jSO and GWO Algorithms Optimize Together

Radka Poláková[1], Daniel Valenta[1]

[1]Silesian University in Opava, Faculty of Philosophy and Science in Opava, The Institute of Computer Science, Opava, Czech Republic

Abstract

This paper deals with global optimization. There are many algorithms for global optimization in the literature. In this text, we focus on two effective optimizers. The first one is an adaptive version of Differential evolution algorithm which was the most successful version of the algorithm on CEC 2014 congress and is called the jSO algorithm. The second one is the Grey wolf optimizer which was introduced in 2014. We propose new algorithm *cooperation* in which both of these algorithms are used together to get better results when optimization problems are being solved. In our attempt, both algorithms take turns making the optimization process. We tested the proposed algorithm on four multi-modal functions on two levels of dimension. The results are quite promising.

Keywords

Optimisation, GWO, jSO, Cooperation, Optimization algorithms

1. Global Optimization

In human life, there are many different problems which lead to optimization of a mathematical function. To optimize a function f in a search space means to find a point in the search space in which the function f has global optimum, thus global minimum or global maximum, which depends on the task. It is well-known that when we are searching for global maximum of function f, it is the same as finding a global minimum of function $g = -f$, so we can talk only about minimization in the following. The minimization problem is defined as follows. Let's have a function f,

$$f : S \to \mathcal{R}, \quad S \subset \mathcal{R}^D \tag{1}$$

f is minimized function, D is dimension of problem. S is search space, here continuous.

$$S = \Pi_{j=1}^{D}[a_j, b_j]; \quad a_j < b_j, \quad j = 1, 2, \ldots, D \tag{2}$$

Π notation (also called Product notation, Cartesian in this case) is used here in the standard way and indicates repeated multiplication.

A point $\vec{x^*}$ is global minimum point of the function f in the search space S if the following condition holds.

$$\forall \vec{x} \in S; \quad f(\vec{x^*}) \leq f(\vec{x}) \tag{3}$$

It is possible to find the minimum of a function by analytical way, but there are functions for which such process is difficult, long, or impossible because of function features. Then stochastic algorithms could help us.

The most difficult functions to solve with optimization algorithms are those that are not easily differentiable at some points or have many local extremes. The result may not always correspond exactly to the real optimum. However, it could be enough with regard to the complexity of the optimized function. In other words, the result of a stochastic optimizer is a sufficient estimation of the real solution of the problem, and it is not so expensive. According to the No Free Lunch theorem [1], there is no optimization algorithm that solves all functions best. Therefore, new methods are being developed. We describe two already established and one new method in the following sections.

2. Differential Evolution and jSO

There are many different algorithms to optimize-minimize a mathematical function. In this paper, we work with jSO which is an adaptive version of Differential evolution algorithm and also with Grey wolf optimizer. We describe all three algorithms briefly in this section.

2.1. Differential Evolution

Differential evolution (DE) algorithm was introduced in [2]. It is an efficient optimizer. It is population-based algorithm in which a population evolves during the run in order to have better and better members. Better in a sense of lower function value in the point. The best member of the population is the result at the end of the run. It is the point of global optimum found by the algorithm.

The algorithm works with the population of points which are at the beginning of the run of the algorithm generated randomly (with uniform distribution) in the search space S. Then, each population member is moved around the search space and is evolved to have better values of optimized function.

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia

radka.polakova@fpf.slu.cz (R. Poláková);
daniel.valenta@fpf.slu.cz (D. Valenta)

In a generation (iteration) of the algorithm run, a trial point $\vec{y_i}$ is computed for each population member $\vec{x_i}$, $i = 1, 2, \ldots, NP$, NP is the size of the population. The point $\vec{y_i}$ is produced by two operations, the fist one is mutation. A mutant point $\vec{u_i}$ is computed based on a kind of mutation. Then the mutant $\vec{u_i}$ enters into crossover operation together with the original point $\vec{x_i}$. The result of the crossover is the trial point $\vec{y_i}$. If the trial point $\vec{y_i}$ is better than the original point $\vec{x_i}$, the new point $\vec{y_i}$ enters into the next generation instead of $\vec{x_i}$. If not, the original point $\vec{x_i}$ enters into the next generation of the population.

There are several types of mutation, e.g. rand/1, randrl/1, current-to-rand/1, rand/2, current-to-pbest/1, etc. The most used one is the rand/1 mutation - eq. (4). By this mutation, each mutant $\vec{u_i}$ is computed from three randomly chosen points $\vec{r_{1_i}}$, $\vec{r_{2_i}}$, and $\vec{r_{3_i}}$ from current population which are different from original point $\vec{x_i}$.

$$\vec{u_i} = \vec{r_{1_i}} + F(\vec{r_{2_i}} - \vec{r_{3_i}}) \tag{4}$$

The F is mutation parameter of DE. Also the crossover could be made in one of several ways in DE. The most frequently used two types of crossover are binomial and exponential ones. The algorithm uses parameter CR as a crossover parameter. It influences the count of coordinates which are inherited by a trial point from the mutant point $\vec{u_i}$. In the binomial crossover, inherited coordinates are distributed uniformly. The trial point takes consecutive series of coordinates from the mutant in the exponential crossover.

Thus, DE algorithm has several parameters. They are mutation parameter F, crossover parameter CR, mutation type, type of crossover, and also population size NP. There are many adaptive versions of the algorithm in the literature. Several of them were successful on CEC congresses, e.g. SHADE [3], LSHADE [4], jSO [5].

The principles of the algorithm are shown in the pseudo-code below. Note that this is a simplification, which does not describe how the trial point was created. The trial point $\vec{y}_i$ to point $\vec{x}_i$ is created using two operations, mutation and crossover. We will discuss the specific method of creating the trial point in the pseudo-code presented in the following section for the jSO algorithm (Algorithm 2).

Algorithm 1 DE

```
Generate the initial generation P_0 of the population P; P_0 = (x_1, x_2, ..., x_NP)
Compute the value of the optimized function f at all points of the generation P_0;
Set counter of generations to g = 0;
while termination criterion is not met do
    Q_g = P_g;
    for i = 1 to NP do
        Create a trial point y_i to point x_i;
        Compute the value of the function f at point y_i;
        if f(y_i) ≤ f(x_i) then
            Insert point y_i into Q_g instead of point x_i;
    P_{g+1} = Q_g; g = g + 1;
The result is the best point in P_g.
```

2.2. jSO

jSO [5] is very sophisticated version of DE. The algorithm evolved from its predecessors. The first one, of course except DE, is JADE [6]. Authors of this algorithm developed a new mutation, current-to-pbest/1. This mutation in slightly modified version is used in jSO. They also suggested to employ an archive of old members of the population. The next algorithm is SHADE [3], which is improved version of JADE. SHADE employs historical circle memories to adapt F and CR parameters. Then the LSHADE algorithm [4] was proposed, it is SHADE with a linear reduction of population size mechanism. The next algorithm is iLSHADE [7]. And finally, the last one is jSO. The jSO algorithm uses the linear reduction of population size mechanism, archive, and other tools. It has several features different from iLSHADE, e.g. size of the population is set to $NP = 25 \times \sqrt{D} \times logD$ at the beginning of the search process instead of $NP = 18 \times D$, parameter p, which is the parameter of current-to-pbest/1 (and also of current-to-pbest-w/1) mutation, is handled in a different way in this algorithm compared to its predecessors. The size of historical circle memories H is set to 5 here. For a detailed description of the algorithm, see [5] and Algorithm 2.

2.3. Grey Wolf Optimizer

Grey wolf optimizer (GWO) is a nature-based and already well-established meta-heuristic method inspired by social dynamics found in the pack of grey wolves [8]. In other words, the algorithm simulates the behavior of wolves, that live and hunt together in packs. The algorithm was introduced in 2014.

Let us focus on principles observed in a pack of wolves. There are strict rules that they must follow, and each wolf has a clearly defined role. Based on this, we can classify them into four categories: alpha, beta, delta, and omega. The leader of the pack is the alpha pair of wolves. They are dominant in the group and other wolves follow their lead. They could be substituted by beta wolves if it is necessary. Beta wolves are second in command. They are important because they help and support the alpha pair during its decisions. Delta wolves follow instructions ordered by wolves alpha and beta. Mid-ranking wolves

Algorithm 2 jSO

Generate the initial generation P_0 of the population P randomly; $P_0 = (\vec{x}_1, \vec{x}_2, ..., \vec{x}_{NP})$
Initialize archive $A = \emptyset$;
Compute the value of the optimized function f at all points of the generation P_0;
Set all values in M_F and M_{CR} to 0.5;
Note: M_F and M_{CR} are circle memories, storing the position parameters for the Cauchy (M_F) and normal (M_{CR}) distributions;
The size of these memories is H; $H = 5$;
$g = 0$ (current iteration - generation);
$evals = NP$ (current number of used function evaluations);
$k = 1$ (index counter for circle memories);
while termination criterion is not met **do**
    $S_F = \emptyset$ and $S_{CR} = \emptyset$;
    for $i = 1$ to NP **do**
        Select r_i randomly from $\{1, 2, ..., H\}$;
        if $r_i = H$ **then**
            $M_{F,r_i} = 0.9$; $M_{CR,r_i} = 0.9$;
        if $M_{CR,r_i} < 0$ **then**
            $CR_{i,g} = 0$;
        else Generate CR using normal distribution:
            $CR_{i,g} = N_i(M_{CR,r_i}, 0.1)$;
        if $evals < \frac{1}{4}\ maxevals$; $maxevals$ is the maximum number of allowed function evaluations;
        then
            $CR_{i,g} = max(CR_{i,g}, 0.7)$;
        else if $evals < \frac{1}{2}\ maxevals$; **then**
            $CR_{i,g} = max(CR_{i,g}, 0.6)$;
        Generate F using Cauchy distribution:
        $F_{i,g} = C_i(M_{F,r_i}, 0.1)$;
        if $evals < \frac{1}{6}\ maxevals$ and $F_{i,g} > 0.7$ **then**
            $F_{i,g} = 0.7$;
        A trial point $\vec{y}_{i,g}$ is created using DE/current-to-pbest-w/1/bin strategy (see [5]);
    Evaluate optimized (objective) function f in all NP made trial points $\vec{y}_{i,g}$, $i = 1, 2, \ldots, NP$;
    $evals = evals + NP$;
    for $i = 1$ to NP **do**
        if $f(\vec{y}_{i,g}) \leq f(\vec{x}_{i,g})$ **then**
            Update point for the next generation:
            $\vec{x}_{i,g+1} = \vec{y}_{i,g}$
        else $\vec{x}_{i,g+1} = \vec{x}_{i,g}$
        if $f(\vec{y}_{i,g}) < f(\vec{x}_{i,g})$ **then**
            Insert $\vec{x}_{i,g}$ into archive A;
            Insert $CR_{i,g}$ into S_{CR}; and $F_{i,g}$ into S_F;
    If necessary, shrink archive A;
    Calculate the new value of the first parameter for both distributions S_F and S_{CR}, and store them in M_{F_k} and M_{CR_k}; $k = k + 1$;
    if $k > H$ **then** $k = 1$;
    Apply linear population size reduction mechanism, see [5] (update NP and population P);
    Update parameter p for current-to-pbest-w/1 mutation strategy (see [5]);
    $g = g + 1$;
The result is the best point in P_g.

in the hierarchy are delta ones. They ensure the routine activities of the pack and follow the orders of alpha and beta wolves. Each delta wolf has a specific focus, based on which we can divide them into the following subcategories: scouts, sentinels, and caretakers. The omega wolves are in the lowest position in the hierarchy and others wolves often pick on them. It is important to filter aggression and prevent frustration in the pack. Losing omega wolves can cause fighting between other wolves, and damage to discipline or hierarchy.

In nature, wolves' primary goal is to find and hunt down prey. This process consists of these main steps: searching for prey, encircling prey, and attacking prey. During searching for the prey, wolves are trying to find the most abundant (but easily catchable) prey to hunt. Once they find such prey, they attempt to push it into a situation when it is alone and cannot escape while they encircle it. Finally, when the prey is surrounded and can no longer escape, they attack it. Wolves attack the weak spots of the prey like legs, snout, or belly, until it stops resisting, and afterwards, they bring it down and crush its windpipe.

The Grey wolf optimizer is inspired by the processes described above: the creation of a social hierarchy and hunting technique. Because it is an agent-based algorithm, each agent represents one of the wolves in the pack. Agents are randomly placed into the environment (search space S). The better value (in the sense of minimization) in the position of the current wolf the closer position to the prey (the solution of the solved optimization problem).

GWO is an iterative algorithm. In each iteration, each wolf is assigned to the pack hierarchy according to the value of its fitness function f at its position. The wolf with the best value is ranked as alpha, the second best as beta, the third best as delta, and all the others as omega. Wolves alpha, beta, and delta have the same meaning and save the three best solutions found at the iteration. Positions of wolves in the environment are updated in each iteration. The new position of the agent is based upon the estimated location of the prey, which is probably somewhere between alpha, beta, and delta wolves. We assume this, but the optimum may be located elsewhere, so a mechanism is needed to thoroughly scan the entire environment. The agent approaching the prey hunts it, while the agent moving away from it tries to find even more abundant prey elsewhere. For this purpose, two vectors $\vec{A}$ and $\vec{C}$ are used, thanks to which the algorithm passes smoothly through two phases: scouting and hunting the prey. Both vectors have random components so they help to prevent convergence to a local optimum (not very abundant prey) instead of a global one.

Vector $\vec{A}$ has components $rand(-1, 1) * a$, where $rand(-1, 1)$ generates a random number with a uniform

distribution between -1 and 1 and $a = 2 - (2it/it_{max})$, while it is the algorithm current iteration and it_{max} is the maximum number of iterations. Each component of the vector influences movement of the agent in a specific dimension of the environment (search space). As you can see, the interval in which components of vector $\vec{A}$ lie is narrowing we can say linearly from $[-2, 2]$ to $[0, 0]$ as the number of iterations increases. It is because the parameter a decreases during the whole run from 2 to 0. The closer is the value of component of $\vec{A}$ to 0, the higher is the probability that the agent chooses the hunting phase instead of the scouting one.

Another vector supporting divergence between scouting and hunting phases is $\vec{C}$. Vector $\vec{C}$ is similar to vector $\vec{A}$, but the values of components of this vector do not linearly decrease as the number of iterations grows. This vector has components set to $rand(0, 2)$, which is a random number with the uniform distribution between 0 and 2. The closer the value is to 0, the higher is the probability that the agent chooses the hunting phase. This vector helps wolves to behave more naturally. In nature, there are various obstacles (e.g. bushes, stones, or trees) on hunting trails. Wolves change direction to avoid them, so they do not move directly towards their prey. Vector $\vec{C}$ simulates this part of their behaviour.

Each wolf-agent is on position $\vec{X_j}$ in search space S, j is index of wolf, $j = 1, 2, \ldots, N$, where N is the size of wolf pack. Positional vectors of agents $\vec{X_j}$ are updated according to the formula $\vec{X_j}(it + 1) = \frac{\vec{X_1}+\vec{X_2}+\vec{X_3}}{3}$, where $\vec{X_1}$, $\vec{X_2}$, and $\vec{X_3}$ represent potential new positions of the prey to move (positions close to optimum) and are computed in the following way:

$$\vec{X_1} = \vec{X_\alpha}(it) - \vec{A_1} * \vec{D_\alpha}, \tag{5}$$

$$\vec{X_2} = \vec{X_\beta}(it) - \vec{A_2} * \vec{D_\beta}, \tag{6}$$

$$\vec{X_3} = \vec{X_\delta}(it) - \vec{A_3} * \vec{D_\delta}, \tag{7}$$

where $\vec{X_\alpha}(it)$, $\vec{X_\beta}(it)$, and $\vec{X_\delta}(it)$ are current positions of alpha, beta, and delta wolf, respectively. They represent the current best three preys found, $\vec{A}$ is already defined above and is generated separately for each of $\vec{X_\alpha}(it)$, $\vec{X_\beta}(it)$, and $\vec{X_\delta}(it)$ wolves, so we get $\vec{A_1}$, $\vec{A_2}$, and $\vec{A_3}$. Vectors $\vec{D_\alpha}$, $\vec{D_\beta}$, $\vec{D_\delta}$ represent the distance of the wolf $\vec{X_j}$ from prey. It is computed in the following way:

$$\vec{D_\alpha} = |\vec{C_1} * \vec{X_\alpha}(it) - \vec{X_j}(it)|, \tag{8}$$

$$\vec{D_\beta} = |\vec{C_2} * \vec{X_\beta}(it) - \vec{X_j}(it)|, \tag{9}$$

$$\vec{D_\delta} = |\vec{C_3} * \vec{X_\delta}(it) - \vec{X_j}(it)|, \tag{10}$$

where $|\vec{X}|$ is vector whose components are the absolute values of components of $\vec{X}$. Vector $\vec{C}$ was already defined above and is generated separately for each of $\vec{X_\alpha}(it)$, $\vec{X_\beta}(it)$, and $\vec{X_\delta}(it)$ wolves, similarly as $\vec{A}$, so we get $\vec{C_1}$, $\vec{C_2}$, and $\vec{C_3}$.

GWO has only two parameters, the size of the wolf pack N and the length of the time it can search for the optimum. In the original proposal, it works with the maximum of iterations it can make. Here, we use the maximum number of function evaluations, in order to do a fair comparison of algorithms.

Algorithm 3 GWO

1: Randomly generate an initial population of wolves-agents $\vec{X_1}$, $\vec{X_2}$, ..., $\vec{X_N}$ into the environment;
2: **while** termination criterion is not met **do**
3: Calculate the fitness value of each agent $\vec{X_j}$;
4: Determine the social hierarchy and find position of alpha, beta, and delta wolves;
5: Generate vectors $\vec{A}$ and $\vec{C}$ for all three best wolves;
6: Calculate the new position of each agent $\vec{X_j}$;
7: The result is the position of the best wolf after the last iteration of the algorithm.

Because wolves are moving closer toward prey from various directions with the increasing number of iterations, they are encircling it.

3. Cooperation of Algorithms

We proposed to use both algorithms, jSO and GWO, together in the optimization process of a function. The used idea of common use of both algorithms is very simple. We wanted to do a part of optimization process by one of two mentioned algorithms and then give the results of the algorithm to the other algorithm to do another part of optimization process and then after making its part of the process, it gives its results to the first algorithm, etc. Each algorithm needs some time to optimize a function. It is like when someone needs some time to do something he has to do. It is not the best idea to let somebody do something and immediately after he starts doing it to stop him. Mentioned ideas led us to divide the number of allowed function evaluations into several (not many) parts, we divide the amount into k ($k = 10$) portions here. The length of a portion is l. And then, jSO makes 5 of these parts and GWO does the rest of the parts (also 5 parts). jSO starts doing optimization process and after spending such amount of function evaluation that equals to l, it gives its population to GWO. GWO takes the best N ($N = 6$, the size of the pack of wolves) points of received population and spends next l function evaluations. After each iteration, GWO in *cooperation* tests if its current optimum is better than the optimum in the population which was originally received from

jSO. And if so, it rewrites randomly chosen points in the "temporary" population which is prepared for next work of jSO. When it is the last run of GWO in *cooperation*, it rewrites the whole received population by the wolves, when the best point found in the iteration is better than the interim result. After spending its amount of function evaluations, it gives prepared population (with the best found point) again to jSO. And the process is repeated 5 times. Thus, the last algorithm which works in the search process by our *cooperation* algorithm is GWO.

We wanted to use all advantages of both original algorithms, so we put them together in a way that keeps all the principles of the original algorithms. So, all parameters of both algorithms are set to the values according their original proposal, see [5], [8]. When jSO optimizes, the size of population is gradually linearly decreased. Also, parameter p decreases as proposed in [5]. When the GWO algorithm works inside *cooperation*, also parameter a decreases (as proposed in [8]) during the whole run of *cooperation*. The size of the pack of wolves is equal to N during the whole run of the *cooperation* algorithm.

Algorithm 4 *Cooperation* algorithm

1: Generate the initial generation P_0 of the population P; $P_0 = (\vec{x}_1, \vec{x}_2, ..., \vec{x}_{NP})$; NP is set as in the jSO algorithm;
2: $l = \frac{maxevals}{k}$; k is the total number of runs of both algorithms in *cooperation*; $k = 10$; $maxevals$ is the total number of allowed function evaluations;
3: Set the iteration counter c to 1;
4: $r = 0$;
5: **while** $c \leq k$ **do**
6: **if** $c \bmod 2 = 1$ **then**
7: Read population P_r as input;
8: Run the jSO algorithm for l evaluations;
9: $r = r + 1$
10: *Note:* the output of this run of jSO is P_r;
11: **else if** $c \bmod 2 = 0$ **then**
12: Read the N best points from P_r into $\vec{X}_1$, $\vec{X}_2$, ..., $\vec{X}_N$; $N = 6$ in this case;
13: Run the GWO algorithm for l evaluations;
14: **if** the condition about currently found best point and the best point of P_r holds (see text) **then**
15: **if** $c < k$ **then**
16: Rewrite N points in P_r by current positions of $\vec{X}_1$, $\vec{X}_2$, ..., $\vec{X}_N$ but do not affect the best three points of P_r;
17: **else**
18: Rewrite whole P_r by pack $\vec{X}_1$, $\vec{X}_2$, ..., $\vec{X}_N$
19: $c = c + 1$;
20: The result is the best point in P_r.

Table 1
Search spaces, optimal values, and position of optimal values for used test functions

function	search space S	$f(\vec{x^*})$	$\vec{x^*}$
Ackley	$[-30, 30]^D$	0	$(0, 0, 0, \ldots, 0)$
Rastrigin	$[-5.12, 5.12]^D$	0	$(0, 0, 0, \ldots, 0)$
Rosenbrock	$[-10, 10]^D$	0	$(1, 1, 1, \ldots, 1)$
Happycat	$[-50, 50]^D$	0	$(0, 0, 0, \ldots, 0)$

The presented cooperative algorithm is one of the first attempts to use the GWO and jSO algorithms together. In order to achieve even better results and to find the optimal number of repetitions of the sequence of both algorithms, it would be necessary to perform experiments on a larger number of optimization functions with a variety of features.

4. Experiments and Results

We computed optima by three algorithms, jSO, GWO, and proposed *cooperation* in this paper. Four multi-modal functions were used to make an experimental tests. Used functions are Ackley function - eq. (11), Rastrigin function - eq. (12), Rosenbrock function - eq. (13), and Happycat function - eq. (14).

$$f(\vec{x}) = -20\,\exp\left(-0.02\,\sqrt{\tfrac{1}{D}\sum_{j=1}^{D} x_j^2}\right) - \\ -\exp\left(\tfrac{1}{D}\sum_{j=1}^{D}\cos 2\pi x_j\right) + 20 + \exp(1) \tag{11}$$

$$f(\vec{x}) = 10\,D + \sum_{j=1}^{D}\left[x_j^2 - 10\cos(2\pi x_j)\right] \tag{12}$$

$$f(\vec{x}) = \sum_{j=1}^{D-1}\left[100(x_j^2 - x_{j+1})^2 + (1 - x_j)^2\right] \tag{13}$$

$$f(\vec{x}) = \left|\sum_{j=1}^{D} x_j^2 - D\right|^{1/4} + \\ +\tfrac{1}{D}\left(0.5\sum_{j=1}^{D} x_j^2 + \sum_{j=1}^{D} x_j\right) + 0.5 \tag{14}$$

Used search space S of each used function is displayed in Table 1. In this table, global optimum and point of the optimum are written too. It is not clearly visible, but each of these functions has many local extremes.

Algorithms were tested on two levels of dimension, $D = 10$ and $D = 30$, in this work. In each dimension, we set the total amount of allowed function evaluations to two different values, for $D = 10$, the two values were 3000 and 30000, for $D = 30$, the two values were 10000 and 100000. For each combination of algorithm, function, dimension, and value of allowed function evaluations,

Table 2
Results of three tested algorithms on Ackley, Rastrigin, Rosenbrock, and Happycat functions in dimensions $D = 10$ and $D = 30$ with two different amounts of allowed function evaluations, 3000 and 30000 for $D = 10$, 10000 and 100000 for $D = 30$

func	D	maxevals	jSO mean std	GWO mean std	coop mean std
ack	10	3×10^3	1.2005	**0.0019**	0.0973
			0.1801	**0.0037**	0.1164
		3×10^4	2.1E-06	**4.4E-16**	**4.4E-16**
			5.0E-06	**0**	**0**
	30	1×10^4	1.4459	**1.6E-11**	2.8E-06
			0.0869	**6.2E-11**	6.3E-06
		1×10^5	0.0027	**4.4E-16**	**4.4E-16**
			0.0016	**0**	**0**
ras	10	3×10^3	24.443	**0.0098**	1.0742
			5.0200	**0.0263**	1.8183
		3×10^4	4.8E-08	**0**	**0**
			9.5E-08	**0**	**0**
	30	1×10^4	125.44	**0**	0.0664
			11.986	**0**	0.2570
		1×10^5	0.6580	**0**	**0**
			0.8321	**0**	**0**
ros	10	3×10^3	12.188	9.0068	**8.6566**
			3.3652	**0.0327**	0.3620
		3×10^4	**1.9E-06**	8.9895	0.5470
			5.7E-06	0.0131	0.3062
	30	1×10^4	57.495	28.993	**28.421**
			12.379	**0.0040**	0.3609
		1×10^5	**12.716**	28.974	19.871
			0.8105	**0.0554**	0.7234
hac	10	3×10^3	0.7336	1.7077	**0.6613**
			0.2225	0.3172	**0.1212**
		3×10^4	**0.1094**	1.8145	0.1652
			0.0232	0.3412	0.0316
	30	1×10^4	**0.7271**	1.8126	0.9969
			0.1310	0.2919	0.1742
		1×10^5	**0.2262**	2.0396	0.2878
			0.0362	0.3823	**0.0283**

we made 15 runs. The total amount of runs in our experiments was 720.

All tested algorithms were implemented in GNU Octave, version 7.1.0 and all computations were carried out on a standard PC with Windows 10 Home, Intel(R) Core(TM) i7-7500U CPU 2.70GHz 2.90GHz, 8 GB RAM.

Summarisation of experimental test results is written in Table 2. We highlight the best results in bold. Results are also displayed on the two figures above. There is a boxplot for shorter runs on the left side and the boxplot for longer ones on the right side for all four displayed combinations of function and dimension on these figures.

For Ackley function and dimension $D = 10$ in longer runs, GWO reaches very good values, they are very near optimum. Results of jSO are only a little worse here. The *cooperation* algorithm adopts the results of GWO in this case. In shorter runs, GWO reaches better results than jSO, and when both algorithms optimize together in the *cooperation* algorithm, the results are only a little worse than the results of GWO, but better than the results of jSO. For dimension $D = 30$ in longer runs, GWO reaches again very small values (reached values of jSO are worse) and *cooperation* adopts them again. In shorter runs here, the situation is similar or a little better than for shorter runs in dimension $D = 10$.

When we discuss the optimization process of Rastrigin function with tested algorithms, the situation is very similar to the previous case, in both tested dimensions for both lengths of runs.

For Rosenbrock function for both tested dimensions for shorter runs, GWO is better optimizer than jSO and *cooperation* is little better then GWO. But when algorithms have much more time (larger amount of function evaluations), jSO is better then GWO and *cooperation* reaches better results than GWO but not better than jSO reaches.

When we think about the Happycat function, the results of tests are very similar in both dimensions. It holds for both lengths of runs. We mean the results of comparison of algorithms. Moreover, it seems that the results of *cooperation* are a little better than the results of the jSO algorithm (which is here the better one of GWO and jSO algorithms) for shorter runs and lower dimension. For longer runs in both dimensions, jSO is better then GWO and *cooperation* reaches better results than GWO but a little worse results than jSO.

5. Conclusion

The new algorithm called *cooperation* for global optimization was proposed in this paper. It is based on two very effective optimizers, Grey wolf optimizer and one of many adaptive versions of Differential evolution algorithm, the jSO algorithm.

We proposed to use both algorithms together for optimization. The idea of *cooperation* is very easy, the algorithms take turns performing the optimization process. Four multi-modal functions and two levels of dimension were selected for our experimental comparison. The results of the made experimental comparison are promising.

In this paper, we have used only a basic scheme in which each algorithm has been used repeatedly 5-times for the *cooperation* algorithm. In future work, we plan to develop some more sophisticated schema for the change of controlling of optimization process by these two algorithms, probably based on the stagnation of search

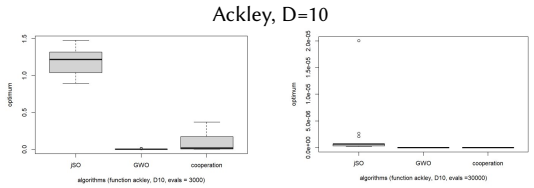

Figure 1: Results of all three tested algorithms on Ackley and Rastrigin functions

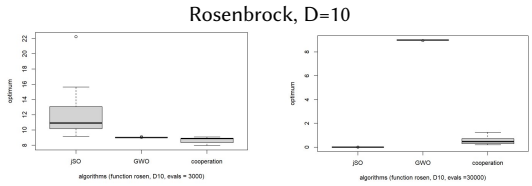

Figure 2: Results of all three tested algorithms on Rosenbrock and Happycat functions

process by currently used algorithm. The goal is to find a scheme that works best for most of the functions tested, ideally better than both original algorithms in most cases.

Acknowledgement: This work was supported by the project no. CZ.02.2.69/0.0/0.0/18_054/0014696, "Development of R&D capacities of the Silesian University in Opava", co-funded by the European Union.

Acknowledgments

This work was supported by the project no. CZ.02.2.69/0.0/0.0/18_054/0014696, "Development of R&D capacities of the Silesian University in Opava", co-funded by the European Union.

References

[1] Wolpert D. H., Macready, W. G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation. **1** (1997) 67–82

[2] Storn R., Price, K.: Differential evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J. Global Optimization. **11** (1997) 341–359

[3] Tanabe R., Fukunaga, A.: Success-history based parameter adaptation for differential evolution. In *IEEE Congress on Evolutionary Computation 2013*. (2013) 71–78

[4] Tanabe R., Fukunaga, A.: Improving the Search Performance of SHADE Using Linear Population Size Reduction. In *IEEE Congress on Evolutionary Computation 2014*. (2014) 1658–1665

[5] Brest J., Maučec M. S., Bošković B.: Single Objective Real-Parameter Optimization: Algorithm jSO. In *IEEE Congress on Evolutionary Computation 2017*. (2017) 1311–1318

[6] Zhang J., Sanderson A. C.: JADE: Adaptive Differential Evolution With Optional External Archive. IEEE Transactions on Evolutionary Computation. **13** (2009) 945–958

[7] Brest J., Maučec M. S., Bošković B.: iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *IEEE Congress on Evolutionary Computation 2016*. (2016) 1188–1195

[8] Mirjalili S., Mirjalili S. M., Lewis A.: Grey Wolf Optimizer. Advances in Engineering Software. **69** (2014) 46–61