Two Effective Algorithms Optimize Together

Daniel Valenta[1,*], Radka Poláková[1,†]

[1]Silesian University in Opava, Faculty of Philosophy and Science, Institute of Computer Science, 746 01 Opava, Czech Republic

Abstract

In this paper, we focus on the ongoing research of the Cooperation algorithm, which combines features of the established and successful GWO and jSO optimization algorithms. We introduce a new enhancement using the stagnation parameter, which ensures switching between the two algorithms when no further improvements are achieved during the optimization process. We test different settings of the stagnation parameter and find its effect on the performance of the Cooperation algorithm.

Keywords

Optimization algorithms, jSO, GWO, Cooperation algorithm, Stagnation

1. Introduction

In this paper, we deal with two well-established and successful optimization algorithms, namely jSO [2] and GWO [3]. We have introduced the earliest experiment on the cooperation of these two methods in [1]. The results of that experiment were promising, which motivated us to continue our research by modifying the proposed algorithm and testing it on more functions. The progress and results of the continuing research will be presented in this paper. First, we briefly introduce the two main algorithms. Then we will introduce a new method for their cooperation, test it on functions of CEC2014 benchmark set [7], and evaluate the effectiveness of the proposed approach.

2. GWO and jSO

GWO and jSO are well-known, successful, and well-established optimization algorithms. Like many other optimization methods, they are stochastic, heuristically derived, and inspired by nature.

2.1. GWO

GWO or also called Grey Wolf Optimizer, was introduced in 2014 by S. Mirjalili and his collaborators [3]. It found inspiration in the hunting and social hierarchy of the wolf pack. The GWO algorithm is multi-agent system. Individual agents represent wolves that move in the environment. The environment is represented by a real function instead of our planet in GWO. The movement of the agents simulates the movement of real grey wolves in their search and hunt for prey and their cooperation with each other.

Let us focus on the hierarchy in the wolf pack. A pair of Alpha wolves are the highest in the hierarchy, are pack leaders and have reproductive duties. The Betas support their decisions and give them feedback. Deltas take care of routine tasks - protecting the pack in case of danger, taking care of old and sick wolves, assisting with hunting, etc. Omegas are the lowest in the hierarchy, for example they can eat last and other wolves can take their frustrations out on them, which helps keep the pack stable.

The hierarchy is also applied in a simplified form in the GWO algorithm. The point in the environment at which the wolf is currently located has a concrete value computable by a fitness function. The fitness function computes the value of the function at the position of the agent that corresponds to a possible solution to the optimization problem. The 3 wolves with the best fitness function values at the moment are labeled as Alpha, Beta, and Delta. From the point of view of GWO, they are equal. And all other wolves are marked as Omega.

Another important characteristic of the wolf pack that GWO is inspired by is the method of hunting. In the real world, wolves hunt in the following phases: searching for prey, stalking and closing in on it, encircling it, and once the prey is encircled, coming into attack to the weak area. GWO distinguishes between a *Searching for prey phase*, when wolves tend to move away from the best solution found so far to the optimization problem, and a *Hunting phase*, when they move closer to the "prey" (currently best position found so far, based on positions of agents Alpha, Beta, and Delta).

Agents, like wolves, move around the environment with the goal of finding the best possible food. Because it is an iterative algorithm, in each iteration each of the wolf-agents moves to a new position. Before each move,

ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Tatranské Matliare, Slovakia

*Corresponding author.

†These authors contributed equally.

daniel.valenta@fpf.slu.cz (D. Valenta);
radka.polakova@fpf.slu.cz (R. Poláková)
0009-0005-0781-7755 (D. Valenta); 0000-0002-0782-5668 (R. Poláková)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3498/paper17.pdf

agents decide whether to hunt (Hunting phase) for the best prey found so far (it moves in direction to currently the best result found), or to further explore the environment (Searching for prey phase) where even more abundant prey may be found (the truly global optimum). Which phase each agent chooses depends on two random vectors, $\vec{A}$ and $\vec{C}$. Each agent has both vectors assigned to it. The vector $\vec{C}$ has components $rand(0, 2)$, where $rand(0, 2)$ generates a random number between 0 and 2 with uniform distribution. The vector $\vec{A}$ has components $rand(-1, 1) \times a$, where $rand(-1, 1)$ generates a random number between -1 and 1 with uniform distribution and $a = 2 - (2it/it_{max})$, where it is the current iteration of the algorithm and it_{max} is the maximum number of iterations of the algorithm, also specified as a termination criterion. The components of both vectors affect the movement of the agent in each dimension of the environment. The closer the value of the component is to 0, the more probably the wolf is to hunt in a given iteration (chooses the Hunting phase), and the closer it is to 2, the more likely the wolf is to explore the environment elsewhere (chooses the Searching for prey phase). Vector $\vec{A}$, unlike vector $\vec{C}$, depends on the current iteration of the algorithm and ensures that wolves explore the environment more in the initial iterations (with higher probability) and hunt towards the end of the algorithm. The vector $\vec{C}$ is purely random and simulates various obstacles in the environment, analogous to nature. As a result, agents tend to choose both phases independently of the iteration of the algorithm. This is needed to reduce the probability of converging to a local optimum instead of finding a global one in later iterations.
Now, we put both vectors into the context of calculating the new wolf-agent position. This is calculated as follows:

$$\vec{X_j}(it + 1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3},$$

where X_j is an agent with index j, it is the current iteration of the algorithm, and $\vec{X_1}$, $\vec{X_2}$, and $\vec{X_3}$ represent potential new positions of the agent, which are calculated based on the positions of the three best Alpha, Beta, and Delta wolves as follows:

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

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

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

$\vec{X_\alpha}(it)$, $\vec{X_\beta}(it)$, and $\vec{X_\delta}(it)$ are the positions of agents Alpha, Beta, and Delta in the current iteration it of the algorithm, $\vec{A_1}$, $\vec{A_2}$, and $\vec{A_3}$ are three different instantiations of the vector $\vec{A}$, for each of the agents Alpha, Beta, and Delta, and $\vec{D_\alpha}$, $\vec{D_\beta}$, and $\vec{D_\delta}$ represent the distance of the agent from the prey, but due to the effect of randomness of instances of vector $\vec{C}$, they are an approximate estimates, and are calculated as follows:

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

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

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

where $\vec{C_1}$, $\vec{C_2}$, and $\vec{C_3}$ are three different instances of the vector $\vec{C}$, for each of the agents Alpha, Beta, and Delta, and $\vec{X_j}(it)$ is the position of the wolf in the current iteration of the algorithm.
The symbol $\times$ in the calculations of $\vec{X_1}$, $\vec{X_2}$, and $\vec{X_3}$ and $\vec{D_\alpha}$, $\vec{D_\beta}$, and $\vec{D_\delta}$ represents multidimensional vector multiplication by components. For example, for $D = 2$, the calculation is performed as follows: $(x_1, y_1) * (x_2, y_2) = (x_1 * x_2, y_1 * y_2)$.
Having described all the principles, we now describe the sequence of steps of the algorithm. The input of GWO is the environment definition, in our case we use the CEC 2014 functions. The next input is the number of agents (pack size). The next and last input is the termination criterion, which can be defined by the maximum number of iterations, or alternatively by the maximum running time of the algorithm in the number of computations of the objective function value.
The pseudocode of the algorithm is as follows:

1. in each position of the environment where the agent is located, the fitness function is calculated,
2. based on fitness values, agents are ranked in a hierarchy: the agent with the best value becomes Alpha, the second best Beta, the third best Delta, and all others Omega,
3. vectors $\vec{A}$ and $\vec{C}$ with random components are generated for each agent j,
4. it calculates new position of each wolf $\vec{X_j}$ in search space,
5. finally, the algorithm checks to verify that the termination criterion - usually the expiration of the maximum number of iterations - has already been met; if so, the algorithm ends; if not, the algorithm continues with the first step.

2.2. jSO

The jSO algorithm is a successful population-based optimization method that is an improved version of the iL-SHADE algorithm [5] derived from differential evolution DE [4]. Its improved strategy uses the weighted mutation *current-to-pbest/1 /bin* and other tools like an archive to store solutions which were rewritten in population by better points and circle memory for storing

parameters of distributions for generating new values of DE parameters computed based on successful values of these parameters.

An equally interesting feature of jSO is that its population size is linearly reduced like in its predecessors L-SHADE [6] and iL-SHADE [5]. Behind the success of jSO is also the appropriate setting of its parameters. We describe these principles in detail in the following text.

The jSO is an evolutionary algorithm that works with a population that is evolving. It is an adaptive variant of the Diferential Evolution (DE) algorithm. The algorithm works with a population of points that evolve (move to different positions in the search space) by several operations, mutation, crossover, and selection. It uses the already mentioned *current-to-pbest-w/1* strategy as a mutation strategy for selecting a so-called trial point, which is a potential new point for the next generation. The advantage of this strategy is that it adaptively controls its parameters and that it uses the archive. Thanks to this mutation strategy, the algorithm selects one point from the top few in the population to work with it and also can use a point from the archive. The formula is as follows:

$$\vec{v_{i,G}} = \vec{x_{i,G}} + F_w(\vec{x_{pbest,G}} - \vec{x_{i,G}}) + F_i(\vec{x_{r_1,G}} - \vec{x_{r_2,G}}),$$

where $\vec{x_{i,G}}$ is the $i-th$ point of generation G. F_w, F_i are the parameters of the point $\vec{x_i}$ from interval $[0, 1]$ that determines the weights of the differences in the mutation, $\vec{x_{r_1}}$, $\vec{x_{r_2}}$ are two different randomly selected points, the first one is randomly selected from the population P in generation G, and the second one is randomly selected from the union of population and the archive, and $\vec{x_{pbest,G}}$ is one of the $NP \times p$ best points in generation G (randomly selected), where NP is the size of population and $p \in [0, 1]$ is a dynamic parameter randomly affecting the number of points from which one is selected for this mutation (this helps to balance between exploitation and exploration phases).

The strategy *current-to-pbest-w/1/bin* additionally includes the binomial crossover operator. The parameter CR of this operator is adapted randomly. It is the probability that a component of the mutant $\vec{v_{i,G}}$ will be used in the trial point. CR is a random number with normal distribution whose first parameter (mean of the distribution) depends on successful values of CR used before. Each component from $\vec{v_{i,G}}$ is written into the trial point with probability CR, when the component does not write, relevant component of original $\vec{x_{i,G}}$ is used. If the trial point is selected as a new point for the next generation, the original $\vec{x_{i,G}}$ is stored in archive A.

More information about this mutation strategy can be found in [2].

The algorithm has a lot of input parameters that need to be set during the initialization phase. First, an initial generation P_0 of size NP is randomly generated, i.e. $P_0 = (\vec{x_1}, \vec{x_2}, ..., \vec{x_{NP}})$. At this point, the archive A is empty, so $A = \emptyset$. Also, for each point in the initial population P_0, the value of the objective function is calculated. Finally, the default values of the parameter settings are set during initialization.

As another tool, the jSO algorithm uses two circle memories: M_F for storing the parameters of Cauchy distribution for generating values of mutation parameter F and M_{CR} for storing the parameters of normal distribution for generating values of crossover parameter CR. The size of these circle memories is $H = 5$ and they have all values set to 0.5 when the algorithm is initialized.

The jSO algorithm works as described in the following pseudocode. After initializing the population points and the parameters listed above, the while loop follows these steps:

1. For each point generate a random number r in the range $1, 2..., H$, where H is the size of the circle memory. If $r = H$, set both M_{CR_r} and M_{F_r} to 0.9. If M_{CR_r} is not greater than or equal to 0, set CR to 0. And finally, if $M_{CR_r} > 0$, generate CR using normal distribution $N(M_{F_r}, 0.1)$.
2. For each point, generate parameter F using Cauchy distribution $C(M_{F_r}, 0.1)$.
3. If the current number of evaluations is less than quarter of the maximum number of evaluations, the probability CR is limited to $max(CR, 0.7)$ for the point, and if the number of evaluations is less than half, CR is limited to $max(CR, 0.6)$ for the point.
4. If the current number of evaluations is less than six tenths of the maximum number of evaluations, the new value of parameter F is limited to $max(F, 0.7)$ for the point.
5. For each point $\vec{x}$ from population, a new trial point $\vec{y}$ is created using *DE/current-to-pbest-w/1/bin* strategy, and then the value of the objective function f in $\vec{y}$ is computed.
6. If $f(\vec{y})$ is better than $f(\vec{x})$, update the point $\vec{x}$ by $\vec{y}$ for further evolution. Otherwise, discard the trial point $\vec{y}$ and keep the original $\vec{x}$.
7. If point $\vec{x}$ is updated (by trial point $\vec{y}$ in the previous step), put it in the archive A. If necessary, shrink archive A. Also insert value of parameter CR into S_{CR} and F into S_F.
8. After doing steps 1-7 for all points in population calculate the new value of the first parameter for both distributions used for generating F and CR from S_F and S_{CR}. Then, put them into M_{F_k} and M_{CR_k}.
9. Increase the pointer k in the circular memory by 1, and if it is greater than H, set $k = 1$ again.
10. Apply linear population size reduction mechanism and update parameter p for mutation *current-to-pbest-w/1*.

11. If the termination criterion is not met, the computation continues by repeating the cycle for the new generation $G = G + 1$. Otherwise, the computation terminates and the result is the best point in population P at current generation G.

3. Cooperation algorithm

As we have already mentioned, both algorithms GWO and jSO are very useful for solving optimization problems and provide very good results in solving them. According to the No free Lunch theorem [8], it is not possible to say which one is better, each is suitable for different types of tasks, which is confirmed also by our testing results on CEC 2014 functions. For some functions, GWO provides better results, while for others jSO provides better results. The results of both algorithms are shown in Tables 1 and 2. We wondered how to take advantages of both of them. And so we introduced the first version of the Cooperation algorithm using the findings of GWO and jSO in [1]. The principle of Cooperation is based on switching both GWO and jSO algorithms. Already in this first early version, which we tested only on a few functions (optimization problems), the results were promising. Based on the results, we proposed further modifications to the Cooperation and tested the improved algorithm on all thirty CEC 2014 functions [7] at two levels of dimension. In this section, we describe the principles of its working and present new adjustments. In the following sections, we evaluate the performance of the new proposed Cooperation in relation to the testing results.

Let us focus on the principles of the Cooperation algorithm first. We run one of the original algorithms (GWO or jSO) for a limited time, let it solve the problem for some time, and pass its result as input to the second algorithm (which has not yet run), which we also run for a limited time to solve the problem for another piece of time.

This simple idea allows us to use all advantages of both algorithms, but in fact it is a bit more complicated. Both algorithms work with different population sizes (number of agents in the case of GWO and number of points in population in the case of jSO). To simplify the following text we will use a single term - a "point" for both algorithms, and in GWO, we can imagine by the term "point" the position where the agent is located.

In GWO, we use the usual number of points (agents) - which is 6. This number remains constant during the whole calculation of the algorithm. While in the jSO the number of points is dynamically changed during the optimization, the population size decreases linearly.

First, we describe the switching of the GWO algorithm to jSO. The output points of the GWO algorithm are read into the jSO algorithm and they replace the random points of the jSO algorithm except for the top three, which are labeled as non-rewritable and are kept as representatives of the current best solution. Now, we describe the switching of the jSO algorithm to GWO. The GWO algorithm reads the 6 best points (generally the number corresponding to the size of its population, n) from the jSO output and replaces all existing points with them. If the output of jSO (the population is continuously decreasing) contains fewer points than need to be replaced in GWO when the algorithm is switched (i.e. the jSO population is smaller than the GWO population), then only the maximum possible number of points from jSO is replaced in the GWO population. Which points are replaced in the GWO population is chosen randomly, however the best point in the GWO (Alpha agent position) is always kept.

The next issue is how to properly determine after what time the algorithms should switch to another one. In previous research, we changed algorithms always after a static number of evaluations. Each algorithm participated equally in the solution. That is what we improved, and for this research, we decided to choose a new strategy of switching between both algorithms after a constant number of generations/iterations when the algorithm is already stagnant and no further improvements are achieved. Specifically, we test the algorithms to switch after 30, 60, 90, and 120 iterations (for GWO) or generations (for jSO) after it starts to stagnate. This allows us to find the optimal running time of the algorithms and outline a way for further research, in which we try to change the running time dynamically after stagnation.

When the algorithm stagnates, before passing the population to the second algorithm, we bring the configuration to a state just before stagnation. The positions of the agents (for GWO) or points (for jSO) at this moment are passed to the second algorithm as input. This improvement ensures that the second algorithm starts from a better starting position, reducing the probability of rapid stagnation immediately after execution.

Which algorithm starts the computation first is randomly selected. Each of the algorithms has specific advantages, and by randomly selecting the first one to run, we ensure independence of their characteristics.

The flow of the Cooperation algorithm is shown in Figure 1 and proceeds as follows: first, an initial population of points (or agent positions) is randomly generated for the first algorithm, which is randomly selected from GWO and jSO. Then, the following steps are performed in a cycle until the maximum number of evaluations is reached:

1. If the GWO algorithm is selected, let it run until there is no further improvement in the results and thus the algorithm stagnates for more than l iterations, where l in our case is set to 30, 60, 90, or 120.

   Once the GWO run is interrupted due to stagna-

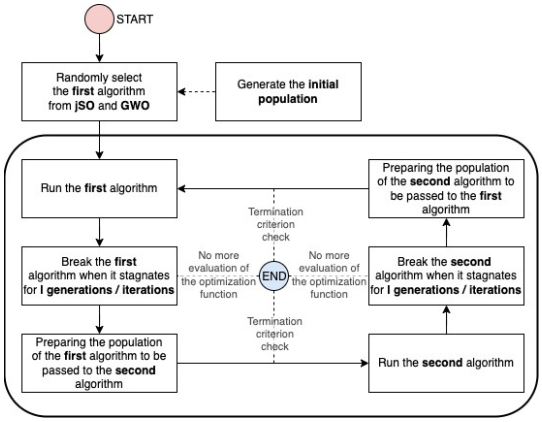

Figure 1: Flow chart showing how the GWO and jSO algorithms switch during Cooperation.

tion, bring the configuration to a state just before stagnation. The n random points in the jSO input are overwritten by the n points from the GWO, where n is the population size of the GWO.
However, the best 3 points in jSO population cannot be overwritten.

2. If the jSO algorithm is selected, let it run until there is no further improvement in the results and thus the algorithm stagnates for more than l generations, where l in our case is 30, 60, 90, or 120.
Once the jSO run is interrupted due to stagnation, bring the configuration to a state just before stagnation.
The n best jSO points (n is the population size of the GWO) replace each of the original GWO points and are used as its input.

4. Test results

The CEC2014 competition introduced benchmarking of optimization algorithms on a set of 30 problems with real function. They provide us an opportunity to test the effectiveness of our optimization algorithms on a set of problems with real function [7].
In this section, we discuss the results of the presented Cooperation algorithm, which was tested on the CEC2014 function set, and its performance was compared with the original GWO and jSO algorithms. We will examine how these two algorithms solved the optimization problems and compare their results to evaluate the efficiency of the new algorithm. The results provide insights into the strengths and weaknesses of these optimization algorithms in direct comparison.
The CEC2014 functions represent the search space for the used optimization algorithms (jSO, GWO and Cooperation) and are defined in multiple dimensions, where the number of dimensions D is chosen by us. We tested a total of 6 algorithms: GWO, jSO, and Cooperation with the stagnation parameter set to 30, 60, 90, and 120. Testing was performed at two levels of dimension, $D = 10$ and $D = 30$, and each configuration was run 15 times for each of the 30 test functions.
In this section, we focus on the results of testing the Cooperation algorithm with different settings of the stagnation parameter l ($l = 30$, $l = 60$, $l = 90$, and $l = 120$), compare it with the original algorithms, and derive conclusions.

4.1. Test results in dimension $D = 10$

The results of testing the algorithms in dimension $D = 10$ on all CEC2014 problems (functions) are shown in Table 1. In the columns GWO and jSO, we can see the median results of 15 runs of the original algorithms with these names. The best among these two medians is shown in bold. The fact that jSO gives better results in the medians for all 30 functions is interesting. In the first iterations of the computation, the GWO results seem promising, but it converges too early and if we let both algorithms run long enough (in our case, the total amount of allowed function iteration (in GWO) or evaluations (in jSO) is $10000 \times D$ for each dimension D), jSO finds a more accurate result (closer to the optimum) in the median of 15 runs.
The following columns show the results of the Cooperation algorithm with different settings of the stagnation parameter l, which is given in corresponding column. For example, *Coop 30* means that a Cooperation algorithm with stagnation parameter $l = 30$ was used, and similarly for the other columns of the table. Each value recorded in that table is the median of 15 results of 15 runs of respective algorithm (or setting of Cooperation algorithm) for specific problem (function). Value of the median which is better than values of the median of both original algorithms GWO and jSO is highlighted in bold.
From the results, it can be observed that as the stagnation parameter l increases, the number of cases where the Cooperation algorithm brings improvement also increases, initially significantly, but only up to a certain value, specifically $l = 90$. Increasing it to $l = 120$ no longer brings improvement. In fact, there is a little degradation. This demonstrates that stagnation is a significant parameter for achieving improvement and should be chosen carefully. Experiments involving dynamic adjustment of the stagnation parameter have even greater potential for achieving better results than trying to find the optimal stagnation setting. We plan to focus on this aspect in the future.
The most interesting fact, as shown by the results of this

experiment, is that even though jSO outperforms GWO in all cases, their Cooperation brings a noticeable improvement. The GWO algorithm supports faster convergence, while jSO contributes to finding a more accurate optimum.

4.2. Test results in dimension $D = 30$

The results of testing the algorithms in dimension $D = 30$ on all CEC2014 functions are shown in Table 2. The structure of the table is the same as described in the section for summarizing the results in dimension $D = 10$. And again, the medians are displayed and compared.
In contrast to the $D = 10$ dimension, GWO yields better results in the $D = 30$ dimension. Out of a total of 30 functions, GWO outperforms jSO in 3 cases (median of results is lower). While this is still not a significant number, it suggests that in higher dimensions ($D = 50$, $D = 100$, etc.), GWO may still be more beneficial. We plan to verify this hypothesis in future research.
The results obtained in the $D = 30$ dimension provide further confirmation that increasing the stagnation parameter leads to significant improvements in Cooperation, particularly in the initial stages (improvement between $l = 30$ and $l = 60$ is significant), however, in $l = 90$ and $l = 120$, the improvements become either marginal or non-existent. Compared to the results in $D = 10$, there is not as much improvement in $D = 30$, but the general trend of improvement is the same.

4.3. Summary of results in dimensions $D = 10$ and $D = 30$

In Table 3, we can see a summary of the results of the research.
In the first table section, we can observe the outcomes of the original GWO and jSO algorithms, indicating the number of functions for which they provided better results. We can compare the results in both $D = 10$ and $D = 30$ and the better value is highlighted in bold.
The subsequent table section displays the frequencies at which the Cooperation algorithm, with varying stagnation settings, outperformed the original algorithms. Again, we can compare the results in both $D = 10$ and $D = 30$ and the better value is highlighted in bold.
The next table section displays the frequencies at which the Cooperation algorithm, with varying stagnation settings, provided worse results than the better of the two original algorithms.
And the last table section illustrates the instances in which the Cooperation algorithm, with various stagnation settings, yielded the same result as the better of the two original algorithms.
Let us note that at stagnation parameter $l = 30$, not only the Cooperation algorithm does fail to bring significant improvement, but in most cases, the results are even worse compared to the original algorithms in both tested dimensions, $D = 10$ and $D = 30$.
When setting the stagnation parameter to $l = 60$, the situation noticeably improves, but the results are still worse for more than half of the tested functions in both dimensions. In contrast, at stagnation parameter $l = 90$, the results become quite satisfactory, with more than one-third of the cases outperforming the original functions, and less than one-third of the cases yielding worse results. The parameter $l = 120$ does not bring any further significant improvement, but the results do not differ much from $l = 90$ and are still more than satisfactory.
Recall that all compared values are medians of 15 algorithm runs. As we can see, the stagnation parameter l significantly affects the performance of the Cooperation algorithm, and the best setting is somewhere between $l = 90$ and $l = 120$. This is the same in both tested dimensions $D = 10$ and $D = 30$.
We believe that the reasons for this behavior are as follows. When the stagnation parameter l is set low ($l = 30$), the algorithms switch too early. The algorithm switch occurs before the algorithm can search a sufficient number of points and "focus" on the area of search space closest to the current optimum. In contrast, when the stagnation parameter l is set high ($l = 120$), the algorithm runs too long when it is no longer improving, wasting time for the next algorithm that could have already been run.
In future, we expect to achieve even better results by using an adaptive parameter dependent on the computation runtime separately for GWO and jSO. This is how we plan to get the most efficiency out of both algorithms.

5. Conclusion

In this paper, we investigated an algorithm we developed, named Cooperation, which combines two well-established and successful optimization algorithms, GWO and jSO. Each of these algorithms is executed for a period of time, determined by a stagnation state, where no further improvement is observed. We tested different settings of the stagnation parameter and identified its significant impact on the overall performance of the new algorithm, independent of the problem dimension. The results are promising because we found a suitable setting for the stagnation parameter. However, our future plans include achieving even better results by further adjusting not only the stagnation parameter.

References

[1] Poláková, R., Valenta, D.: jSO and GWO Algorithms Optimize Together. In: *Conference Information Tech-*

Table 1
Median results of tested algorithms, dimension $D = 10$

Function	jSO	GWO	Coop 30	Coop 60	Coop 90	Coop 120
1	**0.00E+00**	5.16E+07	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	**0.00E+00**	2.71E+09	0.00E+00	0.00E+00	0.00E+00	0.00E+00
3	**0.00E+00**	1.79E+04	0.00E+00	0.00E+00	0.00E+00	0.00E+00
4	**3.48E+01**	4.80E+02	3.48E+01	3.48E+01	3.48E+01	3.48E+01
5	**2.00E+01**	2.06E+01	**2.00E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**
6	**0.00E+00**	8.30E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
7	**0.00E+00**	7.59E+01	9.86E-03	0.00E+00	0.00E+00	0.00E+00
8	**0.00E+00**	6.37E+01	2.98E+00	0.00E+00	0.00E+00	0.00E+00
9	**1.99E+00**	5.86E+01	6.96E+00	2.98E+00	1.99E+00	2.98E+00
10	**0.00E+00**	1.24E+03	1.86E+01	6.25E-02	0.00E+00	0.00E+00
11	**1.53E+01**	1.73E+03	3.67E+02	2.19E+01	**1.52E+01**	**1.52E+01**
12	**9.18E-02**	1.48E+00	**5.83E-02**	**7.59E-02**	**5.01E-02**	**5.40E-02**
13	**5.42E-02**	8.79E-01	8.31E-02	6.92E-02	7.12E-02	5.44E-02
14	**5.03E-02**	5.73E+00	7.27E-02	7.37E-02	6.69E-02	5.51E-02
15	**3.90E-01**	1.25E+01	7.08E-01	5.21E-01	4.66E-01	4.03E-01
16	**9.68E-01**	4.28E+00	1.92E+00	1.29E+00	1.16E+00	**9.16E-01**
17	**1.42E+00**	9.55E+05	8.17E+00	1.64E+00	**1.20E+00**	**1.41E+00**
18	**4.95E-02**	1.40E+04	4.04E-01	2.16E-01	1.09E-01	6.70E-02
19	**3.90E-02**	1.40E+01	4.57E-01	8.42E-02	**3.65E-02**	4.89E-02
20	**7.06E-02**	1.17E+04	6.02E-01	2.13E-01	**4.75E-02**	**5.48E-02**
21	**5.86E-02**	3.56E+05	3.63E-01	6.35E-01	3.40E-01	3.22E-01
22	**4.34E-01**	2.62E+02	9.89E-01	**3.27E-01**	**1.73E-01**	**1.99E-01**
23	**3.29E+02**	3.52E+02	3.29E+02	3.29E+02	3.29E+02	3.29E+02
24	**1.08E+02**	2.01E+02	1.11E+02	1.09E+02	**1.08E+02**	1.09E+02
25	**1.16E+02**	2.00E+02	1.20E+02	1.17E+02	**1.14E+02**	**1.12E+02**
26	**1.00E+02**	1.87E+02	1.00E+02	1.00E+02	**1.00E+02**	1.00E+02
27	**1.47E+00**	6.16E+02	2.06E+00	1.89E+00	**1.44E+00**	**1.43E+00**
28	**3.72E+02**	1.22E+03	3.72E+02	3.72E+02	3.72E+02	3.72E+02
29	**2.22E+02**	4.55E+07	2.22E+02	2.22E+02	2.22E+02	2.22E+02
30	**4.62E+02**	3.46E+04	4.63E+02	4.62E+02	4.62E+02	**4.62E+02**

nologies - Applications and Theory. Slovakia (2022).

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

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

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

[5] 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

[6] 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

[7] Special Session & Competition on Real-Parameter Single Objective (Expensive) Optimization at CEC-2014, Beijing, China.

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

Table 2
Median results of tested algorithms, dimension $D = 30$

Function	jSO	GWO	Coop 30	Coop 60	Coop 90	Coop 120
1	**0.00E+00**	7.51E+08	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	**0.00E+00**	4.64E+10	0.00E+00	0.00E+00	0.00E+00	0.00E+00
3	**0.00E+00**	8.42E+04	0.00E+00	0.00E+00	0.00E+00	0.00E+00
4	**0.00E+00**	5.31E+03	0.00E+00	0.00E+00	0.00E+00	0.00E+00
5	**2.02E+01**	2.11E+01	**2.01E+01**	**2.00E+01**	**2.01E+01**	**2.01E+01**
6	**0.00E+00**	3.57E+01	0.00E+00	0.00E+00	0.00E+00	0.00E+00
7	**0.00E+00**	4.30E+02	0.00E+00	0.00E+00	0.00E+00	0.00E+00
8	**0.00E+00**	3.22E+02	2.59E+01	0.00E+00	0.00E+00	0.00E+00
9	**1.10E+01**	3.16E+02	2.39E+01	1.59E+01	1.29E+01	1.20E+01
10	**1.17E+00**	6.83E+03	1.90E+03	**1.46E-01**	**2.15E-01**	**3.01E-01**
11	**1.12E+03**	7.23E+03	3.06E+03	1.86E+03	1.24E+03	1.18E+03
12	**1.83E-01**	2.52E+00	2.88E-01	**1.43E-01**	**1.34E-01**	**1.60E-01**
13	**1.51E-01**	6.29E+00	2.23E-01	1.87E-01	1.82E-01	1.56E-01
14	**1.34E-01**	1.75E+02	2.05E-01	1.88E-01	1.62E-01	1.61E-01
15	**2.35E+00**	5.47E+04	3.06E+00	**2.12E+00**	**2.29E+00**	**2.25E+00**
16	**8.26E+00**	1.39E+01	1.15E+01	9.41E+00	8.41E+00	8.52E+00
17	**6.19E+01**	7.14E+07	1.23E+02	7.31E+01	**5.92E+01**	**5.93E+01**
18	**1.50E+00**	2.47E+08	5.35E+00	5.23E+00	3.38E+00	2.44E+00
19	**2.61E+00**	2.96E+02	2.72E+00	**2.45E+00**	2.88E+00	2.79E+00
20	**2.45E+00**	2.55E+05	6.96E+00	2.79E+00	2.73E+00	2.64E+00
21	**2.43E+01**	2.63E+07	1.46E+02	**1.76E+01**	**1.71E+01**	**2.20E+01**
22	**2.39E+01**	2.99E+03	1.40E+02	1.45E+02	2.66E+01	**2.25E+01**
23	**3.15E+02**	4.61E+02	3.15E+02	3.15E+02	3.15E+02	3.15E+02
24	**2.00E+02**	2.00E+02	2.00E+02	**2.00E+02**	**2.00E+02**	**2.00E+02**
25	2.03E+02	**2.00E+02**	2.00E+02	2.03E+02	2.03E+02	2.00E+02
26	**1.00E+02**	2.00E+02	1.00E+02	1.00E+02	1.00E+02	1.00E+02
27	3.00E+02	**2.00E+02**	3.00E+02	3.00E+02	3.00E+02	3.00E+02
28	8.38E+02	**2.00E+02**	8.43E+02	8.42E+02	8.38E+02	8.38E+02
29	**7.15E+02**	4.69E+04	7.15E+02	7.16E+02	7.16E+02	7.16E+02
30	**7.15E+02**	3.84E+06	1.14E+03	1.08E+03	1.83E+03	1.44E+03

Table 3
Summary of our experiments

Median comparison:	**D=10**	**D=30**
Number of jSO wins	**30**	27
Number of GWO wins	0	**3**
Number of times *Coop 30* is better than the winner	**2**	1
Number of times *Coop 60* is better than the winner	3	**7**
Number of times *Coop 90* is better than the winner	**11**	7
Number of times *Coop 120* is better than the winner	**10**	8
Number of times *Coop 30* is worse than the winner	21	21
Number of times *Coop 60* is worse than the winner	18	**15**
Number of times *Coop 90* is worse than the winner	**7**	15
Number of times *Coop 120* is worse than the winner	**9**	13
Number of times *Coop 30* is same as the winner	7	8
Number of times *Coop 60* is same as the winner	9	8
Number of times *Coop 90* is same as the winner	12	8
Number of times *Coop 120* is same as the winner	11	9