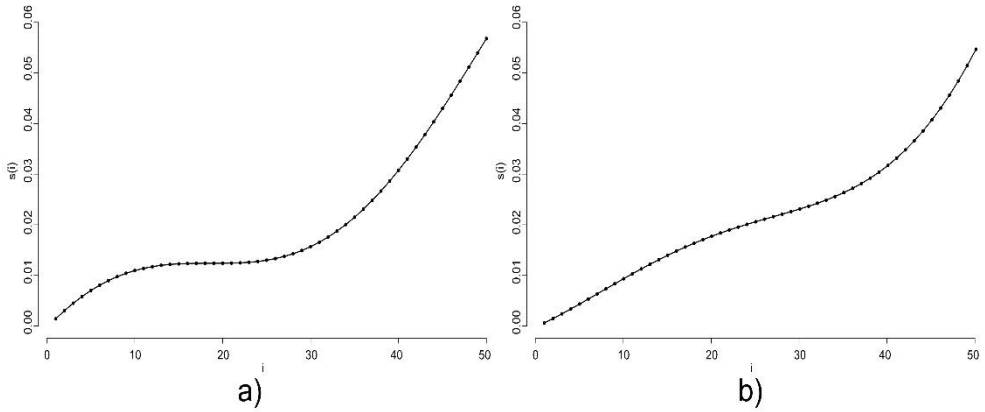

Fig. 3. The graph of the best-found selection operator for a) BBOB problems and b) classification using NN

We have compared the results obtained by the GPHH best-found solution with the standard GA with tournament selection. The results are presented in Table 3.

As we can see, the proposed GPHH approach has synthesized a new selection operator that can outperform the standard GA. It provides better results with 6 of 8 training problems and 2 of 3 test problems.

Table 3. The results of classification using NN.

Dataset	**NN + tournament selection GA**	**NN + GPHH best-found selection**
Australian Credit Approval	***83.9/5.11***	*82.1/9.64*
Breast Cancer Wisconsin	*96.13/1.7*	***98.3/0.5***
Pima Indians Diabetes	*72.56/8.59*	***78.0/5.07***
Hepatitis	*89.09/6.91*	***89.99/4.28***
Iris	*99.9/0.05*	***100/0.0***
Wine Quality	*61.7/21.4*	***64.03/16.74***
Glass Identification	*65.88/22.05*	***68.2/16.8***
Seeds	***87.4/3.22***	*85.1/7.35*
Ionosphere	*74.7/18.8*	***78.9/13.3***
German Credit Data	*78.8/9.5*	***81.07/8.6***
Heart Disease	***85.57/11.7***	*83.09/18.3*

5. CONCLUSIONS

In this study, we have proposed a hyper-heuristic approach based on genetic programming which is used for the automated synthesis of selection operators in genetic algorithms. The approach implements the generalization conception taken from the machine-learning field. A selection operator is designed in order to maximize the average performance of a GA with a given set of training instances of black-box optimization problems.

As numerical experiments have shown, the approach can deal with the problem of automated synthesis. Moreover, the synthesized selection operator provides statistically significant better performance or at least performance not worse than standard operators do. It also performs well on a test set with new, previously unseen instances of optimization problems, and thus, the generalization feature of the proposed GPHH is proved. Our experiments with real-world problems have also proved that a synthesized selection operator can improve a GA performance.

In our further works, we will extend training and test sets and will perform more numerical experiments. We will also try to apply the approach to the problem of the automated synthesis of other genetic operators such as crossover and mutation. The standard GP algorithm can be substituted with Grammatical Evolution in order to better control the syntax of synthesized operators.

REFERENCES

[1] Eiben, A.E. et al. Parameter Control in Evolutionary Algorithms. *Parameter Setting in Evolutionary Algorithms*, Volume 54 of the series Studies in Computational Intelligence, 2007, pp. 19-46.

[2] Burke, E. et al. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64 (12), 2013, pp. 1695-1724.

[3] Burke, E. et al. A classification of hyper-heuristic approaches. *Handbook of metaheuristics. International Series in Operations Research & Management Science,* Vol. 146, 2010, pp. 449-468.

[4] Niehaus, J. Banzhaf, W.Adaption of Operator Probabilities in Genetic Programming. *EuroGP 2001, LNCS 2038*, 2001, pp. 329.

[5] Oltean, M. Evolving evolutionary algorithms using linear genetic programming. *Evol. Comput.*, 13, 2005, pp. 387–410.

[6] Tang, K., et al. Population-Based Algorithm Portfolios for Numerical Optimization. *IEEE Transactions on Evolutionary Computation*, 14(5), 2010, pp. 782-800.

[7] Yuen, S.Y. et al. Which Algorithm Should I Choose at any Point of the Search: An Evolutionary Portfolio Approach. *Proceedings of the 15th annual conference on Genetic and evolutionary computation, GECCO'13*, 2013, pp. 567-574.

[8] Sopov, E. A Self-configuring Multi-strategy Multimodal Genetic Algorithm. *Advances in Nature and Biologically Inspired Computing,* Volume 419 of the series Advances in Intelligent Systems and Computing, 2016, pp. 15-26.

[9] Burke, E. Exploring hyper-heuristic methodologies with genetic programming. *Computational Intelligence: Collaboration Fusion and Emergence,* New York, Springer, 2009, pp. 177-201.

[10] Hong, L. et al. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. *Genetic Programming: 16th European Conference, EuroGP'13,* 2013, pp. 85-96.

[11] Woodward, J.R., Swan, J. The automatic generation of mutation operators for genetic algorithms. *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation GECCO'12*, 2012, pp. 67-74.

[12] Woodward, J.R., Swan, J. Automatically designing selection heuristics. *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation GECCO'11,* 2011, pp. 583-590.

[13] Blickle, T., Thiele, L. A comparison of selection schemes used in evolutionary algorithms. *Evol. Comput.*, 4(4), 1996, pp. 361-394.

[14] Luke, S., Panait, L. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3), 2006, pp. 309-344.

[15] The Black-Box-Optimization-Benchmarking (BBOB) documentation and source codes [online] Available at: http://coco.gforge.inria.fr/

[16] RGP: R genetic programming framework [online] Available at: https://cran.r-project.org/web/packages/rgp/index.html

[17] Kasabov, N. From Multilayer Perceptrons and Neuro-Fuzzy Systems to Deep Learning Machines: Which Method to Use? – A Survey. *International Journal on Information Technologies and Security*, No. 2 (vol. 9), 2017, pp. 3-24.

[18] Ding, Sh. Et al. Evolutionary artificial neural networks: a review. *Artif. Intell. Rev.*, 39 (3), 2013, pp. 251-260.

[19] UC Irvine Machine Learning Repository [online] Available at: http://archive.ics.uci.edu/ml/index.php

[20] Package 'neuralnet' [online] Available at: https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

Information about the authors:

Evgenii Sopov - PhD, associate professor in Department of Systems Analysis and Operations Research, Siberian State University of Science and Technology, Russia. Areas of scientific research are evolutionary computation, intelligent information techniques, modelling and optimization in complex systems. Phone: +73912919141. evgenysopov@gmail.com

Eugene Semenkin – Dr. Sc., professor at the Department of Systems Analysis and Operations Research, Siberian State University of Science and Technology, Russia. Areas of scientific research are complex systems modelling and optimization, computational intelligence, data mining. Phone: +7-391-2919141. eugenesemenkin@yandex.ru

Manuscript received on 05 October 2017