The comparison of different PDP-type self-adaptive schemes for the cooperation of GA, DE, and PSO algorithms

Anton Sopov[1*], and *Tatiana* Karaseva[2]

[1]Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarsk, Russia
[2]Siberian Federal University, Department of Business Informatics and Business Process Modeling, Krasnoyarsk, 660074, Russia

Abstract. Many global optimization problems are presented as a black-box model, in which there is no information on the objective function properties. Traditional optimization algorithms usually can't effectively solve that kind of problems. Different heuristics and metaheuristics are usually applied in that case. Evolutionary algorithms are one of the most popular and effective approaches to black-box optimization problems. However, it's hard to choose one specific method that will solve the given problem better than other algorithms. For dealing with this issue, self-adaptive schemes are usually implemented. In this paper we have investigated the performance of different PDP-type adaptive schemes using such popular evolutionary-based algorithms as Genetic Algorithm, Differential Evolution, and Particle Swarm Optimization. The experimental results on a set of benchmark problems have shown that investigated schemes can improve the performance compared with the performance of a stand-alone evolutionary algorithm. At the same time the choice of a scheme and its parameters affect the results.

1 Introduction

The global optimization problem can be stated in the following way (1):

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

here $f(\boldsymbol{x})$ is an 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. The search space (S) is defined by the lower value l_i and upper value r_i, which are determined for each dimension: $l_i \le x_i \le r_i, i = \overline{1,n}$.

The solution to the problem can be stated as below (2):

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

here x^* can be stated as the problem solution, if the objective function value in x^* is lower than the value of every possible solution in the search space. In other words, the problem solution matches the global minima of the objective function. It means that the problem can

* Corresponding author: antonsopov2004@gmail.com

be describes as the global minimization problem, as it indicated in form (1). There are no additional requirements or conditions for the objective function f and we don't use any properties of the objective function and its derivatives. Therefore, the function itself can be described as a black-box model.

For that case stochastic population-based algorithms can be implemented. In the study we have used several of them to estimate their performance on its own, and then have implemented 2 self-adaptive approaches and have compared the results with the original algorithms.

The paper is structured the following way. Section 2 describes approaches for solving black-box optimization problems that have been used in the work. The implementation of self-adaption schemes is shown in the Section 3. Section 4 describes the setup, discussion, and detailed results of implemented numerical experiments. Finally, in the Section 5, the conclusion and further research are presented.

2 Related work

2.1 Evolutionary algorithms

Traditional optimization algorithms usually can't effectively solve black-box optimization problems. Different heuristics and metaheuristics are usually applied in that case. One of the most wide-spread and efficient metaheuristics for solving global optimization problems are evolutionary algorithms, which are population-based approaches that are closely related to biological evolution [1, 2].

Binary genetic algorithm (GA) is one of the most common evolutionary-based algorithms. This method is represented by several biology-inspired operators, including initialization, selection, crossover, and mutation procedures, each of them has different schemes of performing the process. At the beginning, a population of agents is initialized randomly. Next, on every iteration selection, crossover, and mutation are applied to each individual. Selection and crossover operator are used to form new offspring population, while mutation is needed to prevent the degeneration by keeping the population diverse. Fitness function is evaluated for each individual in the population, which is the metric for the suitability of solutions. The most common approach for GA uses the binary form for coding of individuals [3].

GA has a lot of options for performing its operations. The most common selection schemes are proportionate (roulette wheel selection), rank, and tournament. The crossover is usually presented in two main ways: uniform or one-point. In our work we have chosen two GA settings with different crossover schemes, because it affects search space distribution of the population and can potentially bring to the algorithm a better diversity.

Differential evolution (DE) is another global search algorithm. In comparison with GA, it calculates candidate solutions using the mutation vector made of several random agents. The canonical version of this method consists of four operators, but, unlike in GA, mutation performs first and determines an offspring [4, 5]. Many modifications of the mutation strategy were implemented, including the ones with the population division and the parameters' adaption.

Another class of heuristics is swarm algorithms. Particle swarm optimization (PSO) is one of popular swarm-based methods. At first, a population of agents and the velocity vector for each of them are initialized randomly. Next, in every iteration, the movement of each particle is determined by velocity changes, which is calculated using the position and fitness of other individuals in the population and the best-found solution [6]. Many other approaches based on the canonical PSO were implemented, one of them is the Fully Informed Particle

Swarm algorithm, which, besides that, uses information about topological neighbors of individuals [7].

These methods are widely used in solving hard black-box optimization problems. The choice of method can be a complex task, as every algorithm can show different performance depending on the objective function properties.

2.2 Self-adaptive schemes

Each of the algorithms presented above demonstrates different performance depending on the objective function characteristics. As the problem is presented as a black-box model, it's hard to choose one specific method that will solve the problem more efficiently that the others. In that case self-adaptive schemes are implemented. The most common way uses a set of existing operators or methods, independently runs them at the same time or switch them, and updates the preference of components, which perform better.

We have chosen the Population-Level Dynamic Probabilities scheme (PDP) [8, 9]. The main idea of this method is the probabilities distribution based on the success rate of algorithms in the cooperation. After the certain number of generations, during the run of the adaptation operator, the better-performed algorithms are given a higher probability, while probabilities of other components are decreased. We have implemented the similar approach that redistributes population sizes of component algorithms instead of probabilities.

We have proposed 2 different PDP approaches in this work. The first one (denoted as PDP-best) determines the component algorithm with the best fitness and increases its population, while decreasing the population size of every other algorithm. The second one (denoted as PDP-ranked) redistributes the population based on the rank of each of the component algorithms.

3 Proposed approach

PDP approach was originally implemented to use it inside a single algorithm as a self-adjustment of probabilities for choosing one of operators. As we have several methods to choose from, several modifications were proposed. Firstly, every component algorithm is given the same number of agents. Then, after each adaptation period with length of k generations, the adaption operator is used to evaluate new population sizes for each method.

In this study we use implementations of PDP adaptive schemes as presented in Tables 1 and 2 using pseudocode.

Table 1. The PDP-best scheme implementation.

Line	Pseudocode
1	Set Δ, k, PS
2	Generate an initial population for each of algorithms, with the same PS size
3	**while** *stop criteria* is not satisfied **do**
4	Run k generations for each algorithm
5	Determine the algorithm with the best performance
6	**if** *population size* of the best performed algorithm is less that *higher boundary*, **do**
7	Redistribute the population size: move Δ random solutions from each other algorithm to the pool
8	Add the pool population to the best performed algorithm

9	Replace the worst solution of each algorithm instance with overall best solution
10	**end if**
11	**end while**

Table 2. The PDP-ranked scheme implementation.

Line	Pseudocode
1	Set Δ, k, PS
2	Generate an initial population for each of algorithms, with the same PS size
3	**while** *stop criteria* is not satisfied **do**
4	Run k generations for each algorithm
5	Move Δ random solutions from each algorithm to the pool
6	Determine the rank of each algorithm using the best solution fitness as a criterion
7	Redistribute the population size: move Δ random solutions from the pool to each of the component algorithm based on its rank
8	Replace the worst solution of each component algorithm with overall best solution
9	**end while**

In tables above, Δ is a parameter that determines how many individuals are taken from each algorithm before the redistribution process. To prevent the decreasing to zero of population sizes and, consequently, eliminating an algorithm from cooperation, the minimum and maximum of population sizes are defined. The higher boundary of the possible population size was set to $2n\Delta$, where n is a number of competing algorithms, and the lower was set to 4Δ. Also, to keep the competitiveness, the elitism feature is implemented, which copies the overall best solution to every algorithm population and therefore gives every component algorithm a chance to continue the search from the same position.

As there are many options for setting parameters for self-adaptive approaches, it's important to estimate the performance of different settings and choose the best one. Also, it's necessary to compare the performance with the performance of stand-alone algorithms that are used in self-adaptive schemes.

4 Numerical experiments

4.1 Experimental settings

We have compared two different adaptive schemes with several parameter configurations. We used the following 6 algorithms: the binary genetic algorithm with uniform (titled as GA_1) and one-point crossover (GA_2), the differential evolution algorithm with two non-standard mutation strategies: *DE/rand-to-best/1* (DE_RB) and *DE/current-to-best/1* (DE_CTB), the canonical PSO and the Fully Informed PSO (FIPS). In order to investigate the performance of every method, we have chosen a set of 4 benchmark optimization problems, which are multimodal and are known as hard: Ackley, Rastrigin, Griewank, and Rosenbrock functions.

The PDP-best and PDP-ranked schemes were tested with 2 different Δ values. The settings for algorithms used in these schemes are standard, except for ones that are used in both adaptive and basic methods. Both of GA use the tournament selection and the standard

mutation probability. The population size for each algorithm is equal to 50, which means that the overall population size for both of self-adaptive schemes is equal to 300. The adaptation period is equal to 10 generations for each run.

The number of variables for optimization problems is equal to 10 and 20 for each problem. We used the maximum number of fitness evaluations as a stopping criterion. The number of fitness evaluations is 100000 for problems with 10 variables and 400000 for problems with 20 variables. The performance of all algorithms is evaluated using 40 independent runs, and estimated using the minimum and median of the best-found value, and the error of finding the global optimum.

All algorithms in the comparison are implemented using the Python programming language with no external packages. In order to improve the computational performance, the multi-core parallelization was used.

4.2 Experimental results

The results for 10 and 20 variables are presented in Tables 3-6. The first column denotes the name of each algorithm. Other columns represent the results for each benchmark problem we used in the study. All of these columns contain two criteria: the objective function value of the best-found solution (f) and the error value.

Table 3. The experimental results for Ackley and Rastrigin problems with 10 variables.

Function	**Ackley**		**Rastrigin**	
Algorithm	f	**Error**	f	**Error**
PDP-best, $\Delta = 5$	**7.63E-16**	**4.72E-15**	**9.31E-05**	9.96E-02
PDP-best, $\Delta = 3$	6.02E-12	2.76E-11	7.33E-02	8.86E-01
PDP-ranked, $\Delta = 5$	6.86E-11	1.66E-10	8.00E-02	1.85E-00
PDP-ranked, $\Delta = 8$	5.97E-13	8.35E-12	1.33E-03	**5.30E-02**
GA_1	7.96E-10	2.85E-10	1.75E-01	8.52E+00
GA_2	7.52E-13	8.01E-13	1.38E-02	8.04E-01
DE_RB	3.17E-14	5.24E-13	4.03E-01	7.26E+00
DE_CTB	6.75E-11	2.05E-10	7.71E-02	4.38E+00
PSO	3.17E-14	3.57E-13	9.00E-04	9.24E-02
FIPS	1.50E-12	8.34E-12	5.58E-04	6.05E-02

Table 4. The experimental results for Griewank and Rosenbrock problems with 10 variables.

Function	**Griewank**		**Rosenbrock**	
Algorithm	f	**Error**	f	**Error**
PDP-best, $\Delta = 5$	8.09E-05	5.71E+00	3.61E-06	5.10E-04
PDP-best, $\Delta = 3$	4.60E-05	6.12E-01	9.79E-07	7.86E-05
PDP-ranked, $\Delta = 5$	**7.11E-06**	**3.86E-01**	4.29E-07	3.28E-05
PDP-ranked, $\Delta = 8$	8.02E-03	7.72E-01	5.97E-06	1.60E-03
GA_1	5.08E-02	8.13E+00	8.13E-04	9.56E-03

GA_2	7.25E-03	4.60E+00	8.06E-04	8.61E-03
DE_RB	2.75E-04	6.59E-01	**1.57E-12**	**5.09E-09**
DE_CTB	9.84E-04	8.29E-01	6.18E-06	1.66E-04
PSO	5.86E-05	6.16E-01	6.34E-07	7.47E-05
FIPS	5.52E-05	6.01E-01	1.60E-04	8.24E-03

Table 5. The experimental results for Ackley and Rastrigin problems with 20 variables.

Function	**Ackley**		**Rastrigin**	
Algorithm	***f***	**Error**	***f***	**Error**
PDP-best, $\Delta = 5$	8.59E-11	6.62E-10	**4.75E-03**	4.09E-01
PDP-best, $\Delta = 3$	7.24E-07	1.86E-06	9.23E-01	8.74E+00
PDP-ranked, $\Delta = 5$	2.00E-08	7.41E-07	5.91E-01	5.28E+00
PDP-ranked, $\Delta = 8$	7.58E-09	1.75E-08	5.07E-01	3.35E+00
GA_1	7.92E-07	5.04E-07	4.25E+00	6.28E+00
GA_2	3.52E-10	5.26E-10	6.77E-01	6.87E+00
DE_RB	**6.25E-11**	**8.77E-11**	8.06E+00	2.06E+00
DE_CTB	1.75E-09	8.18E-09	1.01E-01	**3.07E-01**
PSO	2.63E-10	2.07E-10	9.42E-02	8.70E-01
FIPS	9.91E-09	5.80E-08	8.05E-02	7.04E-01

Table 6. The experimental results for Griewank and Rosenbrock problems with 20 variables.

Function	**Griewank**		**Rosenbrock**	
Algorithm	***f***	**Error**	***f***	**Error**
PDP-best, $\Delta = 5$	2.03E-04	3.86E+00	4.86E-04	8.85E-02
PDP-best, $\Delta = 3$	2.86E-04	6.60E-01	3.86E-04	4.29E-02
PDP-ranked, $\Delta = 5$	**6.60E-05**	**3.07E-01**	5.62E-05	9.50E-04
PDP-ranked, $\Delta = 8$	7.96E-01	5.49E+00	7.58E-04	7.01E-03
GA_1	5.69E-02	2.08E+00	4.75E-02	6.93E-01
GA_2	6.00E-02	6.28E+00	8.27E-02	1.06E-01
DE_RB	7.07E-01	8.57E+00	**7.48E-09**	**2.76E-07**
DE_CTB	9.02E-02	9.20E+00	2.27E-04	6.02E-03
PSO	1.86E-04	7.05E-01	2.86E-06	8.40E-05
FIPS	4.38E-04	8.52E-01	5.70E-03	7.28E-02

We have also visualized the population distribution between the algorithms during the best run for each adaption scheme. The X-axis denotes the population size per single algorithm, the Y-axis shows the current total number of fitness function evaluations (FEV). The colored charts describe the population size changes during the run, using FEV as a time factor.

The growth of an algorithm chart at the specific FEV value means that the algorithm had the best performance during the PDP-best scheme adaption or was successful enough during the PDP-ranked adaption. If the population size didn't change at that moment, it means that the population size reached the lower or higher limit in case of PDP-best, or returned the exact same number of individuals from the pool during the PDP-ranked adaption.

An example for Griewank and Rosenbrock functions with 20 variables is presented on Figures 1-4.

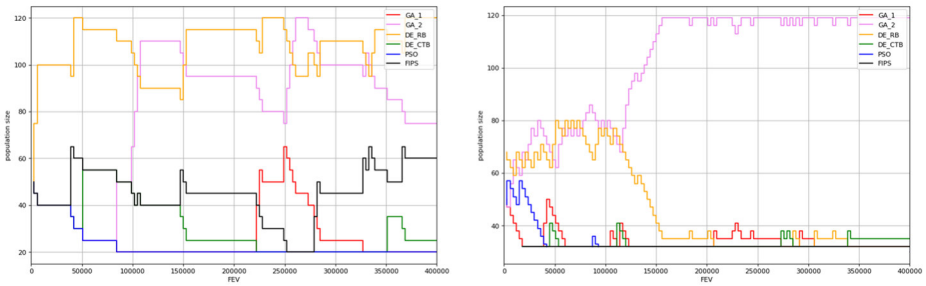

Fig. 1. The population distribution plot on the Griewank problem of PDP-best scheme, $\Delta = 5$ (left) and $\Delta = 3$ (right).

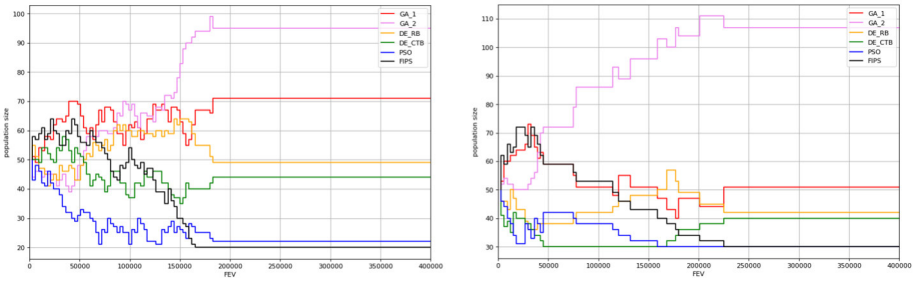

Fig. 2. Population distribution plot on the Griewank problem of PDP-ranked scheme, $\Delta = 5$ (left) and $\Delta = 8$ (right).

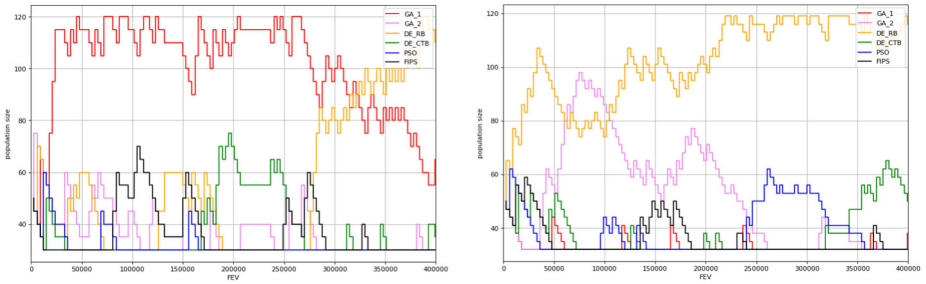

Fig. 3. Population distribution plot on the Rosenbrock problem of PDP-best scheme, $\Delta = 5$ (left) and $\Delta = 3$ (right).

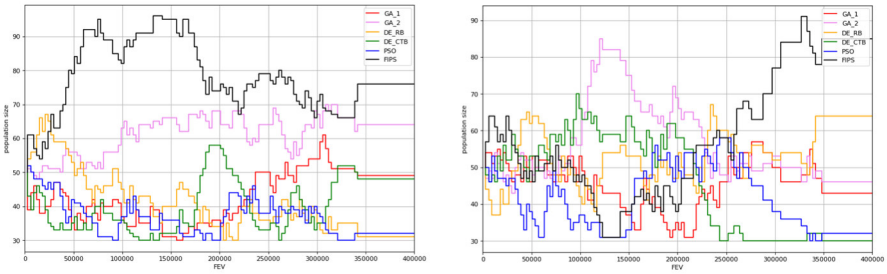

Fig. 4. Population distribution plot on the Rosenbrock problem of PDP-ranked scheme, $\Delta = 5$ (left) and $\Delta = 8$ (right).

4.3 Discussion

As we can see from Table 3-6, PDP-best with the higher pool size Δ could find a better solution compared to other PDP settings. The higher Δ value gives better algorithm stability in the population size distribution. The Figure 1 shows that GA with one-point crossover and DE with *DE/rand-to-best/1* mutation scheme was demonstrating better performance almost during the whole run. The least competitive method on the Griewank problem was the classic PSO, while its modification got the better result with the $\Delta = 5$.

On the Rosenbrock problem (Figure 3) both of GA schemes in PDP-best have shown the better performance than in the other experiments. It worth noting that DE became the most populated closer to the end of the run, although it had the minimum possible population size value for a long time before.

As we can see in Figure 2, the PDP-ranked scheme converges to a set of fixed population size values and doesn't change them until the stopping criteria is reached. It happens because of the constant low ranking of the worse performing algorithms, which means that the population distribution is always near the lower limit and therefore is unable to change. On the ravine Rosenbrock function (Figure 4) the PDP-ranked scheme keeps the competitiveness between the algorithms longer. GA with one-point crossover performed better on the multimodal problems, while the Fully-Informed PSO showed better result on Rosenbrock function.

The implemented schemes performed better than the basic algorithms on two benchmark optimization problems. On the ravine functions some of the used algorithms, especially DE with *DE/rand-to-best/1* mutation scheme, showed better performance.

Overall, both of self-adaptive schemes demonstrate the similar performance, but the population size distribution in the PDP-best approach continues to change during the whole run, while the stagnation is observed in the PDP-ranked scheme. It's possibly solvable by changing the approach of population distribution or setting other boundaries of possible population size values. Also, both of self-adaptive approaches had different the best performing component algorithm on the same optimization problems.

5 Conclusions

In the study, we have implemented and evaluated two approaches of self-adaptive evolutionary algorithm schemes for solving global optimization problems presented by a black-box model. The proposed approaches were investigated on a set of benchmark optimization problems. Numerical experiments have shown that the self-adaptive schemes can improve the performance of stand-alone evolutionary algorithms on hard multimodal and ravine optimization problems. At the same time the performance highly depends on the

parameter settings for self-adaptive approaches. In further work, we will investigate modifications of self-adaptive algorithms in order to decrease the effect of the parameter setting.

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

References

1. D. Javier et al., Swarm and Evolutionary Computation, 48 (2019)
2. D. Molina, A. LaTorre, F. Herrera, Cognitive Computation **8**, (2018)
3. K. Sastry, D. Goldberg, G. Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (Springer, 2nd ed., 2005)
4. R. Storn, K. Price, Journal of Global Optimization 11, (1997)
5. A. Vakhnin, E. Sopov, Algorithms **14**, 146, (2021)
6. J. Kennedy, R. Eberhart, *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks. IV (1995)
7. L. Szymon, A. Kowalski, *Fully Informed Swarm Optimization Algorithms: Basic Concepts, Variants and Experimental evaluations*, Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (2014)
8. E. Sopov, Computational Intelligence. IJCCI 2015. Studies in Computational Intelligence **669**, (2017)
9. J. Niehaus, J. Banzhaf, Lecture Notes in Computer Science **325**, 36, (2001)