

Rain-Fall Optimization Algorithm with new parallel implementations

Juan Manuel Guerrero-Valadez, Felix Martínez-Rios*

Universidad Panamericana, Facultad de Ingeniería, Augusto Rodin 498, Ciudad de México, 03920, México

Abstract

Rainfall Optimization Algorithm (RFO) is a nature-inspired metaheuristic optimization algorithm. RFO mimics the movement of water drops generated during rainfall to optimize a function. The paper study new implementations for RFO to offer more reliable results. Moreover, it studies three restarting techniques that can be applied to the algorithm with multithreading. The different implementations for the RFO are benchmarked to test and verify the performance and accuracy of the solutions. The paper presents and compares the results using several multidimensional testing functions, as well as the visual behavior of the raindrops inside the benchmark functions. The results confirm that the movement of the artificial drops corresponds to the natural behavior of raindrops. The results also show the effectiveness of this behavior to minimize an optimization function and the advantages of parallel computing restarting techniques to improve the quality of the solutions.

Received on 29 February 2020; accepted on 06 April 2020; published on 15 April 2020

Keywords: Optimization, Metaheuristics, Rainfall Optimization Algorithm, Multithreading, Simulated Annealing, Genetic Algorithm, Nature-inspired

Copyright © 2020 Juan Manuel Guerrero-Valadez *et al.*, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi:10.4108/eai.13-7-2018.163981

1. Introduction

The optimal solution for a specific function can be obtained with different methods. In the last decades, people started to research new methods to find global maximums and minimums without using traditional algorithms. Researchers have found that heuristic-based algorithms could find the global optimum for highly complex functions in less time than the traditional way. And in the real world, the efficiency and precision of the algorithm are crucial. Moreover, these algorithms are needed in many different fields, like physics, engineering, computer science, industrial processes, demography, and economics. Some application models are represented with functions that could take a lot of time to find their global optimums with the traditional methods, and that is why researches started to develop new optimization algorithms.

A vast number of approaches have been suggested. But not all the algorithms are as precise as they

are required to be in a real-world problem. Some algorithms fail to find the optimum solution in specific functions. To attend those problems, researches have to design meta-heuristic optimization algorithms. Meta-heuristic algorithms are problem-independent algorithms that can be applied to solve an optimization problem. A well-written meta-heuristic optimization algorithm will adapt to a given specific problem without a lot of modifications. [1]. The research and proposal presented in this paper are based on a nature-inspired optimization algorithm named as Rainfall Optimization Algorithm (RFO). Nature-inspired algorithm terminology is used for all the algorithms that were inspired on a biological, physical or chemical process that can be found in nature. Nature-inspired algorithms used to solve optimization problems have been very successful in the last decades [2] and have allowed researchers to have a large number of sources of inspiration [3].

RFO was written by *Kaboli, Sevbaraj* and *Rahim* and it was inspired by the behavior of raindrops. The algorithm performs fast in multi-variable functions, but it sometimes fails to find the global optimum in some

*Juan Manuel Guerrero-Valadez. Email: juanmanuel.guerrerovaladez@up.edu.mx

functions. It might get results that are not as precise as other optimization algorithms because it can run out of iterations, or when raindrops start moving in the wrong direction or get stuck on local optima. The main goal of this paper is to introduce new implementations that can help RFO to make it more precise. The proposed implementations exploit the advantages of multi-threaded processors found in modern computers. The RFO of this paper can perform parallel executions of the algorithm with different restarting techniques: restart to the best, genetic restart to the best and simulated annealing. Restart to the best is a method that can help the quality of a solution of an optimization problem by getting more precise results with a decent computation time by the use of parallel threads. The other techniques are an improvement of the restart to the best, because they can make RFO more meta-heuristic based, and as a result, RFO will perform better in more functions.

If multi-threaded computers are used correctly, they can offer high computational power which makes them useful for different science and engineering problems. The new approach discussed in this paper presents the advantages of the implementation of parallelization in RFO. These implementations intend to use the benefits of parallel computing by running the algorithm in different parallel processes at the same time with fewer iterations on the threads. The results obtained by each thread can be compared to obtain the global optimum or continue the process with the restarting techniques previously mentioned to find better quality results.

The rest of the paper is structured as follows. First, the description and flowchart of the RFO are presented. In section 3, the parallel implementations with the proposed restart techniques are described. Next, the performance of the different implementations of the algorithm is evaluated by the use of benchmark functions. Finally, the conclusions are outlined in the last section.

2. Rain-Fall Optimization Algorithm

The RFO is inspired by the behavior of drops during a rainfall. In nature, raindrops drip down from a peak to form streams and rivers that reach the sea or lakes [1]. This process can be implemented in optimization algorithms because the behavior of the raindrops is similar to an exploration process that can be used to find the minimums of a mathematical function. Raindrops will keep falling until they reach a place where they cannot continue to fall. In nature, they can get stuck before reaching the sea level (the global minimum), such as ponds or lakes that can be translated to an optimization problem as local minimums. Also, RFO simulates the tendency of drops to move towards the steepest slopes [1].

In the first iteration of the algorithm, a population of raindrops is generated in random positions of the optimization function, which can be associated with the geographical terrain. This first process represents the simulation of the new raindrops produced by rainfalls. Next, it is necessary to simulate the natural movement of raindrops. After being generated, the artificial drops must move to the steepest slope of the radius that surrounds them. Their next position can be determined using several methods. For a similar process in other optimization algorithms, gradient descent is used. But in RFO, a method called random search is used. For this method, the algorithm has to generate neighbor points around every artificial drop. The area where the neighbor points are generated can be called the neighborhood of the drop. After generating the neighbor points of a drop, the algorithm evaluates them according to the optimization function. The result of each neighbor point is compared with the previous position of the drop to decide which point corresponds to the lowest position of the neighborhood. For each drop, this process will continue to execute until the drop reaches the lowest point of the terrain (the global optimum) or gets stuck in a puddle (local optima).

2.1. Description of the algorithm

The flowchart in Figure 1 describes the main algorithm of RFO. It is based on the original algorithm written by Kaboli, Selvaraj, and Rahim, but with some improvements. To understand the algorithm, it is necessary to explain some concepts.

Raindrop: A rainfall generates a population of artificial drops that contain the following attributes:

- \mathbf{X}_k It refers to the position of the raindrop at the k^{th} dimension of the optimization problem.
- **Status:** It is a flag that marks the drop as active or inactive. When the artificial drop is inactive, it means that the drop is stuck or far from the global optimum. Hence, the raindrop stops moving.
- **Value:** It represents the fitness of the solution by evaluating the position of the drop at iteration i .
- **Rank:** It is an attribute that determines the position of the raindrop at the merit-order list at iteration i , and it is calculated using the following equation:

$$D_{rank}^i = \begin{cases} C1_t^i = f(D_t^i) - f(D_1^i) \\ C1_t^i = f(D^i) \\ Rank_t^i = w_1 \times order(C1_t^i) + \\ w_2 \times order(C2_t^i) \end{cases} \quad (1)$$

Where:

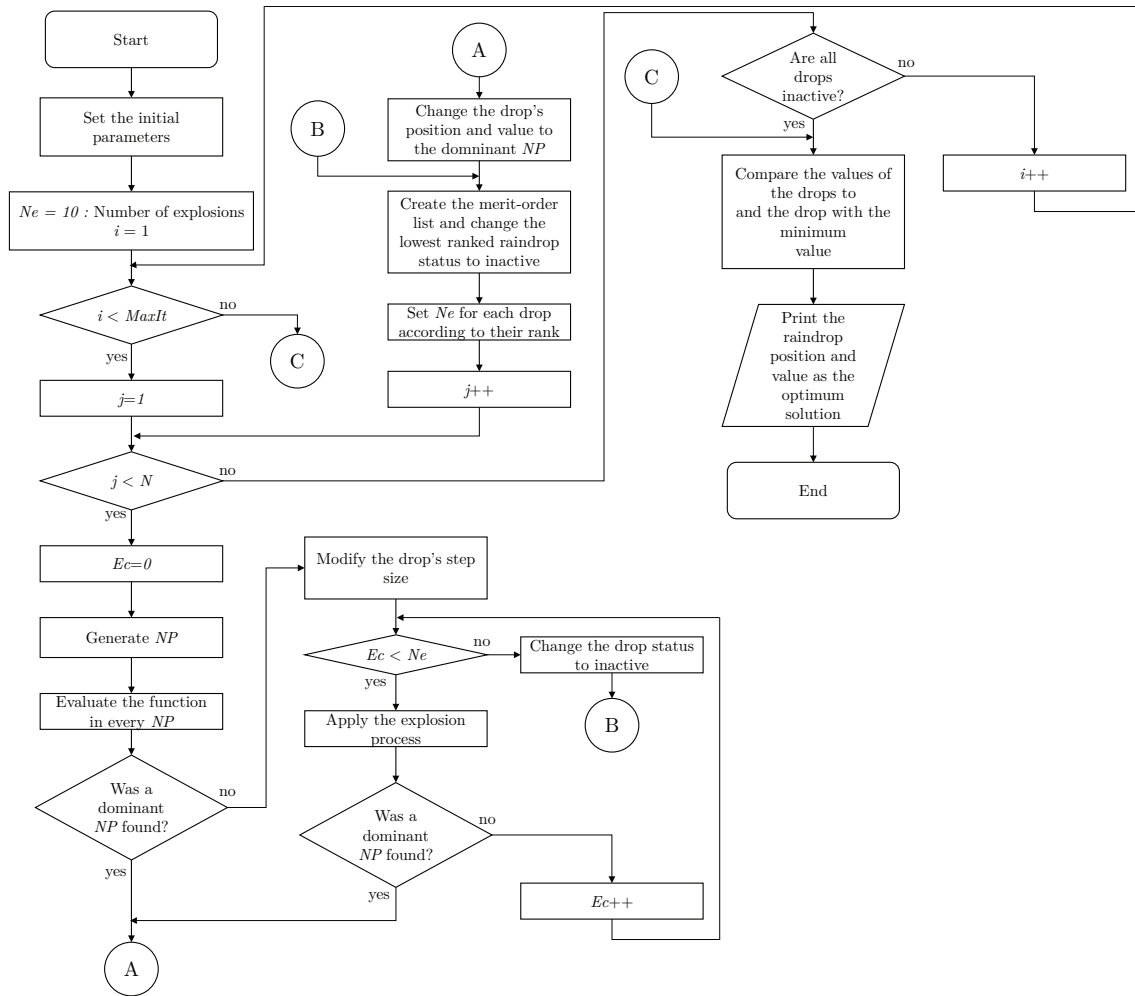


Figure 1. Flowchart of the RFO algorithm.

- $C1_t^i$ is the rate of change of value at iteration t for raindrop D^i with respect to its initial value.
- $C2_t^i$ is the current value for raindrop D^i
- **Order** $C1_t^i$ and **Order** $C2_t^i$ are sorted in ascending order.
- w_1 and w_2 are the weighting coefficients that have a constant value of 0.5 [1].
- **Rank** $_t^i$ returns the rank for raindrop D^i at iteration t .

- **Step size:** It is necessary to use random search to define the next position of the drop. The step size is the property that defines the area of the neighborhood in which neighbor points can be generated. To simulate the descent of a drop from a peak, the neighborhood size must change according to the speed of the drop at iteration i . If the delta between the fitness of the last iterations is greater, it means that it is moving

to a steeper area. And the speed of the drops is directly proportional to the slope of the terrain. To simulate that behavior in the artificial raindrops, Equation 2 was designed.

$$D_{step}^i = \begin{cases} step_0 & \text{for } i == 0 \\ step_{i-1} \times (1 + R_s) & \text{for } Ec_{i-1} == 0 \\ \frac{step_{i-1}}{1+R_s} & \text{for } Ec_{i-1} > 0 \end{cases} \quad (2)$$

Where:

- $step_0$ is the initial step size given as a parameter for the solution.
- R_s is a random number between 0 and 1.
- $step_{i-1}$ is the step size at the previous iteration.
- Ec_{i-1} is the explosion counter at the previous iteration.

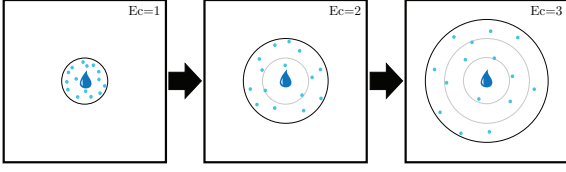


Figure 2. Visual representation of the explosion process.

Neighborhood: As mentioned before, the neighborhood is the space where the neighbor points of the N^{th} raindrop can be evaluated. According to Equation 3 [1], the size of the neighborhood is wider when the drop is moving faster. This allows the drop to do bigger jumps in the exploration phase and move slower in the exploitation phase to produce high-quality solutions [4].

$$N_i^k = D_i^k + (N_0^k \times D_{step}^i) \quad (3)$$

Where:

- N_0^k is the initial neighborhood size of all the drops at k^{th} dimension according to Equation 4.
- D_i^k is the position of the artificial raindrop at k^{th} dimension in the iteration i .
- D_{step}^i is the step size at of the drop at the current iteration.

$$N_0^k = (|up_k| + |low_k|) \times 0.02 \quad (4)$$

Where:

- up_k is the upper limit of the search space at k^{th} dimension.
- low_k is the lower limit of the search space at k^{th} dimension.

Neighbor points: These new points that spawn in a random position inside the neighborhood of a drop represent the possible positions of the drop at iteration $i + 1$. Each neighbor point generated at iteration i must be evaluated to determine the next dominant position of the drop. The position of the neighbor point NP_j^i of raindrop D_i is generated using the following equation[1]:

$$\begin{aligned} \|(D_i^k - NP_j^i) \cdot \vec{u}_k\| &\leq \|N_i^k \cdot \vec{u}_k\| \\ 1 \leq i \leq m \\ 1 \leq j \leq np \\ 1 \leq k \leq n \end{aligned} \quad (5)$$

Explosion process: It is carried out when an artificial drop does not have a dominant neighbor point. This situation occurs when the raindrop is stuck in a local minimum or when an insufficient amount of neighbor points are generated [1]. The first suggestion to release

a raindrop from this situation is to generate more neighbor points. But in practice, this technique was not helping in all situations. To solve that, a new way to generate the neighbor points can be implemented during the explosion process. The new method is inspired by the expansion of a blast, as shown in Figure 2. To simulate this behavior, Equation 6 can be used. In consequence, the positions of the neighbor points generated during the explosion process can be distributed evenly from the center of the neighborhood (where the raindrop is stuck) to its edges.

$$D_i^k + [(|up_k| + |low_k|) \times E_{base}] \quad (6)$$

Where E_{base} is equal to the next equation:

$$E_{base} = \begin{cases} (0.9)(10^{-exp+a})(Ec^{exp_{base}}) & a \geq 1 \\ (0.9)(10^{-exp_{base}+1})(Ec^{exp_{base}}) & a < 1 \end{cases} \quad (7)$$

Where:

- a : can be defined by Equation 8.
- Ec : is the number of explosion processes that have been carried out in the current drop.

$$a = -\log_{10}\left(\frac{|up_k + low_k| \cdot \frac{3}{200}}{0.9}\right) \quad (8)$$

Merit-order list: It is a list that sorts in ascending order the raindrops at iteration i according to their rank. The positional index of a raindrop at the merit-order list can determine its status. For better performance of the algorithm, the drop with the worst rank will change its status to inactive. A raindrop can also become inactive if the explosion process is carried out and fails to find a dominant point. And the drops with higher ranks can iterate more during the explosion processes before getting inactive.

3. Parallel Implementations for the Rain-Fall Optimization Algorithm

The paper proposes parallel computing to obtain better quality and better performance of the algorithm. Instead of initializing one rainfall, multi-threading gives the ability to generate multiple rainfalls with an independent population of raindrops. But the objective of the parallel implementation is to share information between threads and use it to enhance the route of the raindrops towards the global minimum. As a result, the algorithm needs less amount of iterations to find a suitable or high-quality optimum value with the same or lower CPU time.

With parallel implementations, a new iteration counter called H must be declared. In Figure 3, the H counter indicates the number of cuts in which the

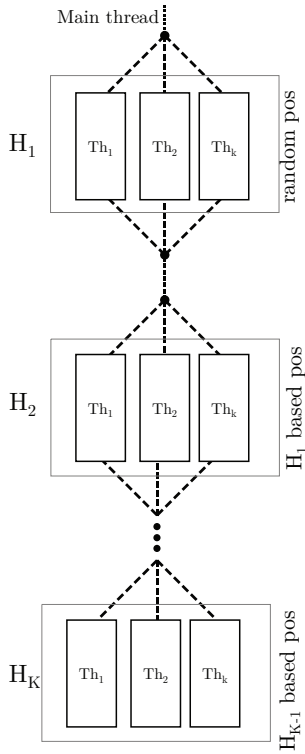


Figure 3. The threading process.

algorithm joins all threads to share information about the artificial drops and their fitness value. At every cut, new rainfalls can be generated in each thread. In the first iteration of RFO (H_1), all raindrops of all threads are generated and positioned randomly inside the search space and according to the original algorithm described in Figure 1. But if H is greater than 1, the positions of the drops will be based on the results that H_{K-1} threads obtained. Furthermore, the way the new drops are generated may vary depending on the restarting technique chosen. The paper suggests three ways to handle the cuts: restart to the best, genetic restart to the best and annealing restart to the best.

3.1. Restart to the best

Restart to the best is a technique that is easy to implement. The algorithm compares among all the best-positioned drops, according to their fitness, at H_{K-1} and picks the best drop as the starting position, as it is shown in Algorithm 1. In other words, after selecting the best drop, instead of generating new random positions for the H_K iteration, all raindrops of all threads can take advantage of the results of H_{K-1} and start at the position of the best drop of all threads of the last H iteration.

In practice, this method can be effective to obtain more reliable results than the ones from the original RFO. But restart to the best technique is not the best

Algorithm 1: Pseudocode to obtain the best drop of H iteration

```

1 Start;
2 Define  $i_{cuts}$  (Iterations before thread cut) ;
3  $H_i = 1$ ;
4  $i = i_{cuts}$ ;
5 Run RFO from Figure 1 with local
    $MaxItera = i_{cuts}$ ;
6 while  $i \leq MaxItera$  do
7   Get the best drops of each thread of  $H_i$  and
   save them inside  $savedDrops$  array;
8   Sort the drops of the  $savedDrops$  array in
   ascending order according to their fitness ;
9    $D_{best} = savedDrops[0]$ ;
10  Run RFO from Figure 1 with local
    $MaxItera = i_{cuts}$  but with  $D_{best}$  position as
   starting position;
11   $i = H_i \times i_{cuts}$ ;
12   $H_i ++$ ;
13 end
14 Sort the drops of  $savedDrops$  in ascending order
   according to their fitness;
15 Print the first drop of  $savedDrops$  as the
   optimum solution ;

```

approach. If the algorithm follows the metaheuristic proposal, the algorithm must be adaptive for all optimization problems. And restart to the best could behave as a glutton algorithm because it always chooses the drop that seems to be the best candidate based on its fitness [5]. But in some circumstances or some optimization functions, a drop with a worse fitness at iteration H might also be closer to the global optimum of the function than the one with the best fitness value. As a result, the algorithm might do regression in the process in some H iterations.

3.2. Genetic restart to the best

Another restarting technique that can be implemented is a genetic restart. The method is a hybridization of the genetic algorithm (GA) and the restart to the best technique that was previously mentioned. This approach was nature inspired by the behavior of two different genes that form a new and better gene. The algorithm uses the crossover of two artificial genes that define the fitness of a raindrop [6, 7]. Before every restart, Algorithm ?? is performed. It mixes the positions of some random dimensions of the best drops of each thread into a new artificial drop with better genetics (Figure 4). The quality of the gene is determined by its fitness value inside the optimization function.

Algorithm 2: Genetic Algorithm Pseudocode

```

1 Start;
2 Define  $i_{cuts}$  (Iterations before thread cut) ;
3  $H_i = 1$ ;
4  $i = i_{cuts}$ ;
5 Run RFO from Figure 1 with local
    $MaxItera = i_{cuts}$ ;
6 while  $i \leq MaxItera$  do
7   Get the best drops of each thread of  $H_i$  and
   save them inside  $savedDrops$  array;
8   Sort the drops of  $savedDrops$  in ascending
   order according to their fitness ;
9    $D_{best} = savedDrops[0]$ ;
10   $rnd_{it} = RANDOM$  ;
11  for  $H < rnd_{it}$  do
12     $rnd_k = RANDOM$ ;
13     $rnd_D = RANDOM$ 
14    ;
15    Replace  $D_{best}$  position at  $rnd_k$  with the
    position at the same  $rnd_k$  of the drop
     $rnd_D$ ;
16    Evaluate the  $D_{best}$ 
17    ;
18    if  $fitness(new\ D_{best}) > fitness(old\ D_{best})$ 
    then
19       $D_{best} = old\ D_{best}$ ;
20    end
21    Run the RFO from Algorithm ??, but
    setting all drops to start at the best drop's
    positions;
22  end
23   $i = H_i \times i_{cuts}$ ;
24   $H_i ++$ ;
25 end
26 Sort the drops of  $savedDrops$  in ascending order
   according to their fitness;
27 Print the first drop of  $savedDrops$  as the
   optimum solution ;

```

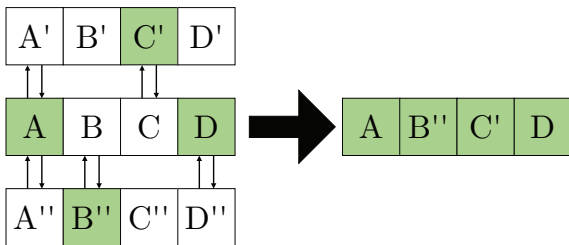


Figure 4. The crossover of different genes.

3.3. Simulated annealing restart to the best

Simulated Annealing algorithm (SA) is a probabilistic technique inspired by the annealing or cooling process

Algorithm 3: Pseudocode to obtain the best drop of H iteration

```

1 Start;
2 Define  $i_{cuts}$  (Iterations before thread cut) ;
3  $H_i = 1$ ;
4  $i = i_{cuts}$ ;
5  $t_{MAX} = \frac{MaxItera}{i_{cuts}}$  Run RFO from Figure 1 with
   local  $MaxItera = i_{cuts}$ ;
6 while  $i \leq MaxItera$  do
7   Get the best drops of each thread of  $H_i$  and
   save them inside  $savedDrops$  array;
8   Sort the drops of the  $savedDrops$  array in
   ascending order according to their fitness ;
9    $D_{best} = savedDrops[0]$ ;
10  for  $j < thread_{count}$  do
11     $t = H_i$ ;
12    Get  $T$  from Equation 9;
13     $R_a = RANDOM$ ;
14    if  $R_a < T$  then
15       $savedDrops[j] = D_{best}$ ;
16    end
17  end
18  Run RFO from Figure 1 with local
    $MaxItera = i_{cuts}$  but with  $savedDrops$ 
   positions as starting position;
19   $i = H_i \times i_{cuts}$ ;
20   $H_i ++$ ;
21 end
22 Sort the drops of  $savedDrops$  in ascending order
   according to their fitness;
23 Print the first drop of  $savedDrops$  as the
   optimum solution ;

```

of metals [8]. In nature, this process occurs when the heat source of molten metal is removed from it. Consequentially, the temperature of the metal starts to decrease. This process will continue to happen until the metal has reached the ambient temperature. At this point, the energy has reached the lowest value and the state of the metal should be fully solid [9, 10].

$$T(t) = e^{\frac{-1}{\ln \frac{t_i}{t_{MAX}}}} \tag{9}$$

The SA algorithm as a restarting technique uses the Equation 9 and a random number R_a between 0 and 1. T represents the temperature of the metal, t_i the time that has elapsed from the beginning of the annealing process, and t_{MAX} the time that has to pass for the metal to become completely solid. In RFO, the elapsed time t_i is equal to the H_K iteration counter, and t_{MAX} can be associated with the total number of H iterations that will be carried out according to the given parameter in the solution. Finally, if the temperature T is greater than the random number R_a , the algorithm will perform the

restart to the best technique. Otherwise, the algorithm should not do anything in the H_K cut. By using the Equation 9 in SA, the scale of the temperature T must be between 0 and 1.

The equation was designed to behave similar to the cooldown temperature of the metal at time t multiplied by a constant K . This process produces a large perturbation in the initial stages of the exploration, ensuring that the positions of the raindrops are well-tuned at the final stages of the optimization [11]. This happens because the probability P to fulfill the condition where $R_a^i < T_i$ is higher at the first H iterations when the temperature of the metal is higher. And the probability P should get lower as the temperature decreases. SA technique can be summarized in Algorithm 3.

4. Experiments

This section shows the performance results of the RFO for continuous optimization problems. The benchmark functions and configurations that were used to obtain the presented results are also disclosed. The experimental results are then analyzed and compared between the different purposed implementations and between other optimization problems.

4.1. Experimental Setup

To evaluate the performance and quality of the results given by the algorithm, it was tested with 5 different benchmark functions. The testing software was written with CSharp and Windows Forms. It can run both single-threaded and multithreaded RFO. The software has a user interface where the different configurations are set up.

Table 1 shows the benchmark functions that were used to test RFO with different configurations. The Rosenbrock function [12], the Ackley function [13], the Sphere function [14], the Griewank function [15] and the Kowalik function [16]. The first four

Table 1. Benchmark functions

Function	Name	Range	Dim	f_{\min}
F1	Ackley	[-30, 30]	30	0
F2	Griewank	[-600, 600]	30	0
F3	Kowalik	[-4, 4]	4	3.075E-04
F4	Rosenbrock	[-30, 30]	30	0
F5	Sphere	[-100, 100]	30	0

Table 2. Experimental configuration parameters

Parameters:	
Neighbor Points	300
Max. Iterations:	3000
Initial Step Size:	0.23
Explosion Base:	3
Iterations before cut:	100

benchmark functions are multidimensional, and the Kowalik function has four dimensions. The 3D shapes of the testing functions can be visualized in Table 5. All four multidimensional functions were optimized with up to 30 dimensions. The only restriction that determined the search space was the suggested range of the benchmark function. All the dimensions had the same range according to the table. The mathematical representations of the benchmark functions are expressed in Table 3. [17].

The suggested parameters were determined after doing some trials in all functions, The configuration for RFO shown in Table 2 should work with any function.

4.2. Experimental results

The search history diagrams that have been drowned in Table 5 were obtained after recording the positions of the artificial raindrops at the first 100 iterations on the 2D version of the benchmark functions. These visual results show that the random distribution that set the initial positions of the drops, and how they move inside the search space towards the coordinates of the global minimum. As expected, the drawings shows how almost all raindrops cluster around the global minimum.

Table 5 also shows the convergence curves for each benchmark function. The x axis represents the iterations and the y axis shows the fitness value. The fitness history of the drop that had the best fitness value of all threads at the end of the algorithm is saved. And the recorded fitness values were plotted with MATLAB after each iteration inside the convergence graphs. The convergence curves of all the testing functions were plotted using the original RFO (single-threaded) and the new algorithm with all the purposed restarting techniques. The complete graphics can help to visually compare the behavior of the different variations of the algorithm. But the main fact that can be analyzed is the speed of the algorithm. The convergence curves showed that RFO is a fast algorithm because it does not need a lot of iterations to optimize near the global optimum value. Because of that, it can be inferred that the parallelization only affects the standard deviation and the quality of the optimization results using the configuration that was used for the experimental analysis of this paper. Having more rainfalls decreases

Table 3. Equations for the benchmark functions

Function	
F1	$F_1(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^n \cos 2\pi x_i) + 20 + e$
F2	$F_2(x) = \frac{1}{4000}\sum_{i=1}^n x_i^2 - \sum_{i=1}^n \cos(\frac{x_i}{\sqrt{i}}) + 1$
F3*	$F_3(x) = \sum_{i=1}^n \left[a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$
F4	$F_4(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2) + (x_i - 1)^2]$
F5	$F_5(x) = \sum_{i=1}^n x_i^2$

* Note: the Kowalik function depends on a_i and b_i arrays as follows [18]:

$$a = [0, 0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246]$$

$$b = [0, 4, 2, 1, 0.5, 0.25, 0.167, 0.125, 0.1, 0.0833, 0.0714, 0.0625]$$

Table 4. Different Optimization Algorithms performance comparison

Function	GA[19]	PSO[19]	GSO[20]	RFO
Sphere	3.1711	3.6927e-37	1.9481e-8	1.4607e-6
Rosenbrock	338.56	37.3582	49.8359	32.4053
Ackley	0.8678	1.3404e-3	2.6548e-5	1.724e-4
Griewank	1.0038	0.2323	3.0792e-2	9.86e-2
Kowalik	7.0878e-3	3.8074e-4	3.7713e-4	3.1553e-4

the possibility of failing to optimize the function and the restarting techniques became relevant in the exploitation processes.

The convergence curves of the figures of Table 5 also confirm that the behavior of the artificial raindrops moving towards the steepest slope is fulfilled. The curves clearly show descending behavior in all five benchmark functions.

The trajectory of the best raindrop in the first dimension was also drawn in Table 5. This diagram records the position of the best drop of all threads of the first variable after every iteration. As expected, the raindrops move uniformly and without abrupt changes towards the optimum value of the variable.

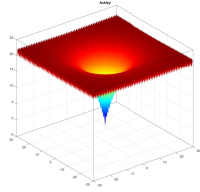
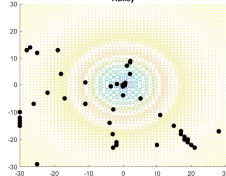
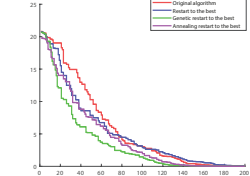
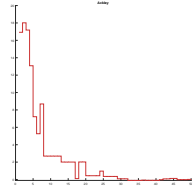
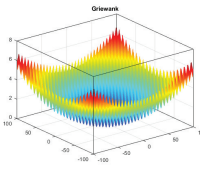
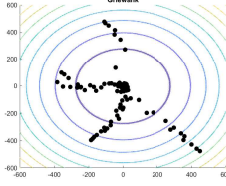
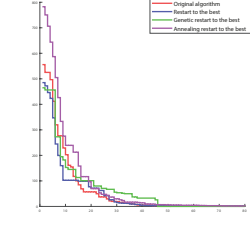
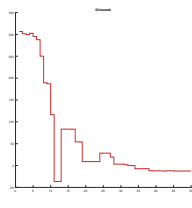
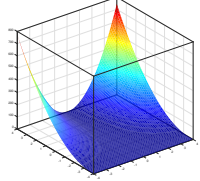
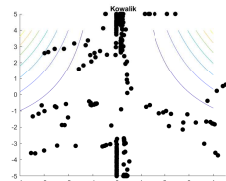
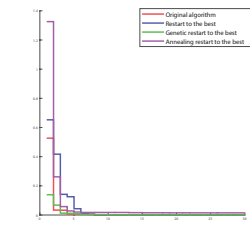
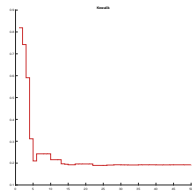
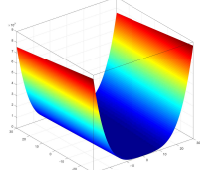
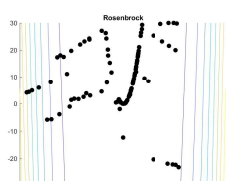
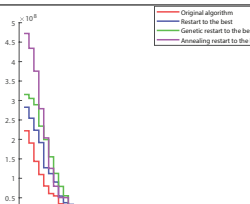
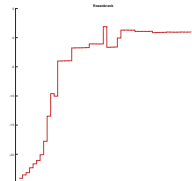
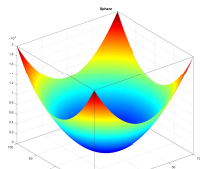
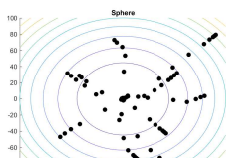
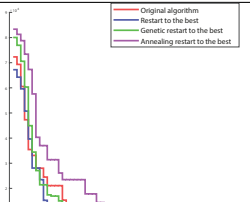
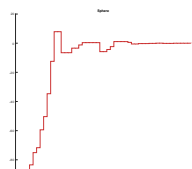
Table 4 shows the performance between RFO and other metaheuristic optimization algorithms. The results of the RFO shown on the table represent the average global minimum obtained after running the algorithm more than 1000 times for the multidimensional functions and 50 times for the Kowalik function. Almost all the individual results of RFO were better than the ones from the other algorithms, but the average penalized the global results when the drops of a rainfall got stuck in local minimums. This behavior only occurs with functions that have many local minimums, like

the Ackley function, but can be solved with the parallel implementations as they generate more raindrops, decreasing the probability of getting all drops stuck.

Tables 7, 8, and 9 represent the average results obtained after running the algorithm several times using the parallel implementations: restart to the best, genetic restart to the best and annealing restart to the best respectively. The parameters used for these tests are defined in Table 2, and the max value of H was 30 in all runs. All four multidimensional functions were minimized with 10 and 30 dimensions. The population and number of threads were different in some runs to determine the population and the number of threads that can help the quality of the result without hurting the performance. For each configuration, the algorithm was tested 50 times.

After comparing the quality and iterations between the 10-dimensional functions and the 30-dimensional functions in Table 6, it can be said that the more variables the function has, the harder it will be for the algorithm to get near to the global minimum. On average, it was found that the algorithm needs twice the amount of iterations to find the global minimum with the benchmark functions set to 30 dimensions. Also, the more population of raindrops the rainfall has, the

Table 5. Behavior of raindrops inside the benchmark functions

Function	3D Plot	Search history	Convergence	Trayjectory
F1				
F2				
F3				
F4				
F5				

closer it gets to the global minimum of the function. But having more raindrops can impact on the CPU time.

5. Conclusions

In this work, new implementations for the Rainfall Optimization Algorithm (RFO) are presented. Also, the RFO used for this paper has some improvements from the original algorithm by mimicking other natural behaviors such as the flow speed of descending raindrop and the distribution of the particles during a blast. These modifications allow a dynamic search

radius during the random search process. In addition to that, the algorithm can run multiple rainfalls in different threads at the same time during the optimization process. Moreover, all the rainfalls during the execution of the algorithm work as a team by sharing information between all the threads. The paper proposed three restarting techniques to share the positions and the fitness values of all raindrops of all threads. As shown in Table 5, RFO is a fast algorithm and the parallel implementations seek to improve the quality of the results without hurting the performance. Running the algorithm with multiple threads and using

Table 6. Comparison of the benchmark results obtained by RFO between the different approaches

Function	Dimensions	Method							
		Single Threaded		Restart to the best		Genetic restart to the best		Annealing restart to the best	
		Global minimum	Iterations	Global minimum	Iterations	Global minimum	Iterations	Global minimum	Iterations
F1	30	1.72E-04	369	4.86E-01	331	1.93E-01	343	3.92E-01	314
F2	30	9.86E-02	376	7.59E-03	274	2.61E-03	265	7.02E-15	371
F3	4	3.16E-04	3457	3.08E-04	239	3.08E-04	249	3.08E-04	480
F4	30	32.41	429	23.727	265	23.151	258	18.157	534
F5	30	1.46E-06	1020	1.09E-11	269	4.78E-13	256	1.01E-16	361

Table 7. RFO performance results in parallel with restart to the best method

Function	Threads	Global Minimum 10D*					Global minimum 30D*					
		Population				Iterations	Population					Iterations
		5	10	15	20		20	25	30	35	40	
F1	5	7.02E-04	1.91E-04	1.14E-09	6.55E-09	284	1.06E+00	3.32E-08	9.50E-01	8.89E-01	1.76E-08	339
	10	1.15E-06	1.61E-10	1.25E-10	1.83E-11	181	1.94E-09	1.09E-08	9.50E-01	7.51E-01	3.48E-09	328
	15	5.38E-09	1.89E-09	8.06E-11	1.98E-10	173	1.32E+00	1.30E-08	3.35E-01	6.70E-01	3.75E-01	326
F2	5	8.36E-02	4.31E-02	6.64E-02	6.64E-02	164	6.16E-03	2.34E-02	9.84E-03	5.04E-02	3.12E-11	273
	10	5.53E-02	6.41E-02	4.93E-02	4.07E-02	154	1.09E-04	9.12E-05	4.98E-03	3.74E-03	2.89E-05	275
	15	7.62E-02	5.29E-02	3.14E-02	4.43E-02	154	3.83E-03	6.81E-05	4.98E-03	4.51E-05	6.19E-03	274
F3*	5	3.08E-04	3.08E-04	3.08E-04	3.08E-04	248	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	248
	10	3.08E-04	3.08E-04	3.08E-04	3.08E-04	246	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	246
	15	3.08E-04	3.08E-04	3.08E-04	3.08E-04	225	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	225
F4	5	0.554	1.435	2.245	0.365	205	26.129	23.110	25.636	23.648	23.511	262
	10	0.706	0.381	0.523	0.144	191	23.146	23.272	22.666	22.643	23.716	267
	15	0.330	0.244	0.271	0.611	195	20.049	23.971	24.807	23.695	25.910	266
F5	5	2.76E-11	3.79E-15	1.46E-16	9.72E-20	163	4.23E-11	7.69E-11	7.74E-14	2.74E-13	2.96E-12	271
	10	2.12E-14	3.50E-14	1.94E-13	8.66E-17	166	6.26E-14	1.64E-14	9.30E-15	1.84E-14	2.11E-16	273
	15	1.88E-14	5.03E-12	6.26E-18	9.16E-15	162	3.79E-16	6.35E-15	1.31E-16	1.35E-16	9.22E-15	264

*F3 is always a four-dimensional function.

a restarting technique should also reduce the time of execution if the maximum number of physical threads is not exceeded.

To ensure that all the different implementations of the algorithm perform at least as good as the original RFO, the testing results are also presented. All the different variations of the algorithm with different configurations of the number of threads, dimensions, and populations were carried out with

different benchmark functions. The chosen functions vary in shape and behavior to ensure that the algorithm is capable to optimize any continuous optimization problem. The results that were shown can validate all the presented variations of the RFO as metaheuristic algorithms.

After analyzing the results of the benchmarking experiments given by the different implementations of the algorithm, it can be concluded that parallelization

Table 8. RFO performance results in parallel with genetic restart to the best method

Function	Threads	Global Minimum 10D*					Global minimum 30D*					
		Population				Iterations	Population					Iterations
		5	10	15	20		20	25	30	35	40	
F1	5	3.89E-04	1.18E-08	8.42E-10	1.40E-09	203	7.50E-01	4.80E-09	2.69E-08	2.93E-04	1.93E-08	392
	10	1.02E-06	3.98E-10	1.12E-10	1.80E-08	165	4.68E-01	5.91E-09	1.82E-04	1.65E-09	2.19E-10	325
	15	3.54E-08	5.10E-09	1.06E-10	4.48E-08	165	4.66E-01	2.27E-09	4.66E-01	2.92E-10	7.51E-01	311
F2	5	1.13E-01	6.03E-02	7.02E-02	5.05E-02	156	1.65E-04	8.51E-05	5.62E-05	3.75E-03	3.36E-05	273
	10	5.54E-02	5.90E-02	9.23E-02	3.69E-02	152	1.46E-04	7.96E-05	6.22E-05	1.45E-14	2.09E-02	265
	15	5.78E-02	6.27E-02	5.66E-02	5.17E-02	152	1.20E-04	3.78E-03	5.45E-05	3.99E-05	9.89E-03	258
F3*	5	3.08E-04	3.08E-04	3.08E-04	3.08E-04	270	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	270
	10	3.08E-04	3.08E-04	3.08E-04	3.08E-04	227	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	227
	15	3.08E-04	3.08E-04	3.08E-04	3.08E-04	249	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	249
F4	5	0.339	0.366	0.280	0.086	199	25.775	24.788	25.018	19.129	23.515	260
	10	0.494	0.850	0.176	0.176	200	23.978	24.294	21.289	21.435	23.324	257
	15	1.411	0.162	0.221	0.129	190	23.815	21.884	24.608	21.618	22.793	257
F5	5	3.08E-12	1.10E-13	3.80E-16	9.18E-17	158	1.66E-12	8.06E-14	5.13E-12	9.15E-14	5.91E-14	261
	10	7.24E-15	7.38E-12	3.87E-17	9.19E-15	155	4.69E-14	3.84E-14	6.97E-16	5.14E-16	7.24E-16	253
	15	2.98E-14	1.99E-14	4.63E-16	1.86E-13	156	5.88E-14	3.10E-17	7.06E-17	2.84E-16	4.22E-17	252

*F3 is always a four-dimensional function.

improves the solutions given by RFO. With multi-threading, more raindrops can be generated and satisfy the CPU time of the Equation 10. Having more artificial drops decreases the proportion of stuck drops and increases the probability of having drops exploring towards the global minimum. And the restarting techniques allowed raindrops that were heading in the wrong direction to restart to a position closer to the global minimum. As a result, all raindrops were exploring near the global minimum at the end of the execution.

$$\text{if } (N_{threads} \leq CPU_{threads}); \text{ then:} \quad (10)$$

$$t = K \cdot N_{drops} \cdot N_{threads} = K \cdot N_{drops}$$

In general terms, *simulated annealing restart to the best* was the restarting technique that returned the best quality solutions. But the behavior of raindrops in some functions can be benefited by the use of the other restarting techniques, especially the genetic restart to the best. In terms of that, possible further investigations about the hybridization between the different restarting techniques presented in the paper are worth to be done. Studying the hybrid restarting techniques as new

approaches for RFO can improve even more the solution accuracy.

In conclusion, if the purposed parallel approaches are to be considered as new implementations for RFO, this nature-inspired algorithm can satisfy the needs to solve real-world optimization problems. And the paper demonstrated the benefits of parallelization and restarting techniques to offer reliable solutions for metaheuristic optimization algorithms.

Table 9. RFO performance results in parallel with simulated annealing restart to the best method

Function	Threads	Global Minimum 10D*					Global minimum 30D*					
		Population				Iterations	Population					Iterations
		5	10	15	20		20	25	30	35	40	
F1	5	1.00E+01	6.15E-07	5.34E-09	2.05E-09	331	6.80E-01	3.92E-01	9.41E-01	6.10E-01	5.70E-01	333
	10	7.11E+00	5.94E-09	2.30E-09	9.34E-10	303	9.54E-01	6.30E-08	2.09E-01	2.55E-08	1.16E-01	305
	15	1.85E+00	2.38E-09	1.62E-09	1.21E-09	299	6.08E-01	2.70E-08	3.15E-01	1.03E-08	8.18E-09	303
F2	5	1.23E-01	7.75E-02	7.88E-02	6.91E-02	200	4.24E-14	3.35E-14	4.84E-15	2.16E-15	1.33E-16	377
	10	8.81E-02	6.69E-02	6.35E-02	5.17E-02	193	1.59E-14	2.55E-15	9.33E-16	4.00E-16	0.00E+00	372
	15	8.02E-02	6.60E-02	5.61E-02	6.25E-02	194	1.51E-15	7.77E-16	6.66E-17	1.55E-16	0.00E+00	363
F3*	5	3.08E-04	3.08E-04	3.08E-04	3.08E-04	583	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	583
	10	3.08E-04	3.08E-04	3.08E-04	3.08E-04	416	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	416
	15	3.08E-04	3.08E-04	3.08E-04	3.08E-04	410	3.08E-04	3.08E-04	3.08E-04	3.08E-04	3.08E-04	410
F4	5	0.489285	0.281047	0.189625	0.184099	324	21.7435131	19.9252982	17.6710526	16.7337463	14.041721	608
	10	0.325604	0.782692	0.436424	0.273304	307	21.4397651	19.2571002	17.2151353	18.9404568	16.4706906	495
	15	0.282014	0.339173	0.453077	0.113242	312	17.7622947	18.5656311	18.5975076	16.6545613	17.343305	500
F5	5	1.39E-15	7.10E-19	1.89E-18	7.96E-23	206	3.89E-16	8.53E-16	1.36E-16	2.77E-17	2.93E-20	369
	10	9.59E-19	1.57E-20	7.54E-23	1.50E-22	207	1.67E-17	2.30E-17	4.23E-17	5.26E-19	5.01E-19	360
	15	2.02E-18	2.83E-21	5.91E-22	2.50E-23	205	2.22E-17	7.13E-19	4.63E-18	6.70E-19	2.75E-19	353

*F3 is always a four-dimensional function.

References

- [1] KABOLI, S.H.A., SELVARAJ, J. and RAHIM, N. (2017) Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems. *Journal of Computational Science* **19**: 31–42. doi:10.1016/j.jocs.2016.12.010.
- [2] YANG, X.S. (2005) Engineering optimizations via nature-inspired virtual bee algorithms. In *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach* (Springer Berlin Heidelberg), 317–323. doi:10.1007/11499305_33.
- [3] FISTER JR, I., YANG, X.S., FISTER, I., BREST, J. and FISTER, D. (2013) A brief review of nature-inspired algorithms for optimization. *Elektrotehnikski Vestnik/Electrotechnical Review* **80**.
- [4] WEDYAN, A., WHALLEY, J. and NARAYANAN, A. (2017) Hydrological cycle algorithm for continuous optimization problems. *Journal of Optimization* **2017**: 1–25. doi:10.1155/2017/3828420.
- [5] ZANCHETTIN, C. and LUDERMIR, T.B. (2008) Feature subset selection in a methodology for training and improving artificial neural network weights and connections. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (IEEE). doi:10.1109/ijcnn.2008.4634065.
- [6] HARIK, G.R., LOBO, F.G. and GOLDBERG, D.E. (1999) The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation* **3**(4): 287–297. doi:10.1109/4235.797971.
- [7] HOU, E., ANSARI, N. and REN, H. (1994) A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems* **5**(2): 113–120. doi:10.1109/71.265940.
- [8] COSOLA, E., GENOVESE, K., LAMBERTI, L. and PAPPALERTERE, C. (2008) A general framework for identification of hyper-elastic membranes with moiré techniques and multi-point simulated annealing. *International Journal of Solids and Structures* **45**(24): 6074–6099. doi:10.1016/j.ijsolstr.2008.07.019.
- [9] ZHANG, Y. and WU, L. (2012) A robust hybrid restarted simulated annealing particle swarm optimization technique. *Advances in Computer Science and its Applications* **1**(1): 5–8.
- [10] MCGOOKIN, E.W. and MURRAY-SMITH, D.J. (2006) Submarine manoeuvring controllers' optimisation using simulated annealing and genetic algorithms. *Control Engineering Practice* **14**(1): 1–15. doi:10.1016/j.conengprac.2005.01.002.
- [11] ZHANG, Y., WU, L., NEGGAZ, N., WANG, S. and WEI, G. (2009) Remote-sensing image classification based on an improved probabilistic neural network. *Sensors* **9**(9): 7516–7539. doi:10.3390/s90907516.
- [12] SHANG, Y.W. and QIU, Y.H. (2006) A note on the extended rosenbrock function. *Evolutionary Computation* **14**(1): 119–126. doi:10.1162/evco.2006.14.1.119.
- [13] KARABOGA, D. and BASTURK, B. (2007) A powerful and efficient algorithm for numerical function optimization:

- artificial bee colony (ABC) algorithm. *Journal of Global Optimization* 39(3): 459–471. doi:10.1007/s10898-007-9149-x.
- [14] TANG, K., LI, X., SUGANTHAN, P.N., YANG, Z. and WEISE, T. (2008) Benchmark functions for the cec'2010 special session and competition on large-scale. In *Computer Science*.
- [15] LOCATELLI, M. (2003) A note on the griewank test function. *Journal of Global Optimization* 25(2): 169–174. doi:10.1023/a:1021956306041.
- [16] WINFIELD, D. (1973) Function minimization by interpolation in a data table. *IMA Journal of Applied Mathematics* 12(3): 339–347. doi:10.1093/imamat/12.3.339.
- [17] SURJANOVIC, S. and BINGHAM, D., Virtual library of simulation experiments: Test functions and datasets, Retrieved February 26, 2020, from <http://www.sfu.ca/~ssurjano>.
- [18] SCHICHL, H. and NEUMAIER, A. (2004) The NOP-2 modeling language. In *Applied Optimization* (Springer US), 279–291. doi:10.1007/978-1-4613-0215-5_15.
- [19] EBERHART, R.C. and SHI, Y. (1998) Comparison between genetic algorithms and particle swarm optimization. In *Lecture Notes in Computer Science* (Springer Berlin Heidelberg), 611–616. doi:10.1007/bfb0040812.
- [20] HE, S., WU, Q. and SAUNDERS, J. (2009) Group search optimizer: An optimization algorithm inspired by animal searching behavior. *IEEE Transactions on Evolutionary Computation* 13(5): 973–990. doi:10.1109/tevc.2009.2011992.