Adaptive component crossover for differential evolution in solving single-objective optimization problems

Anton Sopov[1,*], and *Evgenii* Sopov[1,2]

[1]Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarsk, Russia
[2]Siberian Federal University, Institute of Space and Information Technology, Krasnoyarsk, Russia

Abstract. With the increase of the complexity of engineering problems, evolutionary algorithms became an effective approach to black-box optimization problems. One of the most popular and promising evolutionary methods is the Differential Evolution algorithms. This method involves several evolutionary operators, including crossover, which is used to form offspring based on mutant and parent vectors, and is important in forming new generations of solutions. However, the classic differential evolution and its numerous modifications usually tends to use the single crossover mechanism to each of the variables of the system, therefore the properties and role of the subcomponents are not considered. That may lead to a slower convergence and increasing demands on computing resources. In this study we have proposed a novel Adaptive Component Crossover strategy for differential evolution, in which the crossover rate parameter is represented by a vector and its values are based on the behavior of the objective function on separate components. The experimental results on a set of benchmark problems have shown that the proposed scheme can improve the performance of the algorithm and, in particular, increase the convergence speed and crossover success rate.

1 INTRODUCTION

In the real world, there are many applied optimization problems that are presented as a black-box model. Such problems are either do not contain any information on properties of the objective function and its derivatives or are too complex to solve with traditional optimization methods. With the increase of the complexity of engineering problems, the development of new optimization techniques is one of the important tasks of computational intelligence. Different blind search methods, heuristics, and metaheuristics became a popular way to deal with black-box optimization problems in recent decades, including evolutionary algorithms (EAs). This class of stochastic methods is based on the use of populations and modeling evolutionary processes from biology [1]. EAs involve populations of candidate solutions, each new generation is created by applying evolutionary operators to existing individuals and

* Corresponding author: antonsopov2004@gmail.com

updating the population with the more suitable solutions. As a result, candidate solutions gradually increase their fitness [2].

Differential evolution (DE) is one of evolutionary algorithms, which demonstrates the high performance in solving continuous global black-box optimization problems [3]. DE is a stochastic population-based method, which uses the difference-based mutation of random solution vectors made of several random agents. Differential evolution led to the development of a series of successful algorithms, which implement various mechanisms, including adaption strategies, in order to improve the performance of solving hard optimization problems [4]. The performance of differential evolution is affected by both evolutionary operators and parameter values control. While the mutation, as the key DE operator, was a part of many studies, the crossover operation and its role in evolutionary process is less discussed [5]. The classic crossover variants use the same parameter value for each component, which may lead to the fewer successful trial individuals and slower convergence.

This paper is devoted to a novel parameter adaption strategy, which is based on the usage of the information about the behavior of the objective function on separate components. The proposed scheme is used to improve the crossover operator. The numerical experiments are performed on the CEC 2022 benchmark problems [6].

2 RELATED WORK

The differential evolution algorithm and its main steps are described, including the binomial crossover, which will be expanded and modified.

2.1 Problem statement

A global minimization problem can be stated as follows:

$$f(\boldsymbol{x}) \to \min_{x \in R^n}, f: R^n \to R, \tag{1}$$

where $f(x)$ is the objective function, $f: R^n \to R, x \in R^n, X = (x_1, \dots, x_n), x_i \in S$ is an arbitrary n-dimensional solution. S is the box constrained search space, defined by the lower value l_i and upper value u_i, which are determined for each dimension: $l_i \le x_i \le r_i, i = \overline{1,n}$. f is assumed to be continuous.

The solution to the problem can be stated the following way:

$$x^*: \forall x \in S \subset R^N, f(x^*) \le f(x), \tag{2}$$

where x^* is called a global optimum. As there is no additional information about the objective function properties and its derivatives, the problem can be described as a black-box model. For that case stochastic optimization algorithms are used, as they only use the information about the inputs and outputs of the considered system. One of the most common and efficient heuristics for solving global optimization problems are evolutionary algorithms, which are population-based approaches that are related to biological evolution processes.

2.2 Differential evolution

The different evolution (DE) algorithm was proposed by Storn and Price in 1995 and became one of the most popular continuous optimization algorithms, leading to the foundation of many successful methods. DE is the stochastic population-based numerical optimization algorithm, which uses the difference-based mutation of random solution in order to explore the search space.

In the initialization step, a population of size N is generated randomly within the search space. For each of the solutions, a fitness function is computed that demonstrates the quality

of the individual relative to others in the population. In case of bound-constrained optimization, the fitness function is equal to the objective function.

Next, the main loop of the algorithm begins. The main DE operator is mutation, a number of individuals are randomly selected from the population, then the mutant vector is constructed based on difference of the existing solutions. An equation, which forms the new solution vector, is called mutation strategy. The most common strategy is called *rand/1*:

$$V_{ij} = x_{r1j} + F(x_{r2j} - x_{r3j}), \tag{3}$$

where V_i^t is a mutant vector, indexes $r1$, $r2$, and $r3$ are randomly selected, i is the current individual, j is the variable, F is a scaling factor parameter, $F \in [0; 2]$.

If the j-th coordinate of the mutant vector goes beyond the search space (i.e., $u_{ij} \notin [lb_j, ub_j]$), the following handling method is applied:

$$\begin{cases} V_{ij} = \frac{lb_j + x_{ij}}{2}, if\ V_{ij} < lb_j \\ V_{ij} = \frac{ub_j + x_{ij}}{2}, if\ V_{ij} > ub_j \end{cases}, \tag{4}$$

After the mutant vector is formed, the crossover operation is applied. The most widely used crossover scheme is the binomial crossover (*bin*), which uses the following rule:

$$u_{ij} = \begin{cases} V_{ij},\ if\ rand < CR\ or\ j = jrand \\ x_{ij}, otherwise \end{cases}, \tag{5}$$

where u_{ij} is a trial vector, CR is the crossover rate parameter, $CR \in [0; 1]$, $jrand$ is a random index from $[1; D]$. The additional condition is applied to make sure that at least one index from the mutant vector is taken.

After that, the fitness value of the trial vector is calculated. Then selection operation compares two vectors, according to their objective function values. The better vector becomes a member of the next generation. Selection operator is defined as follow:

$$x_{ij} = \begin{cases} V_{ij},\ if\ rand < CR\ or\ j = jrand \\ x_{ij}, otherwise \end{cases}. \tag{6}$$

These three operators run for each individual in population until a new population is formed. Then the next cycle starts with new individuals. The algorithm repeats this sequence until the stopping criterion is reached. Usually, algorithm stops as soon as the user-defined limit of objective function evaluations or working time is reached.

3 PROPOSED APPROACH

In the most common DE implementations parameters are usually represented as real numbers. Although it makes the control process easier to handle, it doesn't consider any information about the behavior of objective function with the change of a single variable or subcomponent. While varying some components can lead to a better result, changing a value of a variable that is already established or is too sensitive can make the descendant less fitted, despite of positive changes of another components.

In case of crossover, the probability parameter CR is a real number which is applied to every variable in the solution vector. The popular adaptation strategies include historical memory [7], weighted means [8], etc. Usually, these schemes generate a probability value for each individual with using Gaussian distribution.

In this study we have proposed a novel binomial crossover scheme, denoted as Adaptive Component Crossover (*aco*), in which CR is represented as a vector with an individual value for each variable. A general description of the crossover strategy and the versions that are used in this paper are presented in this section.

The proposed strategy can be conditionally divided into three steps:

1. Calculation of the weighted change matrix θ.

2. The update of component weights ∇_j, which are used to control crossover rate.
3. The update of parameter CR_j.

The algorithm doesn't have any stored information at the first generation, so Cr_j are initialized to a fixed value, in our case, $Cr_j^1 = 0.8$.

During the crossover, the information about its results is saved in the memory. More precisely, the absolute value of changes of the variable Δx_{ij} and function value changes Δf_i are considered. Using them, a matrix of weighted changes θ is constructed, the elements are calculated using the following rule:

$$\theta_{ij} = -\frac{\Delta f_i}{\max|\Delta f_i|} \cdot W(\Delta x_{ij}, \max \Delta x_{ij}), \tag{7}$$

where W is weight function, $\Delta x_{ij} \epsilon [0, \max \Delta x_{ij}]$, $W \in [0; 1]$, W is monotonically non-decreasing.

The matrix θ is used to estimate the impact on the component j in the function value change for individual i. If an individual had the better change compared to the previous generation, then the variables that have changed more should be given more weight. W defines the dependence of the significance of a variable in an individual on the change in its value. The closer W value is to 0, the less weight is given, and vice versa.

Two weight distribution strategies are used in this study, denoted as W_{Sin} and W_{Step}:

$$W_{Sin}(\Delta x, \max \Delta x) = \frac{1}{2}\left(\sin\left(\frac{\Delta x \pi}{\max \Delta x} - \frac{\pi}{2}\right) + 1\right), \tag{8}$$

$$W_{Step}(\Delta x, \max \Delta x) = \begin{cases} 0, \Delta x < 0.01 \max \Delta x \\ 1, otherwise \end{cases}. \tag{9}$$

The distribution (8) is a section of a sinusoid, which is nonlinearly increasing and sets an individual weight for every possible value of Δx, while the rule (9) considers only the elements that had enough change.

After the weighted change matrix is formed, the changes per each variable are summarized. The component weights ∇_j are computed using the following equation:

$$\nabla_j^{t+1} = \frac{t}{t+1}\left(\alpha * \nabla_j^t + (1 - \alpha) * \underset{j}{\text{mean}}\, \theta_{ij}\right), \tag{10}$$

where α is a coefficient that is used to control the impact of the current generation, $\alpha \in [0; 1]$. In case of $\alpha = 1$, only the previous generation weights are considered. If $\alpha = 0$, ∇_j is equivalent to the mean of changes of the current generation. Also, in case of $\alpha = 1/t$, ∇_j can be interpreted as a mean value of previous and current generations. The following scheme is used to control the parameter:

$$a = \begin{cases} \frac{1}{2}, \ NFES \leq \frac{1}{8}\max FES \\ \frac{1}{4}, \ NFES \leq \frac{1}{4}\max FES \\ \frac{1}{t}, otherwise \end{cases}, \tag{11}$$

During the first generation, α is initialized to 1. Next, during the first stage, θ_{ij} is more sensitive due to the wide spread of solutions in the search space, so the means of current generation are given the same weight as ∇_j^t. In the later stages of evolutionary process, a is lowered.

Next, the weights are normalized to the interval depending on the number of function evaluations:

$$\begin{cases} (CR_{lmax}; 0.9), \ NFES \leq \frac{1}{4}\max FES \\ \left(\frac{Cr_{lmax} + Cr_{lmax}}{2}; 0.9\right), NFES \leq \frac{1}{2}\max FES, \\ (CR_{lmin}; 0.9), \ otherwise \end{cases} \tag{12}$$

where CR_{lmin} and CR_{lmax} are the minimum and maximum lower threshold of CR respectively. In case of our study, $CR_{lmin} = 0.3$, $Cr_{lmax} = 0.6$. Normalized weight values are labelled as δ_j.

Finally, the crossover rate vector is generated by the following formula:

$$CR_j^{t+1} = randn(\delta_j, 0.1), \tag{13}$$

where $randn$ is the Gaussian distribution. If the generated CR_j is greater than 1, then let $CR_j = 1$, and if CR_j is less than 0, then let $CR_j = 0$.

4 NUMERICAL EXPERIMENTS

The performance of the algorithm proposed in this paper is evaluated using the CEC 2022 benchmark for single objective bound-constrained numerical optimization [6].

4.1 The experimental setup

The CEC 2022 benchmark suite contains 12 test optimization problems with various characteristics, including multimodal, shifted, rotated, and ravine-type functions. The number of variables for selected problems is equal to 20 for each problem. The search borders for each problem are $[-100; 100]$. The termination criteria for algorithms are when it reaches the maximum number of objective function (1000000) evaluations and when the difference between the known optimal value and the best solution value is less that 10^{-8}. The performance is evaluated on 30 independent runs. All algorithms were implemented using the Python programming language and multi-core parallelization. As the algorithm was introduced for the standard DE, we have compared it to the *rand/1/bin* scheme. DE with the proposed crossover scheme (denoted as *rand/1/aco*) was tested with 2 different weight distribution strategies: W_{Sin} and W_{Step}. The algorithm parameters are as follows: $F = 0.9$, $PS = 200$. The performance of all comparison algorithms was estimated using the minimum, mean, median, and standard deviation of the best-found solution value. The algorithms were evaluated using the Wilcoxon's signed rank test with a significance level $\alpha = 0.05$. Additionally, the graphs of the convergence speed and crossover success on some of the problems were added.

4.2 Experimental results and analysis

The results are presented in Figures 1–3. The first column denotes the benchmark problems, while the next columns represent the results for each of three algorithms.

Problem	rand/1/bin		rand/1/aco, W_{Sin}		rand/1/aco, W_{Step}	
	Min	**Median**	**Min**	**Median**	**Min**	**Median**
F1	**1.2013E+03**	**1.9285E+03**	3.4088E+03	4.5717E+03	3.1084E+03	4.2304E+03
F2	**4.4489E+02**	**4.4489E+02**	**4.4489E+02**	**4.4489E+02**	**4.4489E+02**	**4.4489E+02**
F3	6.0081E+02	6.0118E+02	**6.0000E+02**	**6.0000E+02**	**6.0000E+02**	**6.0000E+02**
F4	8.8934E+02	9.0569E+02	**8.7811E+02**	**8.9974E+02**	8.7385E+02	9.0149E+02
F5	9.0460E+02	9.0958E+02	**9.0000E+02**	**9.0000E+02**	**9.0000E+02**	**9.0000E+02**
F6	**1.8205E+03**	**1.8296E+03**	3.4404E+05	1.2936E+06	3.6763E+05	1.3165E+06
F7	2.0351E+03	2.0403E+03	**2.0322E+03**	**2.0397E+03**	2.0342E+03	2.0398E+03
F8	2.2290E+03	2.2312E+03	2.2273E+03	**2.2287E+03**	**2.2267E+03**	2.2309E+03
F9	**2.4807E+03**	**2.4807E+03**	**2.4807E+03**	**2.4807E+03**	**2.4807E+03**	**2.4807E+03**
F10	**2.5004E+03**	2.5006E+03	**2.5004E+03**	**2.5004E+03**	**2.5004E+03**	2.5005E+03
F11	2.9001E+03	3.0000E+03	**2.9000E+03**	**2.9000E+03**	**2.9000E+03**	**2.9000E+03**
F12	**2.9289E+03**	**2.9322E+03**	2.9319E+03	2.9322E+03	2.9319E+03	2.9322E+03

Fig. 1. Min and median values of the algorithm comparison on CEC 2022 benchmark in 20D.

Problem	rand/1/bin		rand/1/aco, W_{Sin}		rand/1/aco, W_{Step}	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**
F1	**1.8946E+03**	2.9866E+02	4.5721E+03	5.7142E+02	4.3287E+03	5.6777E+02
F2	**4.4489E+02**	0.0000E+00	**4.4489E+02**	0.0000E+00	**4.4489E+02**	0.0000E+00
F3	6.0128E+02	2.8726E-01	**6.0000E+02**	6.5000E-05	**6.0000E+02**	6.6000E-05
F4	9.0579E+02	7.2156E+00	**8.9883E+02**	7.2636E+00	8.9921E+02	7.3193E+00
F5	9.0978E+02	3.2517E+00	**9.0000E+02**	1.3600E-04	**9.0000E+02**	3.0000E-05
F6	**1.8290E+03**	3.7310E+00	1.2779E+06	4.3419E+05	1.4135E+06	5.3653E+05
F7	2.0414E+03	3.4238E+00	**2.0392E+03**	2.8968E+00	2.0400E+03	2.4124E+00
F8	2.2311E+03	9.5502E-01	**2.2286E+03**	7.9194E-01	2.2306E+03	1.3814E+00
F9	**2.4807E+03**	0.0000E+00	**2.4807E+03**	0.0000E+00	**2.4807E+03**	0.0000E+00
F10	2.5006E+03	8.2387E-02	**2.5004E+03**	4.2517E-02	2.5005E+03	4.3401E-02
F11	2.9800E+03	4.0635E+01	**2.9013E+03**	6.0918E+00	2.9133E+03	3.4574E+01
F12	2.9329E+03	3.2729E+00	**2.9325E+03**	6.5262E-01	2.9329E+03	1.7569E+00

Fig. 2. Mean and std values of the algorithm comparison on CEC 2022 benchmark in 20D.

vs. rand/1/bin	Total	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
rand/1/aco, W_{Sin}	5/5/2	-	=	=	+	+	-	+	+	=	=	+	=
rand/1/aco,W_{Step}	3/7/2	-	=	=	+	+	-	=	=	=	=	+	=

Fig. 3. Wilcoxon rank-sum test at the 0.05 significance level.

Figure 3 demonstrates the comparison of the algorithms using the Wilcoxon rank-sum test with the significance level α = 0.05. The symbols indicate that the algorithm performed significantly better («+»), worse («-»), or not different («=») compared to the basic DE.

It can be seen from Figures 1-3 that in the most cases, the *rand/1/aco* strategy was significantly better than or equal to the *rand/1/bin* scheme. The proposed algorithm with both W_{Sin} and W_{Step} was significantly worse in problems F1, which is unimodal and non-separable, and F6, which is a hybrid function with many local minima. Overall, although the performance of W_{Sin} and W_{Step} differs slightly, W_{Sin} demonstrated the best mean value in 10 out of 12 problems and had the best results in total.

The convergence diagrams are shown in Figures 4-7. The X-axis denotes current number of fitness function evaluations (FEV), the Y-axis shows the best-found objective function value. The colored charts describe the algorithms convergence during the run.

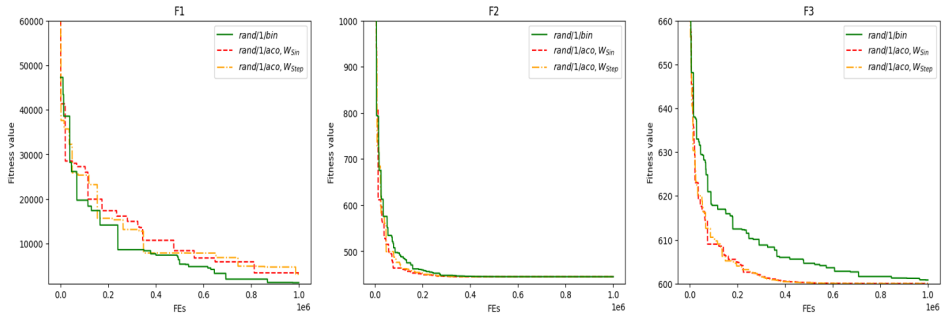

Fig. 4. Convergence graphs comparison on problems F1 – F3.

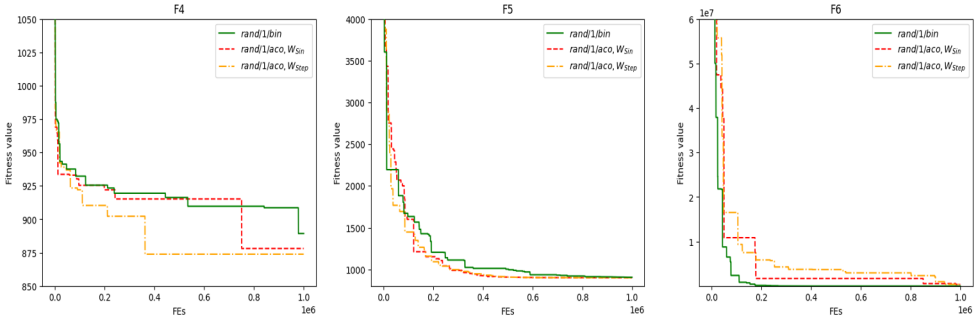

Fig. 5. Convergence graphs comparison on problems F4 – F6.

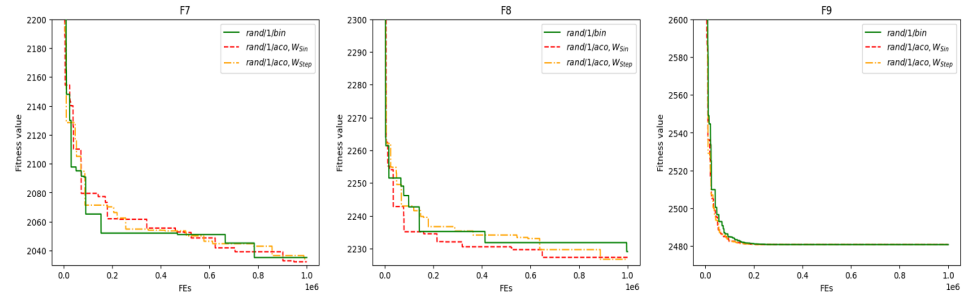

Fig. 6. Convergence graphs comparison on problems F7 – F9.

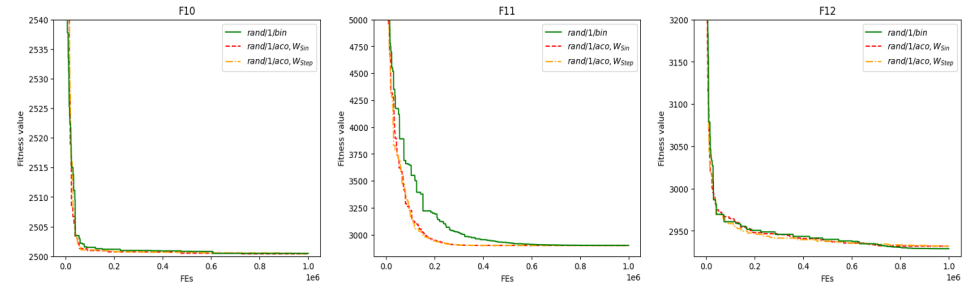

Fig. 7. Convergence graphs comparison on problems F10 – F12.

As we can see in Figures 4–7, the convergence speed of the proposed strategy is higher or equal to the *rand/1/bin* scheme, except for the functions, on which *rand/1/aco* performed significantly worse. It worth noting that the W_{Step} variant tends to a more frequent solution improvement.

We have also visualized the rate of the successful crossover during the run. The X-axis denotes current number of fitness function evaluations (FEV), the Y-axis shows the mean value of the number of successful crossover operations in a single generation. The line shows the smoothed crossover success rate values, while the background coloring demonstrates the spread of the mean value. The graphs are shown in Figures 8–11.

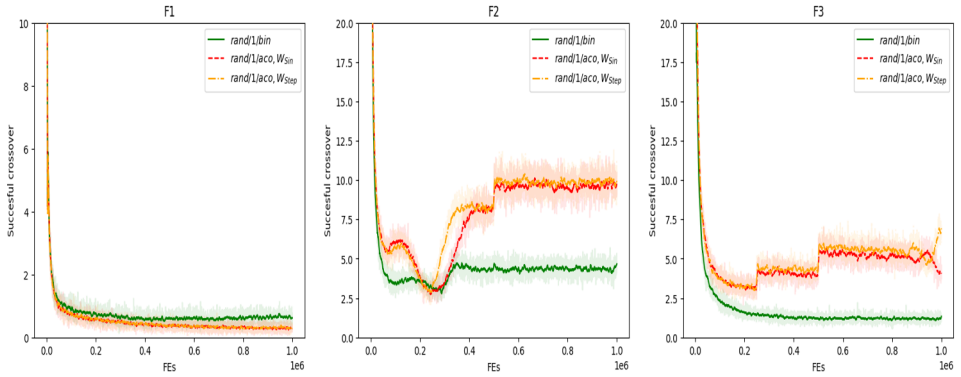

Fig. 8. Crossover success rate comparison on problems F1 – F3.

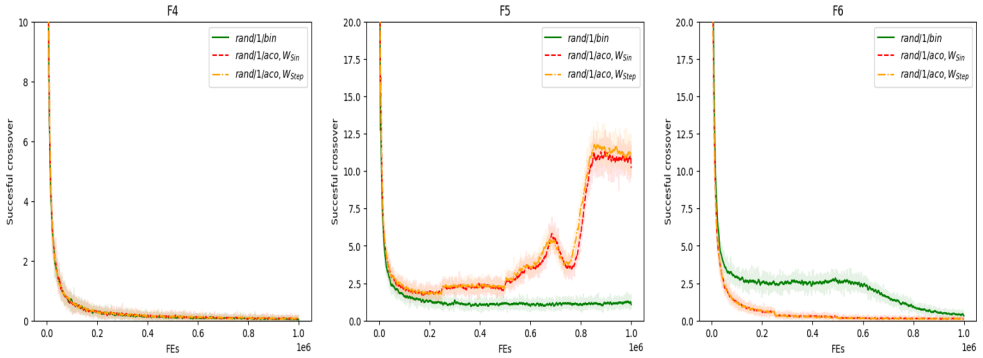

Fig. 9. Crossover success rate comparison on problems F4 – F6.

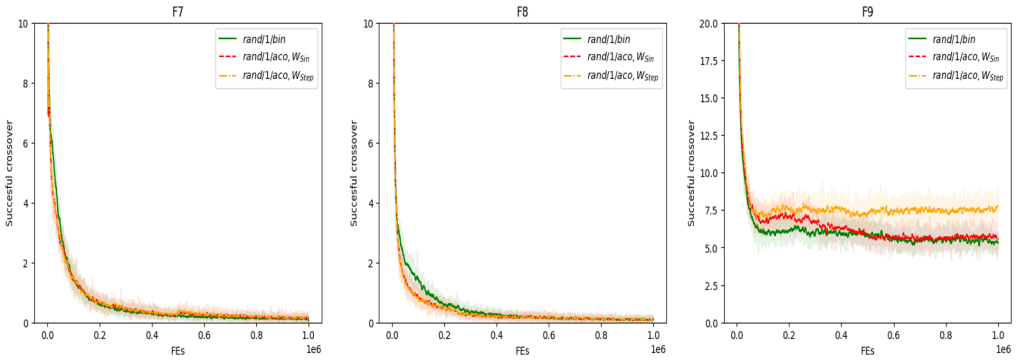

Fig. 10. Crossover success rate comparison on problems F7 – F9.

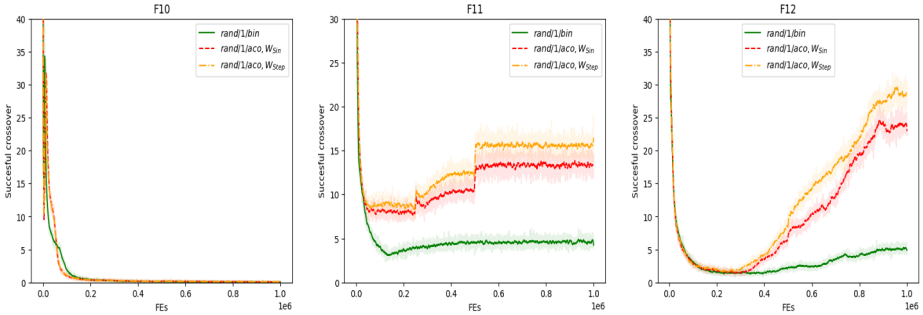

Fig. 11. Crossover success rate comparison on problems F10 – F12.

Overall, as it can be seen in Figures 8–11, the success crossover rate was noticeably higher in the *rand/1/aco* scheme, especially with W_{Step} weight distribution strategy. The exceptions were the problems, in which the algorithm performed worse, and cases when the convergence was too fast at the earlier stages of evolution.

5 CONCLUSION

The paper proposes a novel adaptive crossover strategy for differential evolution algorithm. The proposed scheme extends the classic binomial crossover to a multivariate form and uses the information about the behavior of the objective function on a separate variable or subcomponent. Two approaches based on this method were investigated on 12 CEC 2017 benchmark functions for single objective bound-constrained numerical optimization. Numerical experiments demonstrated that the component-based crossover adaption scheme can improve the DE performance on complex optimization problems.

In further work, we will continue the research of the strategy in order to improve its performance and robustness, apply it to more benchmark and engineering problems. Also, the developed scheme can be implemented to the more complex DE-based methods.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

References

1. Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang, Artificial Intelligence Review **55(1),** 59-110 (2022).
2. D. Molina, A. LaTorre, and F. Herrera, Cognitive Computation **8,** 517–544 (2018).
3. R. Storn and K. Price, Journal of Global Optimization **11,** 341–359 (1997).
4. S. Das and P.N. Suganthan, IEEE Transactions on Evolutionary Computation **15(1),** 4-31 (2011).
5. C. Lin, A. Qing, Q. Feng, J. Heuristics, **17,** 675–703 (2011).
6. A. Kumar, K. Price, A. K. A. A. Mohamed, and P. N. Suganthan, *Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization* (Nanyang Technological University, Singapore, 2021).

7. R. Tanabe and A. Fukunaga, Success-history based parameter adaptation for Differential Evolution in *IEEE Congress on Evolutionary Computation, CEC* 2013, (2013) pp. 71-78.
8. J. Brest, M. S. Maucec, and B. Boskovic, Single objective real-parameter optimization: Algorithm jSO in *2017 IEEE Congress on Evolutionary Computation (CEC)*, (2017) pp. 1311–1318.