IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 3, 2020, accepted March 12, 2020, date of publication March 16, 2020, date of current version March 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2981196

Wingsuit Flying Search–A Novel Global Optimization Algorithm

NERMIN COVIC, (Member, IEEE), AND BAKIR LACEVIC, (Member, IEEE)
Faculty of Electrical Engineering, University of Sarajevo, Sarajevo 71000, Bosnia and Herzegovina
Corresponding author: Nermin Covic (ncovic1@etf.unsa.ba)

ABSTRACT In this paper, a novel global optimization algorithm – Wingsuit Flying Search (WFS) is introduced. It is inspired by the popular extreme sport – wingsuit flying. The algorithm mimics the intention of a flier to land at the lowest possible point of the Earth surface within their range, i.e., a global minimum of the search space. This is achieved by probing the search space at each iteration with a carefully picked population of points. Iterative update of the population corresponds to the flier progressively getting a sharper image of the surface, thus shifting the focus to lower regions. The algorithm is described in detail, including the mathematical background and the pseudocode. It is validated using a variety of classical and CEC 2020 benchmark functions under a number of search space dimensionalities. The validation includes the comparison of WFS to several nature-inspired popular metaheuristic algorithms, including the winners of CEC 2017 competition. The numerical results indicate that WFS algorithm provides considerable performance improvements (mean solution values, standard deviation of solution values, runtime and convergence rate) with respect to other methods. The main advantages of this algorithm are that it is practically parameter-free, apart from the population size and maximal number of iterations. Moreover, it is considerably ''lean'' and easy to implement.

INDEX TERMS Wingsuit flying, metaheuristic search, nature-inspired algorithms, global optimization.

I. INTRODUCTION

A. TYPICAL PROBLEMS OF FINDING A GLOBAL OPTIMUM

A nature has been a primary inspiration for many modern metaheuristic algorithms. For instance, Genetic Algorithm was inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA) [34]. Another popular nature-inspired optimization algorithm is Particle Swarm Optimization [6]. It is inspired by the apparent swarm intelligence of bird flocks and fish schools. Generally, nature-inspired metaheuristic algorithms have been used in a wide range of optimization problems, such as the traveling salesman problem, optimal control, system identification, etc [5], [6], [8], [18], [29], [57]. The power of almost all modern metaheuristics comes from the fact that they imitate the best feature in nature.

Many global optimization metaheuristic algorithms, such as Nelder-Mead's algorithm (NMA) [40], Pattern-Search (PS) [15], Simulated Annealing (SA) [21] and Tabu Search (TS) [12] are not always capable of finding global optimum for challenging multimodal test functions [36], [44].

It is desirable to acquire a global picture of search space, which is usually constrained. Thus, the search is more likely to follow the right direction. For instance, the optimization of Rastrigin function would benefit should the search is more oriented towards the center of the search space. On the other hand, to optimize Holder Table, searching the limits of the search space may seem a better idea.

Challenging global optimization problems have triggered an explosion of nature-inspired search algorithms that usually rely on metaphor-based metaheuristics. Though the metaphor paradigm has been criticized in the literature (see e.g., [47]), several algorithms have turned out to be very efficient in solving complex optimization problems, both in terms of classical benchmarks (e.g., CEC 2020 [25]) and real-world engineering scenarios. Instances of these algorithms are: Genetic Algorithm (GA) [34], Simulated Annealing (SA) [21], Evolution Algorithms (EA) [39], [42], Particle Swarm Optimization (PSO) [6], Artificial Fish-Swarm Algorithm (AFSA) [24], Termite Algorithm (TA) [27], Ant Colony Optimization (ACO) [10] and its improved

The associate editor coordinating the review of this manuscript and approving it for publication was Huaqing Li.

versions [9] and [55], Artificial Bee Colony (ABC) [4], Monkey Search (MS) [38], Cuckoo Search (CS) [60], Firefly Algorithm (FF) [56], Gravitational Search Algorithm (GSA) [43], Bat Algorithm (BA) [58], Flower Pollination Algorithm (FPA) [59], Directed Bee Colony Optimization Algorithm (DBCOA) [20], LSHADE [49], UMOEA [11], Grey Wolf Optimizer (GWO) [33], Lightning Search Algorithm (LSA) [46], Moth-Flame Optimization Algorithm (MFOA) [31], Artificial Algae Algorithm (AAA) [50], Ant Lion Optimizer (ALO) [30], Elephant Herding Optimization (EHO) [52], Whale Optimization Algorithm (WOA) [32], Competitive Optimization Algorithm (COOA) [45], Lion Optimization Algorithm (LOA) [61], Crow Search Algorithm (CSA) [2], Thermal Exchange Optimization (TEO) [19], Lévy flight trajectory-based Whale Optimization Algorithm (LWOA) [26], LSHADE-cnEpSin [3], LSHADE-SPACMA [35], EBOwithCMAR [22], Moth Search (MS) [51], Earthworm Optimization Algorithm (EOA) [54], Butterfly Optimization Algorithm (BOA) [1], Squirrel Search Algorithm (SSA) [16], Harris Hawks Optimization (HHO) [14], Monarch Butterfly Optimization (MBO) [53], etc.

B. CONTRIBUTION HIGHLIGHTS AND PAPER ORGANIZATION

The main contributions of this paper are as follows:

- The formulation of Wingsuit Flying Search (WFS) – a novel metaheuristic search algorithm is provided;
- WFS is practically parameter-free, apart from the population size and maximal number of iterations, which represents the main advantage of this algorithm;
- WFS is considerably "lean" and easy to implement;
- WFS is able to rapidly converge to the global optimum solution of the related objective function while keeping the exploration of the search space;
- WFS is suitable for parallelized computation since it can optimize different parts of the search space independently;
- An extensive computational study, based on 30 classical and 10 CEC 2020 benchmark functions, indicates promising features of WFS algorithm since it outperforms competing methods in most scenarios.

The remainder of the paper is organized as follows. First, the motivation behind WFS is briefly outlined. Section II brings a detailed description of the algorithm, including complete mathematical background and the pseudocode. Sections III and IV provide a theoretical and numerical study, respectively, where WFS is compared to related metaheuristic search methods. Finally, Section V brings some concluding remarks.

C. MOTIVATION BEHIND WFS ALGORITHM

The basic idea is to gain a rough picture of a search space. Even the rough one may provide a lot of useful information. In that regard, this work is inspired by the popular sport – wingsuit flying (Fig. 1). When a flier (algorithm) starts flying (running), she/he sees a rough picture of Earth surface, which is here treated as the search space. The flier's view toward different areas corresponds to a population of points in the search space. It is assumed that the flier wants to land at the lowest point (global minimum). Though the flier cannot see where the minimum is, she/he can roughly see the valleys. Therefore, large portions of terrain (e.g., mountains) can be immediately eliminated, so the flier will not likely return to them. A clear picture of the Earth is not available immediately, due to human eye imperfections, possible clouds, fog and other disturbances (Fig. 1a).


(a)

(b)

FIGURE 1. Display of a rough (a) and clear (b) picture of the Earth surface.

After awhile, the flier approaches the valley and the Earth picture becomes sharper (Fig. 1b). Controlling the flight is ensured by auxiliary wings located between flier's arms/legs and body. Moreover, changing wings' surface (spreading and shrinking arms and legs), it is possible to control flying velocity thus affecting the exploitation-exploration ratio, which will be discussed later (see Subsection II-B). As of now, the flier is more likely to clearly see the regions of interest and stay focused on them. A point at which the flier lands is called terminating point, i.e., a solution computed by the algorithm.

II. ALGORITHM DESCRIPTION

The algorithm is essentially a population-based search that iteratively updates the set of candidate solution points. In the sequel, we describe the proposed approach in detail. First, we need to determine how initial set of points is generated.

A. GENERATING INITIAL POINTS

Let N^* be the number of points in a single iteration (i.e., population size) of the algorithm and let

$$N = N^* - 2 \tag{1}$$

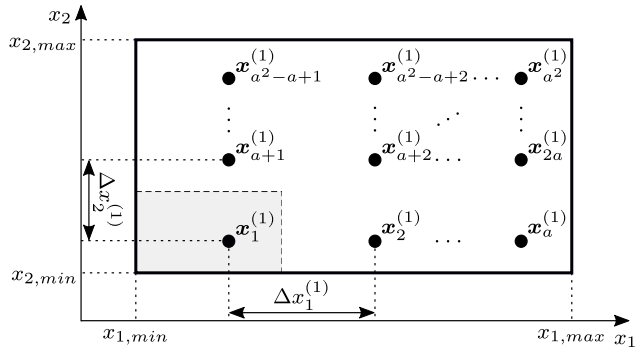

FIGURE 2. The layout of N_0^2 initial points located in grid nodes (in the first iteration) in 2D search space.

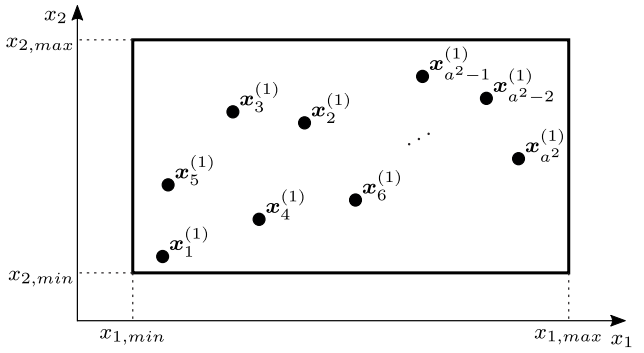

FIGURE 3. The layout of N_0^2 initial points generated by Halton sequence (in the first iteration) in 2D search space.

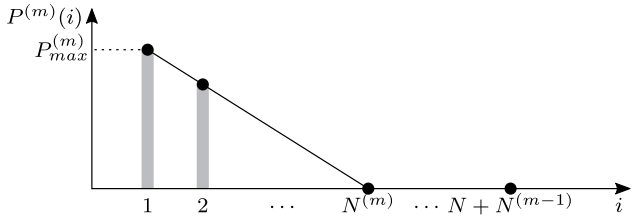

FIGURE 4. The neighborhood size for each point in m-th iteration.

be the number of initial points in a box-constrained n-dimensional search space defined as $\{\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_n]^T \in \mathbb{R}^n : \boldsymbol{x}_{min} \preceq \boldsymbol{x} \preceq \boldsymbol{x}_{max}\}$, in which the cost function $f : \mathbb{R}^n \mapsto \mathbb{R}$ needs to be minimized. Condition (1) is imposed since two more points are added to the search space at the end of each iteration, which will be discussed later (see subsection II-D).

In the first iteration ($m = 1$), each point $\boldsymbol{x}_i$, $i \in \{1, 2, \ldots, N\}$, may be located in a node in n-dimensional grid. The number of nodes per dimension is

$$N_0 = \left\lceil \sqrt[n]{N} \right\rceil, \quad N \geq 2. \tag{2}$$

Clearly, if $(N_0 - 1)^n < N < N_0^n$ than $N_0^n - N$ random chosen nodes will not contain any points, thus the grid will be incomplete.

For instance, the case when $n = 2$ and $N = N_0^2$ (full grid) is shown in Fig. 2. The point $\boldsymbol{x}^{(1)}$ is located randomly into the box constrained with $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{min} + \Delta\boldsymbol{x}^{(1)}$, where $\Delta\boldsymbol{x}^{(1)} = \left[\Delta x_1^{(1)}\ \Delta x_2^{(1)}\ \ldots\ \Delta x_n^{(1)}\right]^T$ is referred to as *initial discretization step*, which is defined as

$$\Delta\boldsymbol{x}^{(1)} = \frac{\boldsymbol{x}_{max} - \boldsymbol{x}_{min}}{N_0}. \tag{3}$$

Other points are located according to $\boldsymbol{x}_1^{(1)}$ in order to form a grid (full or incomplete). Then, the cost function is evaluated at each point and the minimum is selected. This is the end of the first iteration.

To improve the uniformity of points covering the search space, a low-discrepancy sequence can be used as an alternative [23], [41]. For this purpose, we use (scrambled) Halton sequence [13]. Fig. 3 shows the layout of N_0^2 points generated by this sequence. Its low-discrepancy feature becomes more obvious when $N \approx \lfloor (N_0^n + (N_0 - 1)^n)/2 \rfloor$, when the set of points would form an incomplete grid.

Note that the step defined in (3) does not apply in generating points by Halton sequence, yet it will be used in the later stages of the algorithm (see subsection II-C).

B. DETERMINING NEIGHBORHOOD SIZE FOR EACH POINT

For $m \geq 2$, the points from $(m-1)$-st iteration are sorted in ascending order w.r.t. their solution values. The first point is the "best" one, and it is assigned a neighborhood with the largest number of points $P_{max}^{(m)}$. The second point will be assigned a neighborhood with less points. The $N^{(m)}$-th point is assigned zero neighborhood points. This is shown in Fig. 4. Note that $N^{(1)} = 0$.

In general, only the first $N^{(m)}$ points are assigned with neighborhood points. Moreover, there will be $\sum_{i=1}^{N^{(m)}} P^{(m)}(i)$ "new" points that are passed to the next iteration. This number is kept equal to N in order to maintain the constant number of cost function evaluations at each iteration. We assume the linear dependence of the function $P^{(m)}(i)$ w.r.t. point rank i as

$$P^{(m)}(i) = \left\lceil P_{max}^{(m)} \left(1 - \frac{i-1}{N^{(m)} - 1}\right) \right\rceil. \tag{4}$$

Hence, the triangle area from Fig. 4 is equal N. Therefore, it follows that

$$N^{(m)} = \left\lceil \frac{2N}{P_{max}^{(m)}} \right\rceil. \tag{5}$$

Since $P^{(m)} \in \mathbb{N}_0$, it may happen that $\sum_{i=1}^{N^{(m)}} P^{(m)}(i)$ is not exactly N. In that case, $P^{(m)}\left(N^{(m)}\right)$ or $P^{(m)}\left(N^{(m)} - 1\right)$ are modified to satisfy the imposed condition.

Parameter $P_{max}^{(m)}$ represents the maximal number of neighborhood points around $\boldsymbol{x}_i^{(m)}$ in m-th iteration. If their coordinates are generated using arrangements with repetition from the set $\left\{x_{k,i}^{(m)} - \Delta x_k^{(m)},\ x_{k,i}^{(m)},\ x_{k,i}^{(m)} + \Delta x_k^{(m)}\right\}$, $k \in \{1, 2, \ldots, n\}$, then $P_{max}^{(m)} = 3^n - 1$. We refer to this neighborhood as *grid neighborhood*. For instance, if $n = 20$ then $P_{max}^{(m)}$ equals $3.49 \cdot 10^9$ points, which takes about 558 GB of memory (assuming that each coordinate takes 64 bites). For larger n, $P_{max}^{(m)}$ clearly becomes huge, which triggers memory issues.

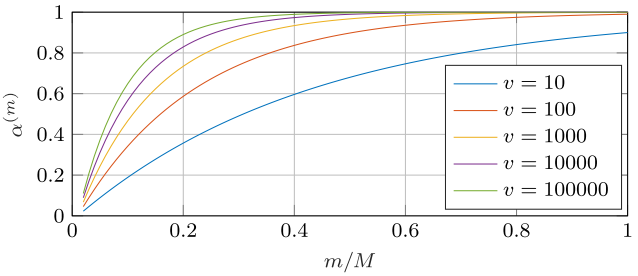

FIGURE 5. $\alpha^{(m)}$ versus m through M iterations for different values of the flier's velocity v.

Thus, we propose that $P_{max}^{(m)}$ is computed as

$$P_{max}^{(m)} = \left\lceil \alpha^{(m)} N \right\rceil, \tag{6}$$

where $\alpha^{(m)} \in (0, 1)$ and is computed as

$$\alpha^{(m)} = 1 - v^{-\frac{m-1}{M-1}}, \quad m \geq 2, \tag{7}$$

where M is the number of the algorithm iterations and $v > 0$ is the algorithm parameter to which we refer to as *flier's velocity*. Fig. 5 shows how $\alpha^{(m)}$ changes through M iterations for different values of v. Experiments suggest that the value of v does not significantly affect the algorithm performance. Therefore, it may be set to a default value, so the user does not need to tune it (see section IV).

Equation (7) is proposed to encourage smaller $P_{max}^{(m)}$ at the beginning, which causes $N^{(m)}$ to be larger according to (5). Moreover, all points are approximately equally treated, because the function slope in Fig. 4 is small. It means that the exploration of the search space is a priority. As m gets closer to M, the slope in Fig. 4 becomes larger, which brings more focus on better points. In other words, the exploitation gets more attention.

C. GENERATING NEIGHBORHOOD POINTS

After assigning neighborhood size to each point, the question is how to generate N new points. The flier's "altitude" is decreasing through iterations rendering resolution of the search space higher. The discretization step consequently gets smaller as

$$\Delta \boldsymbol{x}^{(m)} = \left(1 - \alpha^{(m)}\right) \Delta \boldsymbol{x}^{(1)}, \quad m \geq 2. \tag{8}$$

Thus, we refer to parameter $\alpha^{(m)}$ as *search sharpness* in m-th iteration.

Afterwards, N new points, i.e., neighborhood points denoted $\boldsymbol{y}_j\left(\boldsymbol{x}_i^{(m)}\right)$, $j \in \left\{1, 2, \ldots, P^{(m)}(i)\right\}$, are generated. Fig. 6 shows an example scenario in 2D search space. Clearly, blue points are neighborhood points, and white ones represents their possible locations. The point $\boldsymbol{x}_1^{(m)}$, having the lowest value, determines the current solution $\boldsymbol{x}^{*(m)}$.

For each point $\boldsymbol{x}_i^{(m)}$, a vector $\boldsymbol{v}_i^{(m)} = \boldsymbol{x}_1^{(m)} - \boldsymbol{x}_i^{(m)}$, oriented towards the current solution, is created, to which we refer to as *neighborhood vector*. It determines the direction of generating neighborhood points. Each of its coordinates are

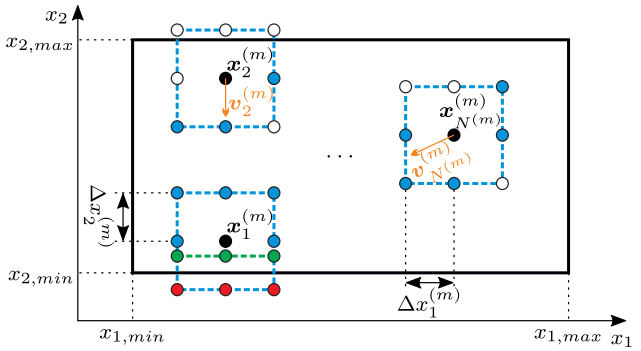

FIGURE 6. The layout of first $N^{(m)}$ points (in m-th iteration) and their grid neighborhoods in 2D search space.

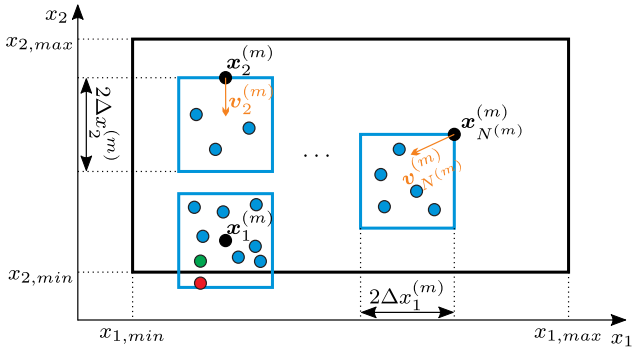

FIGURE 7. The layout of first $N^{(m)}$ points (in m-th iteration) and their Halton neighborhoods in 2D search space.

generated using arrangements with repetition from the three following sets:

$$\begin{aligned}
S_{k,1}\left(\boldsymbol{x}_i^{(m)}\right) &= \left\{x_{k,i}^{(m)} - \Delta x_k^{(m)},\ x_{k,i}^{(m)}\right\}, \quad \text{if } v_{k,i}^{(m)} < 0;\\
S_{k,2}\left(\boldsymbol{x}_i^{(m)}\right) &= \left\{x_{k,i}^{(m)},\ x_{k,i}^{(m)} + \Delta x_k^{(m)}\right\}, \quad \text{if } v_{k,i}^{(m)} > 0;\\
S_{k,3}\left(\boldsymbol{x}_i^{(m)}\right) &= S_{k,1}\left(\boldsymbol{x}_i^{(m)}\right) \cup S_{k,2}\left(\boldsymbol{x}_i^{(m)}\right), \quad \text{if } v_{k,i}^{(m)} = 0.
\end{aligned} \tag{9}$$

Thus, the maximal neighborhood size is $3^{\lambda} \cdot 2^{n-\lambda} - 1$, where λ is the number of vector $\boldsymbol{v}_i^{(m)}$ coordinates that are equal to zero. If $\lambda = 0$, the neighborhood is *strictly directed* (e.g., around $\boldsymbol{x}_{N^{(m)}}^{(m)}$), and if not, it is *non-strictly directed neighborhood* (e.g., around $\boldsymbol{x}_2^{(m)}$). So, there are two options for choosing neighborhood points:

- If $P^{(m)}(i) \leq 3^{\lambda} \cdot 2^{n-\lambda} - 1$, then points are chosen randomly from directed neighborhood (e.g., $\lambda = 1$ for $\boldsymbol{x}_2^{(m)}$);
- If $3^{\lambda} \cdot 2^{n-\lambda} - 1 < P^{(m)}(i) \leq 3^n - 1$, then all points from directed neighborhood and some remaining points from non-directed neighborhood are chosen (e.g., $\lambda = 0$ for $\boldsymbol{x}_{N^{(m)}}^{(m)}$).

This means that directed neighborhood points have a priority. It is important to stress that if some points "fall out" of the search space (red points in Fig. 6), they are mirrored back w.r.t. border of the search space (green points).

Neighborhood points may also be generated using Halton sequence, as it is shown in Fig. 7. We refer to this neighborhood as *Halton neighborhood*. It is obvious that, in general

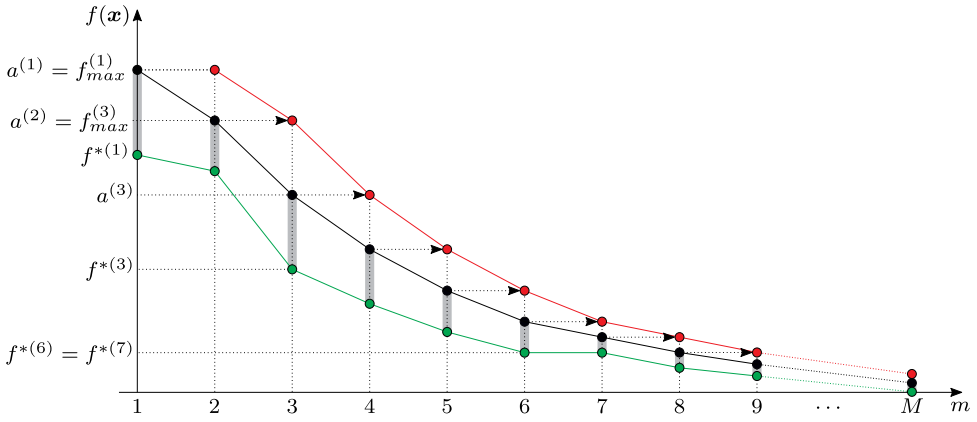

FIGURE 8. The algorithm convergence towards the global minimum $f(x^*) = 0$.

case, points can be located anywhere into n-dimensional box with dimensions of double discretization steps $2\Delta\boldsymbol{x}^{(m)}$.

All the logic described earlier for grid point sets applies for Halton neighborhoods as well.

After generating all neighborhoods, the cost function is evaluated at each of N new points and the minimum is selected from $N^{(m)} + N$ points. Before the end of the m-th iteration, two more points will be generated and added to the population, which is explained in the sequel.

D. GENERATING CENTROID AND RANDOM POINT

In reality, at the beginning, the flier tends to be located above the middle of the Earth surface of interest, since it is a priori the best way to deal with unknown surrounding. It proves beneficial to have a point which plays a role of the flier's current location. We refer to this point as *centroid*, and it is computed as weighted average

$$\boldsymbol{z}_1^{(m)} = \frac{\sum_{i=1}^{N^{(m)}+N} \boldsymbol{W}^{(m)}(i) \cdot \gamma\left(\boldsymbol{W}^{(m)}(i)\right)}{\sum_{i=1}^{N^{(m)}+N} \gamma\left(\boldsymbol{W}^{(m)}(i)\right)}, \tag{10}$$

where $\boldsymbol{W}^{(m)}$ is the matrix[1] containing all points, $\boldsymbol{x}_i^{(m)}$ and $\boldsymbol{y}_j\left(\boldsymbol{x}_i^{(m)}\right)$, represented as columns. Its dimensions are $n \times \left(N^{(m)} + N\right)$ and $\boldsymbol{W}^{(m)}(i)$ denotes its i-th column. Weighting factor $\gamma(\cdot)$ is computed for point $\boldsymbol{x}$ as

$$\gamma(\boldsymbol{x}) = \begin{cases} 1 - \dfrac{f(\boldsymbol{x}) - f^{*(m)}}{a^{(m)} - f^{*(m)}}, & f(\boldsymbol{x}) \le a^{(m)}; \\ 0, & f(\boldsymbol{x}) > a^{(m)}. \end{cases} \tag{11}$$

where $f^{*(m)}$ represents a current solution value in m-th iteration. The value at $N^{(m)}$-th point is referred to as *flier's altitude* in m-th iteration, and is denoted $a^{(m)}$. Clearly, all points above $a^{(m)}$ are ignored and only the first $N^{(m)}$ are analyzed, as it is stated in subsection II-B. This resembles the real scenario, where the flier is unable to see a ''terrain'' above them.

The above description is illustrated in Fig. 8, where the algorithm convergence towards the global minimum $f(\boldsymbol{x}^*) = 0$ is shown. Green line represents the algorithm curve convergence (changing of $f^{*(m)}$), black line is the flier's altitude curve (changing of $a^{(m)}$), and red line represents the profile of $f^{(m)}_{max}$ through iterations. All values within the interval $(f^{*(m)}, a^{(m)})$ lay on gray line, and they are used to generate neighborhoods in the next iteration. The best value is always preserved (e.g., $f^{*(6)} = f^{*(7)}$). Clearly, the condition $a^{(m-1)} < a^{(m)}$ is always satisfied, which resembles the real scenario, where the flier's altitude decreases in time.

Furthermore, to encourage the algorithm exploration capacities, a *random point* $\boldsymbol{z}_2^{(m)}$ is added to the search space. This point is chosen randomly from the minimal hypervolume axis-aligned bounding box which covers all the points from m-th iteration.

Afterwards, the cost function is evaluated at points $\boldsymbol{z}_1^{(m)}$ and $\boldsymbol{z}_2^{(m)}$, and $f^{*(m)}$ is updated if necessary, which ends the m-th iteration. Now, imposing (1) becomes clear.

The algorithm runs until some stopping condition is satisfied. Typically, it is a maximal number of iterations M, or reaching a minimal discretization step Δx_{min}, e.g., $||\Delta\boldsymbol{x}^{(m)}||_\infty \le \Delta x_{min}$.

It should be noted that centroid and random point are also considered in the first iteration, when $a^{(1)}$ represents the maximal evaluated value.

E. COMPARISON BETWEEN GRID AND HALTON POINTS

In order to get a better look on a difference between generating points by grid (grid points) versus Halton sequence (Halton points), Fig. 9 shows their layout in the search space after $m/M \in \{0.1,\ 0.2,\ 0.5\}$ when optimizing Rastrigin function for $n = 2$. Green point represents the global optimum. Red points are centroid and random point, and blue ones are other points. After 10 % of WFS runtime ($m/M = 0.1$), it is easy to spot that Halton points are more uniformly distributed in the search space than grid points. Also, this feature is more evident when the grid contains more ''holes'' (in incomplete grid), which is inevitable in high-dimensional search spaces.

After 20 % of WFS runtime ($m/M = 0.2$), Halton points clearly provide better exploration, while grid points becomes aligned unpredictably, which leaves some parts of the search space, even those close to global optimum completely unexplored. On the other hand, Halton points

[1] Actually, it is *pattern matrix* (see Sec. III).

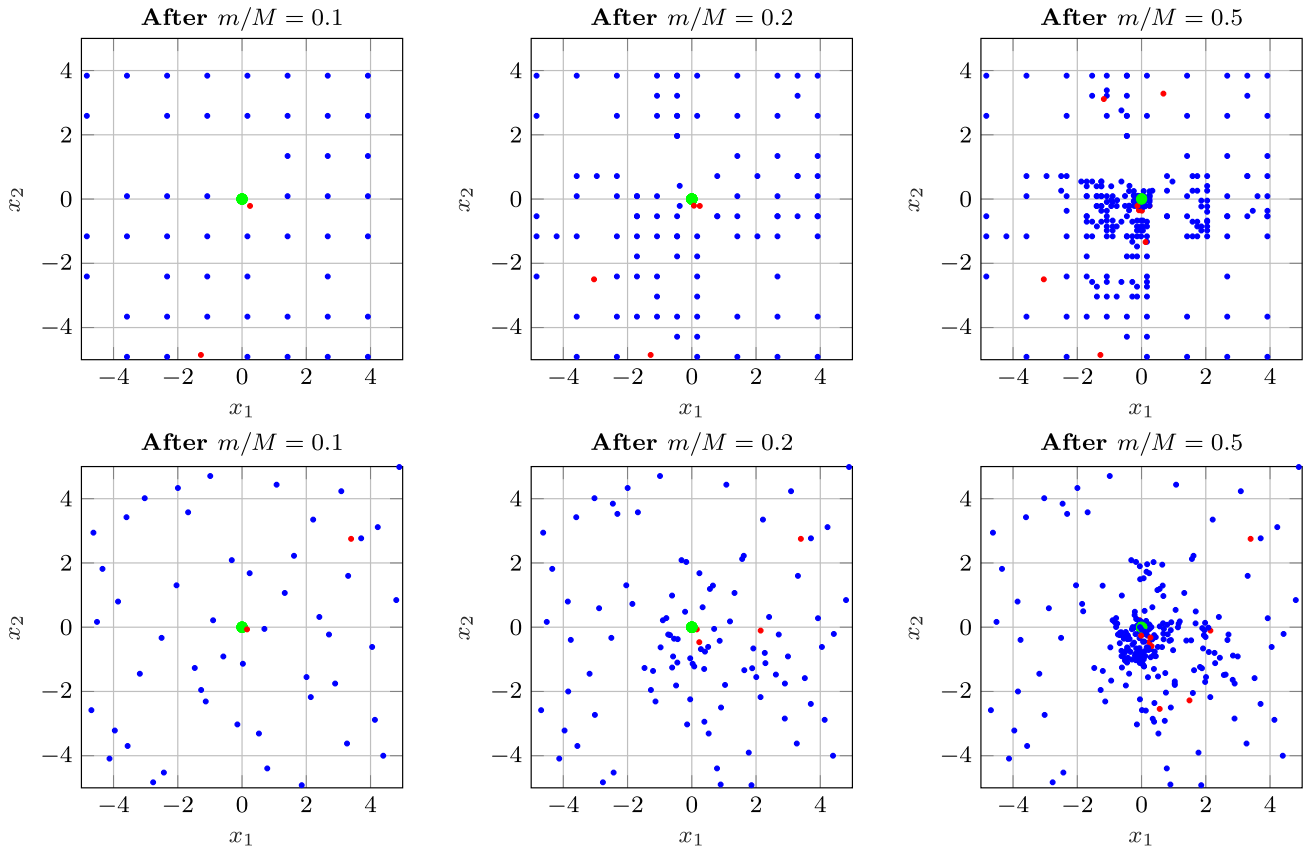

FIGURE 9. Layout of grid points (three top figures) versus Halton points (three bottom figures) in the search space after $m/M \in \{0.1,\ 0.2,\ 0.5\}$ when optimizing Rastrigin function for $n = 2$.

seem to enable better balance between exploration and exploitation. This is somewhat indicated after 50 % of WFS runtime ($m/M = 0.5$).

Similar comparison is performed for some other classical bechmark functions [17], and Halton points consistently provide better results.

F. PSEUDOCODE OF WFS ALGORITHM

Pseudocode of WFS algorithm is given in Algorithm 1. The pseudocode of the function *CRP* is given in Algorithm 2.

Used abbreviations and their meanings are:

- *GenerateIP* – *GenerateInitialPoints*,
- *CRP* – *CentroidAndRandomPoint*,
- *CheckSP* – *CheckStoppingCondition*,
- *GetNS* – *GetNeighborhoodSizes*,
- *GenerateANP* – *GenerateAllNeighborhoodPoints*,
- *GenerateCRP* – *GenerateCentroidAndRandomPoint*.

Some of the variables used in Algorithm 1 are defined as:

- $\boldsymbol{X}^{(m)} = \left[\boldsymbol{x}_1^{(m)}\ \boldsymbol{x}_2^{(m)}\ \ldots\ \boldsymbol{x}_{N^{(m)}}^{(m)}\right]$,
- $\boldsymbol{Y}^{(m)} = \left[y_1\left(\boldsymbol{x}_1^{(m)}\right)\ \ldots\ y_1\left(\boldsymbol{x}_{N^{(m)}}^{(m)}\right)\ \ldots\right.$
  $\left.\ldots\ y_{P^{(m)}(1)}\left(\boldsymbol{x}_1^{(m)}\right)\ \ldots\ y_{P^{(m)}(N^{(m)})}\left(\boldsymbol{x}_{N^{(m)}}^{(m)}\right)\right]$,
- $\boldsymbol{Z}^{(m)} = \left[\boldsymbol{z}_1^{(m)}\ \boldsymbol{z}_2^{(m)}\right]$.

Note that the function $f(\boldsymbol{X})$ returns cost function values at each point from matrix $\boldsymbol{X}$ (e.g., $f\left(\boldsymbol{X}^{(m)}\right) = \left[f\left(\boldsymbol{x}_1^{(m)}\right)\right.$ $\left.\ldots\ f\left(\boldsymbol{x}_{N^{(m)}}^{(m)}\right)\right]^T$). Quantity $\boldsymbol{X}(i)$ denotes i-th column (point) from matrix $\boldsymbol{X}$. The term $[\boldsymbol{X},\ \boldsymbol{Y}]$ denotes horizontal concatenation of matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Having in mind that the cost function evaluation (Line 7 and 20 in Algorithm 1, and Line 3 in Algorithm 2) is computationally the most expensive procedure within WFS, clearly the complexity of WFS is $O(M \cdot N^*)$, where $M \cdot N^*$ stands for total number of evaluations (M is the number of algorithm iterations and N^* is the population size). In case the number of cost function evaluations does not implicitly account for the dimensionality n, the complexity is $O(M \cdot N^* \cdot n)$.

III. A CONCEPTUAL COMPARISON BETWEEN WFS AND SIMILAR ALGORITHMS

Many metaheuristic algorithms use *pattern matrix* [7], which includes considered solutions in a single algorithm iteration. The best solution in the matrix is favorable to be improved rapidly through iterations. The measure of that improvement is referred to as *convergence rate*. In PSO algorithm, the pattern matrix is referred to as swarm and each pattern corresponds to an artificial particle. Similarly, in ABC algorithm each pattern corresponds to a nectar source. In CS algorithm, a pattern is considered as an artificial nest. In GA,

Algorithm 1 Wingsuit Flying Search

Input: $f,\ n,\ [\boldsymbol{x}_{min},\ \boldsymbol{x}_{max}],\ N^*,\ M,\ \Delta\boldsymbol{x}_{min}$
Output: $\boldsymbol{x}^{*(m)},\ f^{*(m)}$
$m \leftarrow 1$
$v \leftarrow rand(10,\ 100)$ // random real number from 10 to 100
$N \leftarrow N^* - 2$
$N_0 \leftarrow ceil(N\text{^}(1/n))$ // Eq.2
$\Delta\boldsymbol{x}^{(1)} = (\boldsymbol{x}_{max} - \boldsymbol{x}_{min})/N_0$ // Eq.3
$\boldsymbol{X}^{(1)} \leftarrow GenerateIP\left(N,\ N_0,\ \Delta\boldsymbol{x}^{(1)}\right)$ // Subsec.II-A, Fig.2 or Fig.3
$\boldsymbol{f}^{(1)} \leftarrow f\left(\boldsymbol{X}^{(1)}\right)$
$a^{(1)} \leftarrow max\left(\boldsymbol{f}^{(1)}\right)$
$\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ \boldsymbol{x}^{*(m)},\ f^{*(m)}\right) \leftarrow CRP\left(\boldsymbol{X}^{(1)},\ \boldsymbol{f}^{(1)},\ a^{(1)}\right)$
$SC \leftarrow CheckSC\left(\Delta\boldsymbol{x}^{(1)},\ \Delta\boldsymbol{x}_{min},\ m,\ M\right)$
while $SC == false$ **do**
  $m \leftarrow m + 1$
  $\alpha^{(m)} \leftarrow 1 - v\text{^}(-(m-1)/(M-1))$ // Eq.7
  $P^{(m)}_{max} = ceil\left(\alpha^{(m)} * N\right)$ // Eq.6
  $N^{(m)} = ceil\left(2 * N/P^{(m)}_{max}\right)$ // Eq.5
  $\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)}\right) \leftarrow Sort\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ N^{(m)}\right)$ // Subsec.II-B, Fig.4
  $P^{(m)} \leftarrow GetNS\left(P^{(m)}_{max},\ N^{(m)}\right)$ // Eq.4
  $\Delta\boldsymbol{x}^{(m)} = \left(1 - \alpha^{(m)}\right) * \Delta\boldsymbol{x}^{(1)}$ // Eq.8
  $\boldsymbol{Y}^{(m)} \leftarrow GenerateANP\left(\boldsymbol{X}^{(m)},\ P^{(m)},\ \Delta\boldsymbol{x}^{(m)}\right)$ // Subsec.II-C, Fig.6 or Fig.7
  $\boldsymbol{f}^{(m)}_{temp} \leftarrow f\left(\boldsymbol{Y}^{(m)}\right)$
  $\boldsymbol{X}^{(m)} \leftarrow \left[\boldsymbol{X}^{(m)},\ \boldsymbol{Y}^{(m)}\right]$
  $\boldsymbol{f}^{(m)} \leftarrow \left[\left(\boldsymbol{f}^{(m)}\right)^T,\ \left(\boldsymbol{f}^{(m)}_{temp}\right)^T\right]^T$
  $a^{(m)} \leftarrow \boldsymbol{f}\left(\boldsymbol{X}^{(m)}\left(N^{(m)}\right)\right)$ // Subsec.II-D
  $\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ \boldsymbol{x}^{*(m)},\ f^{*(m)}\right) \leftarrow CRP\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ a^{(m)}\right)$
  $SC \leftarrow CheckSC\left(\Delta\boldsymbol{x}^{(m)},\ \Delta\boldsymbol{x}_{min},\ m,\ M\right)$
end

each pattern corresponds to an artificial chromosome and the pattern matrix represents a population. Many metaheuristic algorithms somehow use the basic genetic rules (i.e., mutation, crossover, selection, and adaptation) while improving the pattern matrix. However, its attribute variety decreases through iterations. The main question is how to generate new patterns or attributes to the pattern matrix to maintain diversity [7].

In WFS algorithm, the pattern matrix $\boldsymbol{W}^{(m)}$ is referred to as all points that flier sees in a single iteration (all points from the gray line in Fig. 8). WFS does not use mutation or crossover directly, but it develops new solution by generating corresponding neighborhoods around potential solutions

Algorithm 2 Function *CRP*

Input: $\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ a^{(m)}$
Output: $\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ \boldsymbol{x}^{*(m)},\ f^{*(m)}$
$\left(\boldsymbol{x}^{*(m)},\ f^{*(m)}\right) \leftarrow min\left(\boldsymbol{f}^{(m)}\right)$
$\boldsymbol{Z}^{(m)} \leftarrow GenerateCRP\left(\boldsymbol{X}^{(m)},\ \boldsymbol{f}^{(m)},\ f^{*(m)},\ a^{(m)}\right)$ // Subsec.II-D
$\boldsymbol{f}^{(m)}_{CRP} \leftarrow f\left(\boldsymbol{Z}^{(m)}\right)$
$\boldsymbol{X}^{(m)} \leftarrow \left[\boldsymbol{X}^{(m)},\ \boldsymbol{Z}^{(m)}\right]$
$\boldsymbol{f}^{(m)} \leftarrow \left[\left(\boldsymbol{f}^{(m)}\right)^T,\ \left(\boldsymbol{f}^{(m)}_{CRP}\right)^T\right]^T$
$\left(\boldsymbol{x}^{*(m)},\ f^{*(m)}\right) \leftarrow UpdateMin\left(\boldsymbol{Z}^{(m)},\ \boldsymbol{f}^{(m)}_{CRP},\ f^{*(m)}\right)$

(see Subsection II-C). In order to maintain diversity of points, the combination of Halton sequence and grid logic is used (see Subsection II-A). Selection is used when best $N^{(m)}$ points are selected for next iteration (Equation (5)). Adaptation is used when adapting the discretization step (Equation (8)). Beside pseudo-randomly located points in specified neighborhood, there are random point and centroid (see Subsection II-D), added at each iteration, which contribute to maintaining diversity of the pattern matrix.

The general *random-walk* based system-equation of metaheuristic algorithms is given by

$$v \leftarrow X_i + s\delta X, \tag{12}$$

where δX ($\in \mathbb{R}^n$) denotes the step-size and X_i ($\in \mathbb{R}^n$) subtends to a random solution of the related objective function. The scaling factor s ($\in \mathbb{R}$) is selected for δx values [28]. To develop the X_i pattern, the decision rule

$$f(v) < f(X_i) \rightarrow X_i := v \tag{13}$$

is used where f is the objective function. Here, $X_i = [x_1\ x_2\ \cdots\ x_n]^T$ vector shows the i-th pattern of the pattern matrix $P = [X_1\ X_2\ \cdots\ X_m]$, which includes n-dimensional m random solutions [48]. The metaheuristic algorithms differ radically in terms of the methods they use to determine s and δx values [7].

In WFS, clearly, $\delta X = \Delta\boldsymbol{x}^{(1)}$ and $s = (1 - \alpha^{(m)})$ according to (8). Equation (12) is always used when generating neighborhood around each X_i ($= \boldsymbol{x}_i^{(m)}$). Therefore, WFS is suitable for parallelized computation since it can optimize different parts of the search space independently, which differs it from most concurrent algorithms. One more advantage that should be highlighted, is that WFS is able to rapidly converge to the global optimum solution of the related objective function. That convergence is, theoretically, exponential due to the exponential dependence in (7). Thus, combining "parallelized computation" and rapidly convergence helps this algorithm to make a good balance between exploration and exploitation of the search space. The mentioned balance turns out to be the inherent feature of the algorithm due to the fact that WFS is essentially parameter-free. Experimental results in the following section will confirm this claim.

IV. COMPARISON BASED ON EXPERIMENTAL RESULTS

In this subsection, WFS will be compared with 11 similar classical global optimization algorithms, such as: Genetic Algorithm (GA) [34], Particle Swarm Optimization (PSO) [6], Bat Algorithm (BA) [58], Grey Wolf Optimizer (GWO) [33], Butterfly Optimization Algorithm (BOA) [1], Whale Optimization Algorithm (WOA) [32], Moth Flame Optimization (MFO) [31], Harris Hawks Optimization (HHO) [14], Monarch Butterfly Optimization (MBO) [53], Moth Search (MS) [51], Elephant Herding Optimization (EHO) [52], LSHADE-cnEpSin (later denoted as A1) [3], LSHADE-SPACMA (later denoted as A2) [35] and EBOwithCMAR (later denoted as A3) [22]. It is worth pointing out that last three algorithms performed the best at the recent competition based on CEC 2017 benchmark for global optimization [37]. The choice of algorithms included in the computational study is mainly driven by the availability of code, whether the code was accessible online, or was it provided to us upon request by the courtesy of authors. If the code was not available, we would abstain from implementing the corresponding algorithm ourselves. We maintain that, by taking this approach, we avoid the risk of non-optimal implementation and hence unfair comparison. On the other hand, a limited set of algorithms was taken into account due to space constraints.

Before comparison, we should determine the default value for flier's velocity v. After testing WFS on classical functions for a wide range of v (even up to 16000), it turned out that the best results are obtained for $v \in (10, 100)$. Once we constrain the value of v to belong to the interval (10, 100), it turns out that the algorithm becomes rather insensitive to it, at least for majority of test functions. Moreover, v may be set randomly within that interval (see Algorithm 1, Line 2). Also, better results are obtained when using Halton versus grid points, as it is stated in Subsection II-E, so Halton points are used as default.

For the experimental evaluation of selected algorithms, recommendations from CEC 2020 [25] are used. The underlying assumptions ensure conditions for efficient and fair comparison of competing algorithms. The most relevant provisions are the following:

- Default parameters are used;
- Total independent runs[2] for each test function is 30;
- Total number of cost function evaluations is $1000 \cdot n \cdot M$, where $M = 10$ is the number of iterations;
- We used 30 well-known classic benchmark functions [17] and 10 modern numerical optimization problems considered from CEC 2020 special session and competition on single objective bound constrained numerical optimization [25]. Each test function is detrended in order to impose that the real solution value is always $f(\boldsymbol{x}^*) = 0$. Thus, it is convenient to present all results using logarithmic scale;
- Each solution value less than 10^{-8} is treated as zero;
- Several performance indicators for solution values are used: best, worst, mean and standard deviation (Std). The algorithm mean runtime is also considered;
- Testing[3] is performed using MATLAB R2018a at PC with the processor *Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz 1.99 GHz* and *8.00 GB DDR4 RAM*.

In the sequel, the test of reliability, efficiency and validation of the novel algorithm is provided by using a chosen set of common standard benchmarks test functions from the literature [17]. The number of test functions in most papers varies from a few to about two dozens. Here, we chose the test suite to be large enough to include a wide variety of problems, such as unimodal, multimodal, separable, non-separable and multi-dimensional problems. Unimodality confirms the exploitation capability of proposed algorithm, while multimodality checks the exploration capabilities. Also, experimental study on the modern numerical optimization problems considered from CEC 2020 is conducted. These benchmark functions set the bar higher for the optimization algorithms by the fact that the functions are shifted, rotated, expanded, and combined variants of the most complicated mathematical optimization problems.

All results are displayed using tables and boxplots. The five best algorithms in each row of a table are marked green, while the five worst ones are marked red, which makes easier to compare results visually. When boxplot for some algorithm is not shown, it is because all function values are less than 10^{-8}, which are treated as zero. Also, logarithmic scale is used to get a better view on small and huge values at the same plot. However, this scale looks confusing when we want to compare algorithms w.r.t. their standard deviation, since the distance between 10^k and 10^{k+1} looks the same as between $10^{-(k+1)}$ and 10^{-k}, $k \in \mathbb{N}_0$.

A. UNIMODAL AND SEPARABLE BENCHMARK FUNCTIONS

This test evaluates the effectiveness and accuracy of WFS while solving benchmark functions with unimodal and inherently separable characteristics. The results obtained from each algorithm are showed in Tab. 1, while Fig. 10 provides corresponding boxplots. Clearly, BOA, HHO, MS, A1, A3 and WFS show best performance. For example, they provide all values less than 10^{-8} for Quartic function. Convergence rate comparison in Fig. 11 reveals that WFS finds the best solution among the other algorithms even in the first iteration. We assume the main reason for this is centroid point which lies near the global optimum in the first iteration, which is, for Sphere, Sum Squares and Quartic function, in the middle of the search space. However, WFS did not find the solution for Step function immediately, yet it converged to it

[2]It should be noted that MBO, EHO and MS were tested only 10 times, due to its long computational time (see Tab. 8), so obtained results for them may not be very reliable.

[3]All source-codes are available online at https://github.com/ncovic1/Wingsuit-Flying-Search.git

TABLE 1. Comparative results for unimodal and separable classical benchmark functions.

Function		WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
Step $n=30$	Best	2.3693e-02	5.2060e+00	2.2733e+01	1.6662e+01	2.3954e-02	1.3263e+00	3.7357e-02	5.6347e+01	1.2571e-06	8.9427e-01	2.2871e+00	5.4514e+00	8.4729e-07	1.5427e-01	0
	Worst	3.9183e-01	1.1394e+01	4.5374e+01	3.9842e+01	3.2118e-01	2.7740e+00	3.7663e-01	1.1105e+02	7.5763e-05	2.3545e+01	3.7257e+00	6.4346e+00	7.0801e-06	8.4905e-01	0
	Mean	2.1024e-01	8.1736e+00	3.1656e+01	2.6995e+01	1.3595e-01	1.9048e+00	1.6319e-01	8.9765e+01	2.2245e-05	9.8595e+00	3.0179e+00	6.0604e+00	2.3186e-06	4.0864e-01	0
	Std	2.2634e-01	8.3041e+00	3.2135e+01	2.7568e+01	1.6271e-01	1.9305e+00	1.8054e-01	9.0534e+01	3.0252e-05	1.3923e+01	3.0745e+00	6.0760e+00	2.5666e-06	4.3249e-01	0
Sphere $n=30$	Best	1.4600e-03	3.4682e+00	7.4837e+03	6.0434e+03	1.4302e+01	1.9970e-01	7.3176e+00	2.6681e+04	0	3.5227e+01	0	6.6624e-01	1.2649e-04	1.6484e-01	0
	Worst	5.2211e-03	2.0228e+01	1.7066e+04	1.2414e+04	1.0987e+02	1.1961e+01	8.6655e+01	4.0210e+04	1.0632e-07	6.4927e+03	0	1.6547e+00	8.2740e-04	7.0701e-01	0
	Mean	3.1605e-03	9.1418e+00	1.2548e+04	9.2308e+03	4.4665e+01	4.3896e+00	3.3218e+01	3.4898e+04	1.0515e-08	2.8268e+03	0	1.1016e+00	3.7081e-04	3.8714e-01	0
	Std	3.2969e-03	9.9161e+00	1.2780e+04	9.3719e+03	4.9714e+01	5.4041e+00	3.7943e+01	3.5037e+04	2.7140e-08	3.9144e+03	0	1.1761e+00	4.1322e-04	4.0865e-01	0
Sum Squares $n=30$	Best	3.2035e-04	7.3605e+01	1.1892e+03	7.6425e+02	2.5424e+00	1.2983e-08	1.5790e+00	2.9994e+03	0	1.2102e+03	0	2.1896e-01	1.7841e-05	2.4626e+00	0
	Worst	1.3067e-03	2.1107e+02	2.6569e+03	1.9994e+03	1.2255e+01	5.2275e-06	1.5791e+01	5.4136e+03	1.2426e-07	3.4323e+03	7.1593e-08	2.9105e-01	9.7383e-05	1.2174e+01	0
	Mean	9.0338e-04	1.1284e+02	1.7870e+03	1.3243e+03	6.9768e+00	9.3472e-07	5.1162e+00	4.5300e+03	6.3517e-09	2.3166e+03	4.3458e-08	2.5060e-01	5.1125e-05	6.3307e+00	0
	Std	9.3324e-04	1.1770e+02	1.8219e+03	1.3575e+03	7.4507e+00	1.4665e-06	6.1106e+00	4.5714e+03	2.3515e-08	2.4879e+03	5.3481e-08	2.5240e-01	5.5078e-05	6.6716e+00	0
Quartic $n=30$	Best	0	4.5170e-01	2.7964e+00	1.6058e+00	8.6474e-05	0	9.8305e-06	2.2486e+01	0	4.4370e+00	0	1.1731e-06	0	2.1045e-01	0
	Worst	0	1.6231e+00	1.3932e+01	1.0425e+01	2.5928e-03	0	5.3308e-04	4.8869e+01	0	3.3159e+01	0	2.1558e-06	0	4.9741e+00	0
	Mean	0	1.0093e+00	6.4564e+00	5.1253e+00	3.6069e-04	0	1.2641e-04	3.5357e+01	0	1.8165e+01	0	1.5667e-06	0	1.2515e+00	0
	Std	0	1.0461e+00	7.0267e+00	5.5291e+00	5.9118e-04	0	1.8294e-04	3.6005e+01	0	2.1639e+01	0	1.6231e-06	0	1.5455e+00	0

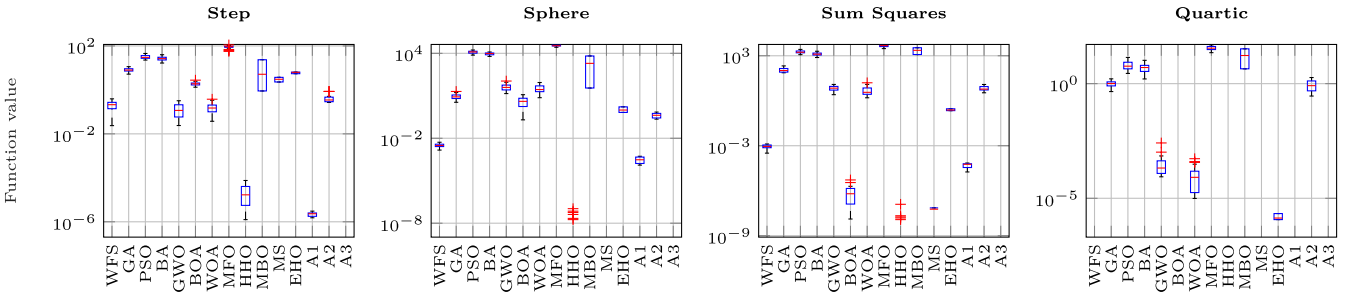

FIGURE 10. Boxplots for unimodal and separable classical benchmark functions.

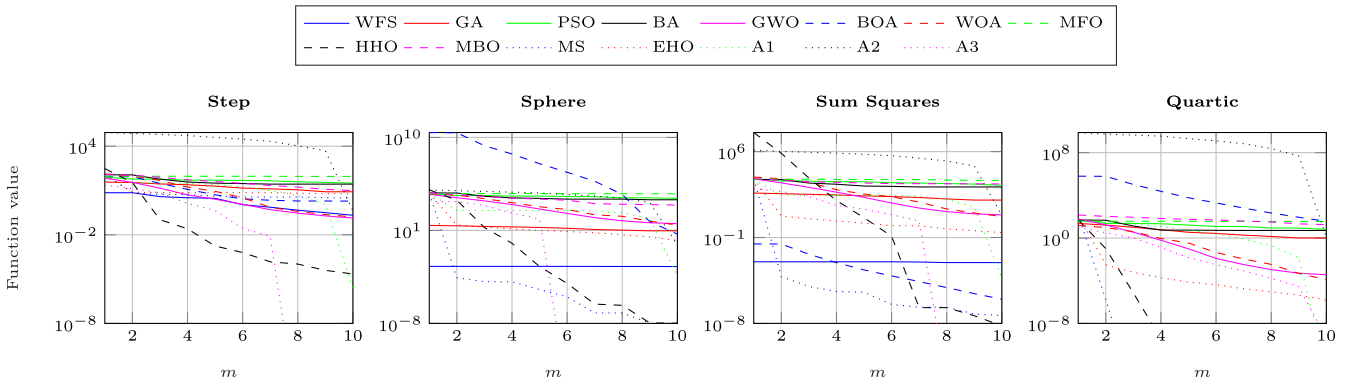

FIGURE 11. Convergence rate comparison for unimodal and separable classical benchmark functions.

very fast. In this case, centroid is not very helpful, but it is not indispensable.

B. UNIMODAL AND NON-SEPARABLE BENCHMARK FUNCTIONS

This test evaluates the performance and consistency of WFS while solving the unimodal but non-separable functions. The results obtained from each algorithm are showed in Tab. 2, while Fig. 12 provides corresponding boxplots. GWO, HHO, MS, A2, A3 and WFS show very good performance, while BOA, WOA and EHO are little bit worse. Convergence rate comparison in Fig. 13 shows that WFS converges very fast for functions containing non-centered global minimum (e.g., Beale, Easom, Collvile function).

C. MULTIMODAL AND SEPARABLE BENCHMARK FUNCTIONS

The purpose of this test is to check the exploration capability of proposed WFS while solving the multimodal and separable functions. The results obtained from each algorithm are showed in Tab. 3, while Fig. 14 provides corresponding boxplots. Again, GWO, WOA, HHO, A2, A3 and WFS have very good performance, but GA is the best for Michalewicz 5 and Michalewicz 10 function, and HHO and MS for Rastrigin function. Fig. 15 reveals that GA converges the fastest for Michalewicz 5 function, HHO for Rastrigin function, but A3 and WFS for Booth, Holder Table and Michalewicz 2 function. It should be noted that sign "/" in table denotes that algorithm did not find feasible solution within the specified bounds.

D. MULTIMODAL AND NON-SEPARABLE BENCHMARK FUNCTIONS

This part of validation study also checks the exploration capability of proposed WFS while solving the multimodal as well as non-separable functions. The results obtained from each algorithm are showed in Tab. 4, while Fig. 16 provides corresponding boxplots. Unequivocally, HHO, A2, A3 and

TABLE 2. Comparative results for unimodal and non-separable classical benchmark functions.

Function		WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
	Best	0	5.3462e-07	1.1261e-06	1.3837e-06	2.2544e-07	1.6126e-04	0	1.0816e-06	0	4.6269e-05	6.0353e-05	1.0794e-05	0	0	0
Beale	Worst	8.4056e-08	1.3885e-03	3.8920e-04	2.0431e-02	2.9238e-05	2.7251e-02	0	1.1878e-03	0	4.0008e-03	7.2001e-04	2.3059e-02	0	1.4537e-05	0
$n = 2$	Mean	1.0760e-08	3.2324e-04	7.9630e-05	3.5967e-03	1.0580e-05	6.5534e-03	0	3.1507e-04	0	1.8924e-03	2.8967e-04	7.8737e-03	0	1.0072e-06	0
	Std	2.3458e-08	4.8480e-04	1.1753e-04	5.5754e-03	1.3387e-05	9.2376e-03	0	4.3368e-04	0	2.4944e-03	4.2029e-04	1.3317e-02	0	2.8981e-06	0
	Best	1.0055e-08	3.5755e-06	2.9717e-04	1.9202e-06	1.7648e-06	1.0000e+00	0	2.2357e-02	0	6.5338e-07	7.6566e-04	1.2271e-01	1.4293e-08	0	0
Easom	Worst	1.0974e-04	6.3422e-02	1.5790e-01	1.0000e+00	1.9397e-04	1.2381e-01	1.0000e+00	9.9234e-01	1.8743e-05	1.5842e-04	1.3233e-03	5.6506e-01	3.0034e-06	0	0
$n = 2$	Mean	8.7211e-06	1.2440e-02	3.3494e-02	3.5076e-01	8.3086e-05	8.4899e-01	3.3335e-02	2.8349e-01	1.1160e-06	5.3656e-05	1.0305e-03	3.7659e-01	1.0590e-06	0	0
	Std	2.1835e-05	2.1297e-02	4.8250e-02	5.0996e-01	1.0160e-04	8.7378e-01	1.8257e-01	3.8076e-01	3.5693e-06	9.1471e-05	1.0555e-03	4.2021e-01	1.4534e-06	0	0
	Best	0	9.3711e-07	4.2506e-07	7.0625e-07	0	7.8183e-05	0	4.8003e-06	0	1.2991e-04	0	4.2665e-07	0	0	0
Matyas	Worst	4.4962e-08	2.8697e-04	5.7686e-05	6.8408e-03	0	1.7958e-03	0	3.2586e-04	0	1.1365e-03	0	3.0216e-06	0	0	0
$n = 2$	Mean	2.4058e-09	6.4570e-05	1.7661e-05	1.0895e-03	0	4.8207e-04	0	8.9792e-05	0	5.0339e-04	0	1.9006e-06	0	0	0
	Std	9.5954e-09	1.0246e-04	2.4426e-05	2.0491e-03	0	6.4852e-04	0	1.1947e-04	0	6.7527e-04	0	2.1902e-06	0	0	0
	Best	4.7310e-05	9.4851e-02	8.3575e-01	9.7653e-01	3.6267e-03	6.4802e-01	2.8016e-03	1.4882e+00	1.6032e-06	1.8290e-01	1.8358e-01	1.1005e+00	1.8785e-01	0	0
Colville	Worst	1.5118e+00	1.2362e+00	7.2415e+00	5.5428e+01	2.7969e+00	9.0001e+00	4.3625e+01	2.0521e+01	7.7378e-01	7.6385e+00	2.8594e-01	3.1872e+00	5.2067e+00	0	0
$n = 4$	Mean	1.0146e-01	4.7723e-01	2.9116e+00	1.4096e+01	2.7011e-01	4.2239e+00	3.7103e+00	8.9667e+00	3.3601e-02	3.0809e+00	2.3474e-01	1.9075e+00	2.0054e+00	0	0
	Std	3.5951e-01	5.6620e-01	3.2792e+00	1.8007e+01	6.1913e-01	4.6763e+00	8.8176e+00	1.0400e+01	1.4247e-01	4.4870e+00	2.3843e-01	2.1157e+00	2.5316e+00	0	0
	Best	6.7678e-04	7.9291e-01	2.4113e+00	9.2728e+00	5.4023e-05	0	1.0526e+01	1.0970e+01	0	4.7800e-01	0	3.5529e-02	1.2862e-01	0	0
Zakharov	Worst	1.4617e-01	5.7194e+00	1.0211e+01	4.3148e+01	5.5410e-03	1.2275e-05	5.9161e+01	4.5312e+01	7.7288e-07	9.5774e-01	1.2772e-08	9.1441e-02	3.0244e-01	5.0591e-08	0
$n = 30$	Mean	2.2376e-02	3.3433e+00	4.7387e+00	2.0611e+01	1.1537e-03	2.7958e-06	3.0691e+01	3.1401e+01	7.9870e-08	7.2294e-01	4.2573e-09	5.9477e-02	1.9232e-01	1.3545e-08	0
	Std	3.8278e-02	3.5669e+00	5.0069e+00	2.1991e+01	1.8131e-03	4.1510e-06	3.2669e+01	3.2540e+01	1.8037e-07	7.4903e-01	7.3738e-09	6.3958e-02	2.0358e-01	2.1411e-08	0
	Best	7.8023e-03	8.2295e+00	4.3767e+01	3.5622e+01	2.1709e+00	2.6976e-04	5.2318e-01	6.8855e+01	4.3278e-07	3.0415e+00	3.4345e-05	4.7471e-01	9.7156e+01	6.3748e-03	8.7086e-06
Schwefel 2.22	Worst	2.0138e-02	1.5019e+01	1.6254e+02	1.3645e+02	4.9125e+00	5.4299e-03	3.0301e+00	3.3382e+04	2.7018e-04	5.9550e+01	4.6418e-04	6.2063e-01	2.9259e+09	2.7446e-02	4.3209e-05
$n = 30$	Mean	1.1734e-02	1.1341e+01	7.5498e+01	5.2709e+01	3.2268e+00	1.5978e-03	1.8350e+00	4.3452e+03	4.6961e-05	2.4629e+01	1.9119e-04	5.4136e-01	2.9469e+08	1.8277e-02	1.8359e-05
	Std	1.2136e-02	1.1448e+01	7.8479e+01	5.5848e+01	3.3025e+00	2.0033e-03	1.9252e+00	9.5402e+03	7.6431e-05	3.5039e+01	2.7220e-04	5.4470e-01	9.2528e+08	1.9223e-02	2.0921e-05
	Best	7.0528e-03	8.8121e+00	1.7746e+04	9.9563e+03	3.0459e+02	4.9823e-01	2.7923e+04	3.1622e+04	0	2.3906e+03	7.9744e-08	3.3732e+00	2.9868e+00	8.2154e-02	0
Schwefel 1.2	Worst	4.2571e-01	4.7425e+01	3.5674e+04	2.2749e+04	2.4854e+03	1.0128e+02	4.9936e+04	5.2105e+04	1.9345e-03	1.3694e+04	3.8195e-07	4.1735e+00	7.8777e+00	5.4539e-01	0
$n = 30$	Mean	1.3377e-01	2.2992e+01	2.6880e+04	1.5120e+04	1.2218e+03	4.2285e+01	3.9421e+04	4.2779e+04	9.3783e-05	7.2752e+03	2.7007e-07	3.8289e+00	4.8649e+00	3.0119e-01	0
	Std	1.7842e-01	2.5334e+01	2.7345e+04	1.5382e+04	1.3079e+03	4.9943e+01	3.9726e+04	4.3010e+04	3.6275e-04	8.6834e+03	3.0206e-07	3.8436e+00	5.0820e+00	3.3867e-01	0
	Best	1.0973e+00	1.2465e+02	3.1641e+04	1.3130e+04	2.5747e+00	1.0210e+00	2.8448e+00	4.1626e+05	8.6340e-02	1.3121e+04	6.6953e-01	1.1413e+00	8.2027e+00	6.6708e-01	6.6667e-01
Dixon-Price	Worst	1.1933e+00	1.6512e+03	2.1962e+05	9.6927e+04	3.9284e+01	5.0904e+05	3.0846e+01	6.5355e+05	2.5601e-01	9.3517e+04	7.0374e-01	1.3251e+00	2.4903e+01	6.6798e-01	6.6667e-01
$n = 30$	Mean	1.1508e+00	5.9354e+02	8.9820e+04	4.5729e+04	1.6921e+01	8.0248e+04	7.9665e+00	5.0923e+05	2.3488e-01	6.5825e+04	6.8665e-01	1.2118e+00	1.6447e+01	6.6734e-01	6.6667e-01
	Std	1.1510e+00	7.3183e+02	9.8618e+04	5.0423e+04	1.8671e+01	1.6959e+05	9.6747e+00	5.1308e+05	2.3745e-01	7.5650e+04	6.8679e-01	1.2145e+00	1.7292e+01	6.6734e-01	6.6667e-01

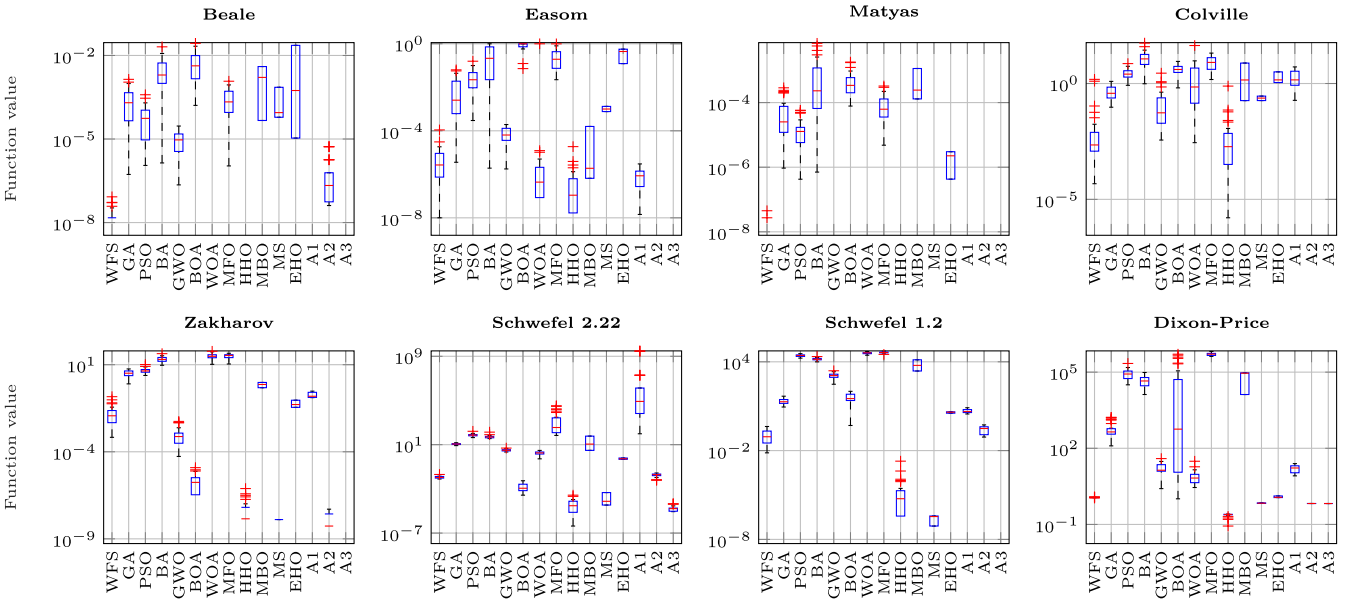

FIGURE 12. Boxplots for unimodal and non-separable classical benchmark functions.

WFS show the best performance for most functions. Nevertheless, GWO, WOA and MS show good performance as well. Moreover, WFS indicates very small standard deviation, which increases reliability of this algorithm. Its fast and early convergence (Fig. 17), especially for Schaffer 2 and 4 functions, Six Hump Camel Back and Shubert function, can be seen.

E. CEC 2020 BENCHMARK FUNCTIONS

This test serves to evaluate the effectiveness and robustness of the novel algorithm. Therefore, most intensely investigated benchmark functions used in CEC 2020 are considered for this purpose. They are designed with special features, which makes very hard to find the global optimum using any algorithm. Congress on Evolutionary Computation (CEC) represents a special session and competition on single and multi-objective real-parameter numerical optimization problems [25]. In this experimentation, all CEC 2020 functions are considered. The results obtained from each algorithm are showed in Tab. 5, while Fig. 18 provides corresponding boxplots. Clearly, GA, GWO, A1, A2, A3 and WFS provide results better than other algorithms. Convergence rate comparison is provided by Fig. 19. Again, mentioned algorithms provide the best convergence. Here, the results are a little bit worse to HHO, MBO and MS, than they were for classical functions.

F. SUMMARY OF RESULTS AND FINAL RANKING OF THE ALGORITHMS

Experiments on unimodal test functions from Subsections IV-A, IV-B and IV-E show that WFS has very good exploitation capabilities compared to the other related

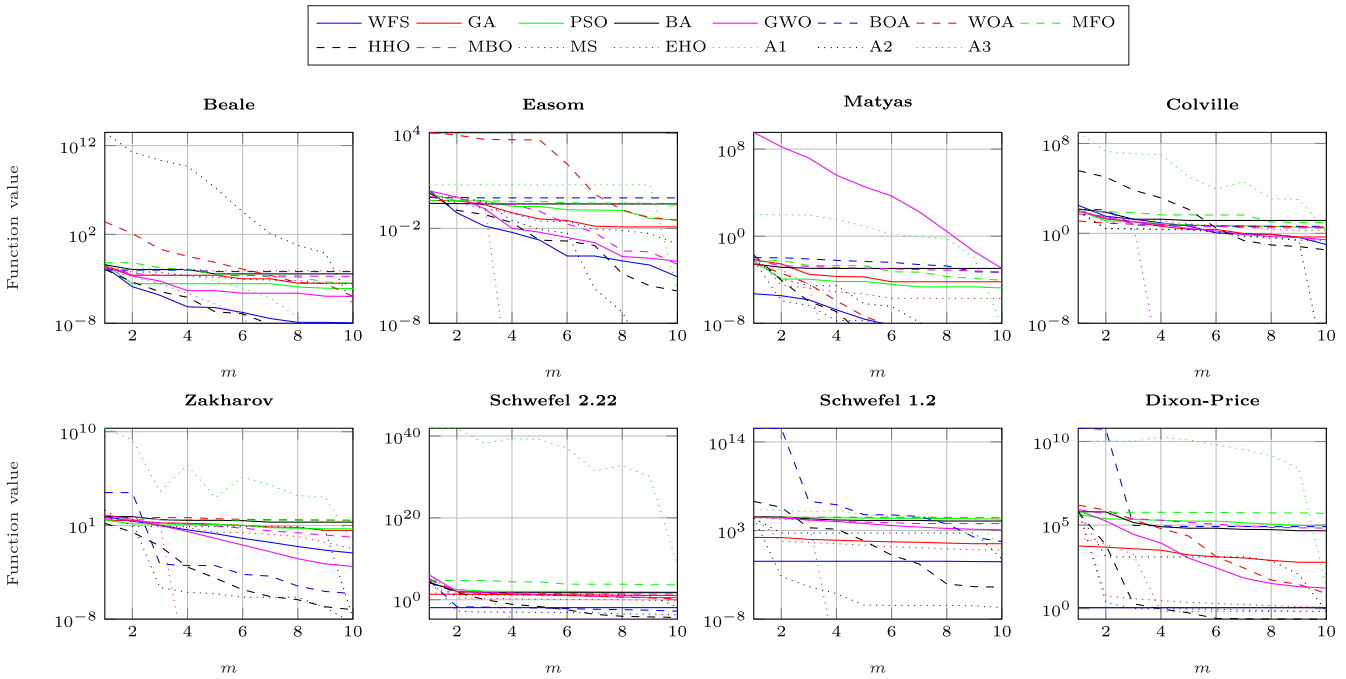

FIGURE 13. Convergence rate comparison for unimodal and non-separable classical benchmark functions.

TABLE 3. Comparative results for multimodal and separable classical benchmark functions.

Function		WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
	Best	1.3830e-08	4.0689e-04	2.8217e-03	3.3057e-04	0	8.6602e-07	0	3.8648e-02	0	3.7573e-06	2.7770e-08	5.4168e-03	0	0	0
Bohachevsky 1	Worst	2.1455e-03	1.1421e-01	4.8087e-01	8.9239e+00	0	3.0672e-04	1.9632e-06	9.5995e-01	2.7428e-08	1.8683e-04	2.4535e-07	1.5336e-02	3.2263e-07	0	0
$n = 2$	Mean	1.3208e-04	2.6248e-02	2.4523e-01	1.2721e+00	0	6.4158e-05	3.5940e-07	4.5183e-01	3.2519e-09	6.5780e-05	1.7108e-07	9.9866e-03	6.5444e-08	0	0
	Std	4.1585e-04	3.6612e-02	2.9499e-01	2.0725e+00	0	1.0429e-04	6.1467e-07	5.1635e-01	8.4723e-09	1.0796e-04	1.9885e-07	1.0790e-02	1.1633e-07	0	0
	Best	0	3.1038e-06	6.5130e-06	3.6300e-05	1.1122e-06	8.7064e-04	2.5179e-07	1.2347e-05	0	3.1066e-04	8.7302e-06	3.7698e-05	0	0	0
Booth	Worst	1.2602e-06	8.5290e-03	1.4756e-03	2.4762e-01	1.0339e-04	1.3679e-01	7.4313e-03	1.0510e-02	4.0475e-04	1.1972e-02	4.1011e-04	3.7285e-02	1.6700e-07	0	0
$n = 2$	Mean	1.3871e-07	1.6847e-03	5.2372e-04	4.6947e-02	2.2055e-05	2.4373e-02	1.0653e-03	2.1939e-03	9.1073e-05	4.6628e-03	2.4380e-04	2.0335e-02	2.3524e-08	0	0
	Std	2.9488e-07	2.7758e-03	6.7630e-04	7.3856e-02	3.2328e-05	3.7160e-02	2.1217e-03	3.2506e-03	1.4473e-04	6.9844e-03	2.9775e-04	2.5501e-02	5.5994e-08	0	0
	Best	0	3.0809e-06	3.6504e-05	4.0428e-04	7.1114e-05	1.0049e-03	0	2.3750e-05	0	/	/	/	/	5.2523e-06	0
Holder Table	Worst	2.6574e-06	4.5350e-03	4.7936e-03	1.7438e-01	1.2861e-01	5.2930e-01	1.1205e-06	3.3476e-02	1.3447e-05	/	/	/	/	3.4426e-04	0
$n = 2$	Mean	3.5604e-07	4.5838e-04	1.0688e-03	3.9588e-02	1.0655e-02	9.2385e-02	1.0321e-07	9.1253e-03	1.3679e-06	/	/	/	/	1.0133e-04	0
	Std	6.7234e-07	9.5550e-04	1.5166e-03	6.0668e-02	2.9580e-02	1.4325e-01	2.4358e-07	1.2545e-02	3.4280e-06	/	/	/	/	1.4594e-04	0
	Best	0	8.0224e-07	1.6499e-05	1.9171e-05	1.1178e-06	6.0484e-04	0	1.7083e-05	0	5.5791e-05	1.1628e-03	6.5241e-03	/	0	0
Michalewicz 2	Worst	2.0758e-07	6.0721e-04	1.3603e-03	6.7401e-02	9.1258e-04	4.6580e-02	1.5116e-06	1.5018e-03	2.3298e-06	3.7739e-04	7.4255e-03	2.9776e-02	/	0	0
$n = 2$	Mean	1.8891e-08	9.4246e-05	2.6489e-04	1.0882e-02	2.2992e-04	9.3174e-03	9.0747e-08	3.2623e-04	2.8756e-07	2.1168e-04	3.3036e-03	1.7412e-02	/	0	0
	Std	4.9098e-08	1.7277e-04	4.0257e-04	1.9549e-02	3.1115e-04	1.3219e-02	2.8772e-07	4.8296e-04	5.4573e-07	2.4918e-04	4.4060e-03	1.9859e-02	/	0	0
	Best	7.1731e-05	5.5216e-04	7.6580e-02	5.2495e-01	9.7115e-02	8.1093e-01	1.5176e-03	3.7387e-01	1.5310e-02	5.6139e-03	2.3641e-01	8.9915e-01	1.7969e-02	2.5830e-06	0
Michalewicz 5	Worst	1.1265e+00	1.0607e-02	8.8376e-01	1.7595e+00	8.5048e-01	1.4614e+00	1.1951e+00	1.1379e+00	1.1934e+00	7.7796e-02	1.2728e+00	1.2325e+00	1.3013e+00	1.4318e-04	4.4443e-02
$n = 5$	Mean	2.4939e-01	4.2097e-03	4.1602e-01	1.1610e+00	2.3414e-01	1.2276e+00	5.0100e-01	8.1446e-01	4.7076e-01	3.1694e-02	8.5042e-01	1.1054e+00	8.2048e-01	3.9576e-05	8.7076e-03
	Std	3.4322e-01	5.0813e-03	4.5587e-01	1.2078e+00	2.7229e-01	1.2381e+00	6.2164e-01	8.4271e-01	6.0930e-01	4.5534e-02	9.5948e-01	1.1151e+00	9.0312e-01	6.1444e-05	1.8345e-02
	Best	4.8624e-01	6.6048e-02	2.9911e+00	3.3837e+00	1.8510e+00	3.2473e+00	1.3342e+00	3.8319e+00	1.5612e+00	5.1203e-01	3.3836e+00	3.3737e+00	3.4590e+00	8.5962e-01	6.5351e-01
Michalewicz 10	Worst	3.3782e+00	3.3552e-01	4.5228e+00	5.1650e+00	3.9905e+00	4.9723e+00	4.0021e+00	4.7902e+00	3.8116e+00	8.8429e-01	4.8477e+00	4.2237e+00	4.6333e+00	1.2820e+00	1.4240e+00
$n = 10$	Mean	1.9730e+00	1.6357e-01	3.7551e+00	4.4693e+00	2.8185e+00	4.2772e+00	2.7523e+00	4.3517e+00	2.8590e+00	7.2955e-01	4.1574e+00	3.9340e+00	4.1188e+00	1.0784e+00	1.0886e+00
	Std	2.1249e+00	1.7599e-01	3.7783e+00	4.4846e+00	2.8531e+00	4.2966e+00	2.8235e+00	4.3586e+00	2.9053e+00	7.4653e-01	4.2005e+00	3.9539e+00	4.1391e+00	1.0887e+00	1.1249e+00
	Best	2.5493e-03	5.3662e+01	2.3832e+02	2.2025e+02	5.7651e+01	2.6792e-08	6.7371e-01	2.9610e+02	0	1.6779e+01	0	8.3244e-01	1.6664e+02	9.1711e+01	1.4924e+01
Rastrigin	Worst	7.9192e-03	8.7133e+01	3.1125e+02	2.9265e+02	1.7692e+02	3.9177e-05	9.1800e+01	3.4349e+02	3.2887e-07	1.8566e+02	2.1764e-07	1.1563e+00	2.1747e+02	1.1040e+02	7.2632e+01
$n = 30$	Mean	4.3160e-03	6.8608e+01	2.7196e+02	2.5510e+02	8.5071e+01	5.1376e-06	2.5645e+01	3.1975e+02	2.1292e-08	9.5345e+01	1.1514e-07	1.0159e+00	1.9365e+02	1.0418e+02	3.4625e+01
	Std	4.5060e-03	6.8948e+01	2.7267e+02	2.5593e+02	8.8500e+01	9.2523e-06	3.7518e+01	3.2000e+02	6.4811e-08	1.1795e+02	1.4571e-07	1.0249e+00	1.9421e+02	1.0435e+02	3.9053e+01

algorithms. This is likely due to discretization step getting smaller through iterations. Thus, the current solution becomes more and more accurate.

Experiments on multimodal test functions from Subsections IV-C, IV-D and IV-E show that WFS has very good exploration capabilities compared to other related methods. We attribute this to the uniformity of initial population and properly controlled neighborhood sizes for each point. Thus, at the beginning, more solutions are being considered, which ensures the search to be decentralized.

In spite of discretization, uniformity of initial population and properly controlled neighborhood sizes, centroid of population represents another crucial mechanism (see Subsection II-D). Indeed, convergence plots indicate the occurrence of the situation when the centroid is positioned near global minimum (e.g., Rastrigin function in Fig. 15) in the first iteration steps. Usually, this occurs when global minimum is located more in the search space interior (global minimum is centered). Moreover, it is shown that even if the global minimum is not centered (e.g., Step function in Fig. 11), the algorithm successfully converges to it, proving that the centroid is not indispensable, but it is useful.

Final ranking of tested algorithms is performed by using obtained results from Tables 1 to 5. Four solution properties are analyzed separately: best, worst, mean and standard deviation. In each category, all algorithms are sorted in ascending order for each test function. Then, these rankings are summed up separately. It should be noted that, if two algorithms give

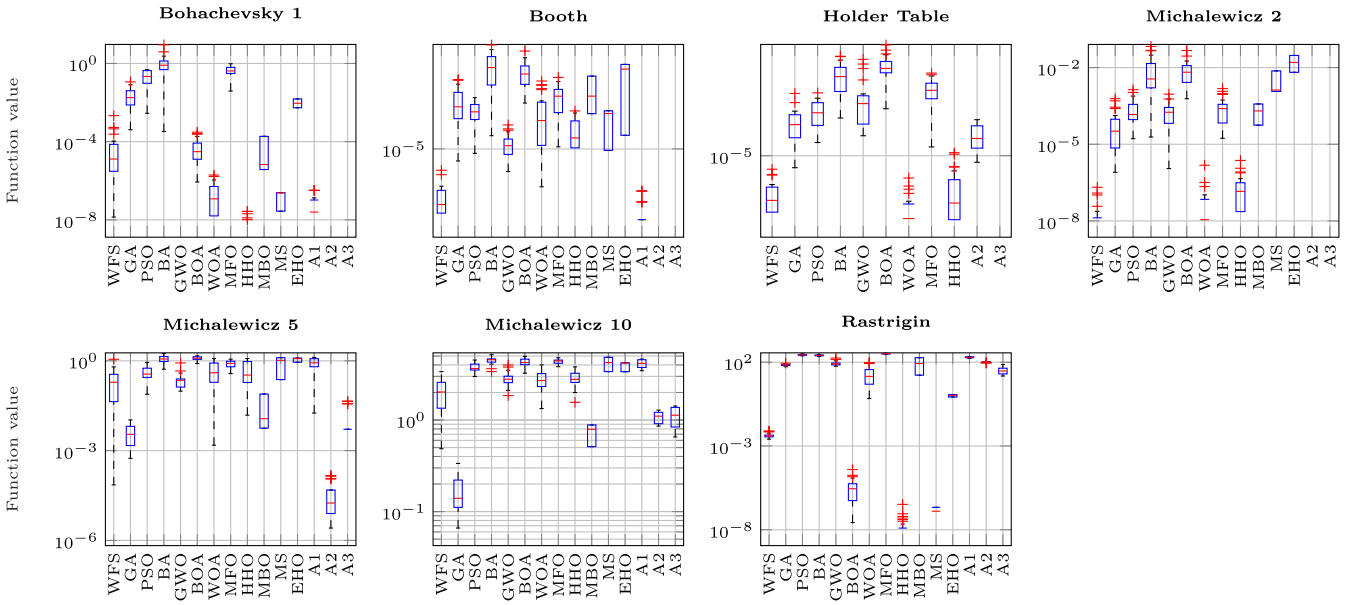

FIGURE 14. Boxplots for multimodal and separable classical benchmark functions.

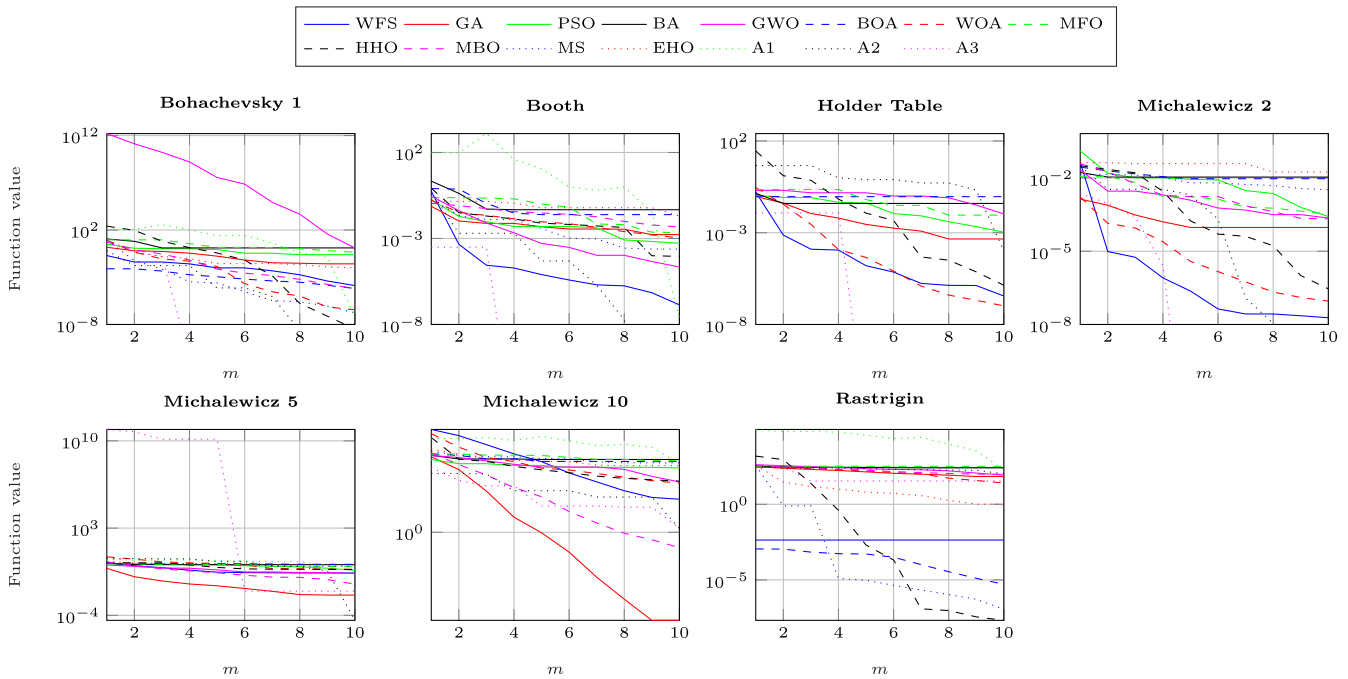

FIGURE 15. Convergence rate comparison for multimodal and separable classical benchmark functions.

the same mean value (e.g., zero), they both get the same ranking (1.5, not 1 nor 2). Clearly, the lower sum implies the better ranking of the algorithm.

The final ranking results for classical benchmark functions are given in Tab. 6, and for CEC 2020 benchmark functions in Tab. 7. In total, WFS clearly provides considerable performance improvements according to all four categories, and for both classical and CEC 2020 benchmark functions. Not surprisingly, HHO, MS, A2 and A3 show excellent performance, while PSO, BA, BOA, MFO and MBO do not appear to be very reliable, at least for the used implementation. Clearly, A3 is the winner in each category, while WFS took 3rd, 5th, 4th and 5th place w.r.t. best, worst, mean and standard deviation values, respectively, for classical benchmark functions. Final ranking for CEC 2020 benchmark functions reveals that WFS took 2nd, 6th, 3rd and 3rd place w.r.t. best, worst, mean and standard deviation values, respectively. It is worth to notice that WFS is the only algorithm, beside A2 and A3, which is

TABLE 4. Comparative results for multimodal and non-separable classical benchmark functions.

Function		WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
Schaffer 2 $n=2$	Best	0	1.7218e-07	5.7148e-07	1.9729e-08	0	1.9278e-06	0	5.6641e-06	0	2.9879e-05	0	3.3372e-07	0	0	0
	Worst	4.7316e-08	1.5417e-04	1.0443e-03	6.2180e-02	0	1.7059e-02	2.8592e-03	1.2834e-02	0	6.3296e-04	0	6.6225e-07	4.3681e-07	0	0
	Mean	2.9690e-09	1.6346e-05	1.5125e-04	1.4381e-02	0	5.0997e-03	3.9383e-04	3.3032e-03	0	2.5242e-04	0	4.8956e-07	1.5956e-07	0	0
	Std	1.0489e-08	3.8371e-05	2.8899e-04	1.9892e-02	0	7.1724e-03	8.8590e-04	4.8687e-03	0	3.6989e-04	0	5.0774e-07	2.2716e-07	0	0
Schaffer 4 $n=2$	Best	0	4.1375e-06	4.4491e-06	2.0427e-05	0	1.0715e-05	0	5.6338e-05	0	3.0856e-05	5.7681e-08	1.2763e-06	5.1282e-05	1.0274e-08	0
	Worst	2.1692e-06	2.3539e-03	1.5046e-03	2.6779e-02	2.1634e-05	8.1478e-03	1.9804e-04	7.8444e-03	1.1243e-05	6.3039e-04	7.4107e-06	1.5039e-04	3.5134e-04	2.4484e-05	0
	Mean	2.2817e-07	5.0213e-04	3.8696e-04	4.3025e-03	2.6505e-06	3.2234e-03	1.6758e-05	2.7407e-03	1.2249e-06	3.2031e-04	2.6159e-06	5.2963e-05	1.3347e-04	6.0730e-06	0
	Std	5.1978e-07	8.0583e-04	5.4172e-04	6.9708e-03	5.5321e-06	3.7913e-03	4.1300e-05	3.4188e-03	3.2283e-06	4.0338e-04	4.2843e-06	8.6933e-05	1.6208e-04	9.2493e-06	0
Schaffer 6 $n=30$	Best	1.1911e-01	5.7267e+00	1.1875e+01	1.1825e+01	8.1085e+00	4.4195e+00	2.7057e+00	1.2240e+01	0	9.5058e+00	0	5.1676e+00	1.0340e+01	6.6860e+00	5.5068e+00
	Worst	3.2995e-01	9.0006e+00	1.2474e+01	1.2788e+01	1.0779e+01	7.7095e+00	9.8607e+00	1.2709e+01	2.3435e-03	9.6152e+00	2.1867e-08	6.1291e+00	1.1104e+01	9.0751e+00	7.5258e+00
	Mean	1.9839e-01	7.9892e+00	1.2135e+01	1.2236e+01	9.5109e+00	6.7376e+00	7.3741e+00	1.2476e+01	9.0257e-05	9.5659e+00	7.2890e-09	5.7346e+00	1.0743e+01	8.1041e+00	6.6183e+00
	Std	2.0753e-01	8.0124e+00	1.2137e+01	1.2239e+01	9.5319e+00	6.7772e+00	7.5979e+00	1.2477e+01	4.2948e-04	9.5660e+00	1.2625e-08	5.7493e+00	1.0746e+01	8.1301e+00	6.6505e+00
Six Hump Camel Back $n=2$	Best	0	8.1926e-07	3.9862e-05	8.9940e-06	1.7951e-07	1.1827e-04	0	5.4937e-05	0	1.4598e-05	8.6281e-07	1.6942e-04	1.2934e-07	0	0
	Worst	5.1607e-07	1.2966e-03	1.3165e-03	1.1443e-01	5.5372e-05	9.5897e-03	7.1898e-08	2.9736e-03	1.7095e-08	4.7720e-05	3.4120e-05	1.8632e-03	5.0828e-06	1.0924e-07	0
	Mean	4.4797e-08	2.5099e-04	2.9880e-04	1.8623e-02	2.9512e-06	1.6418e-03	8.1946e-09	5.4505e-04	2.2354e-09	3.1199e-05	1.6136e-05	8.2003e-04	1.9771e-06	3.2098e-08	0
	Std	1.0897e-07	3.7271e-04	4.2670e-04	3.0842e-02	1.0216e-05	2.5640e-03	1.8370e-08	7.7740e-04	5.5347e-09	3.4003e-05	2.1175e-05	1.1080e-03	2.6901e-06	5.3243e-08	0
Bohachevsky 2 $n=2$	Best	1.4408e-07	6.8526e-04	3.3742e-03	1.8049e-03	0	8.9857e-07	0	6.1017e-05	0	9.3525e-05	1.1541e-08	5.6531e-03	0	0	0
	Worst	2.9620e-04	2.1917e-01	3.2086e-01	3.3271e+00	0	4.3176e-04	2.1831e-01	5.1135e-01	3.3418e-07	2.8590e-02	2.0697e-07	2.7724e-02	3.3493e-07	0	0
	Mean	4.6714e-05	4.8846e-02	1.6507e-01	6.7652e-01	0	1.0312e-04	4.3663e-02	2.5167e-01	2.4714e-08	1.4821e-02	1.1649e-07	1.8948e-02	7.2398e-08	0	0
	Std	7.6108e-05	8.1123e-02	1.9609e-01	1.0286e+00	0	1.6055e-04	9.7633e-02	2.7671e-01	7.6834e-08	1.8854e-02	1.4157e-07	2.1223e-02	1.2605e-07	0	0
Bohachevsky 3 $n=2$	Best	5.8869e-08	3.3609e-05	5.7154e-03	4.7279e-04	0	5.6787e-07	2.0612e-05	1.1717e-02	0	2.1597e-04	0	2.9714e-03	0	0	0
	Worst	1.3768e-03	9.3543e-02	2.3425e-01	9.5691e+00	0	1.8880e-04	1.3902e-01	3.2114e-01	8.9574e-06	4.4918e-03	1.3918e-08	9.5077e-03	1.1745e-07	0	0
	Mean	6.4320e-05	1.3488e-02	6.8200e-02	1.2503e+00	0	2.9953e-05	2.4811e-02	1.3856e-01	6.2223e-07	2.1506e-03	4.6393e-09	7.1769e-03	4.1812e-08	0	0
	Std	2.5342e-04	2.3179e-02	8.9600e-02	2.4953e+00	0	4.9179e-05	4.1073e-02	1.6814e-01	1.8863e-06	2.7847e-03	8.0355e-09	7.7708e-03	5.6327e-08	0	0
Shubert $n=2$	Best	0	1.2967e-05	1.3090e-02	1.1178e-03	1.1993e-04	1.7432e-02	0	1.1083e-02	0	6.1364e-05	1.1169e-02	2.2740e-01	7.5718e-03	0	0
	Worst	1.1957e-03	3.0021e-01	4.0030e+00	1.3970e+01	3.5252e+00	1.0934e+01	1.4307e-03	4.3459e+00	2.6527e-04	3.3472e-02	3.1695e-02	1.7948e+00	3.8796e-01	8.1904e-02	0
	Mean	1.1639e-04	6.0579e-02	5.4355e-01	2.1992e+00	4.3693e-01	1.8000e+00	6.7820e-05	8.8476e-01	2.4285e-05	2.2332e-02	2.1725e-02	1.2186e+00	1.0803e-01	2.8260e-02	0
	Std	2.5591e-04	9.5826e-02	1.0699e+00	3.5164e+00	9.0141e-01	2.7344e+00	2.6979e-04	1.3359e+00	6.5321e-05	2.7326e-02	2.3288e-02	1.4073e+00	1.5518e-01	3.9922e-02	0
Drop-Wave $n=2$	Best	0	2.0891e-06	5.1367e-06	1.8940e-06	0	1.0779e-05	0	1.8731e-04	0	1.2250e-07	0	3.8645e-06	7.0208e-05	0	0
	Worst	6.3755e-02	6.3755e-02	6.5554e-03	6.4131e-02	0	6.3759e-02	6.3755e-02	6.3755e-02	0	6.3758e-02	1.8020e-07	3.7541e-05	6.3756e-02	0	0
	Mean	1.2751e-02	1.0261e-02	1.4846e-03	5.5391e-02	0	1.3767e-02	1.9126e-02	1.5595e-02	0	4.2505e-02	6.5968e-08	2.5903e-05	1.2846e-02	0	0
	Std	2.8512e-02	2.1354e-02	2.2862e-03	5.9290e-02	0	2.3911e-02	3.4920e-02	2.2348e-02	0	5.2058e-02	1.0454e-07	3.0233e-05	2.5529e-02	0	0
Rosenbrock $n=30$	Best	2.9827e+01	8.6213e+02	4.9994e+06	2.8568e+06	3.1399e+02	3.9788e+01	1.1142e+02	5.0266e+07	1.0802e-03	3.6561e+05	2.8735e+01	3.7467e+01	1.0474e+02	2.7118e+01	0
	Worst	3.0734e+01	2.5798e+03	2.5498e+07	1.2094e+07	2.5802e+03	8.8006e+07	1.5331e+03	9.8371e+07	2.5078e-01	2.3960e+07	2.8945e+01	3.9593e+01	3.1443e+02	2.7934e+01	3.9866e+00
	Mean	3.0206e+01	1.4525e+03	1.2057e+07	6.5950e+06	1.4739e+03	5.3420e+07	4.9004e+02	7.6706e+07	4.5239e-02	9.8812e+06	2.8811e+01	3.8669e+01	1.8060e+02	2.7632e+01	1.1960e+00
	Std	3.0207e+01	1.5197e+03	1.3180e+07	6.9570e+06	1.5947e+03	6.2733e+07	5.9870e+02	7.7551e+07	7.1023e-02	1.4172e+07	2.8811e+01	3.8679e+01	1.8969e+02	2.7634e+01	2.1836e+00
Griewank $n=30$	Best	2.0818e-03	2.2447e-01	8.5431e+01	5.0283e+01	1.2236e+00	0	1.0310e+00	2.4814e+02	0	3.6212e+00	0	9.3258e-01	1.3701e-02	6.0915e-04	0
	Worst	8.8701e-03	6.0641e-01	1.5969e+02	1.3716e+02	1.8437e+00	6.9883e-07	1.6710e+00	3.8504e+02	1.0843e-06	1.0008e+02	2.1929e-08	1.0247e+00	4.9461e-02	2.1801e-03	0
	Mean	4.5944e-03	3.8130e-01	1.1177e+02	8.9390e+01	1.4597e+00	1.6413e-07	1.2715e+00	3.2054e+02	6.8560e-08	5.4297e+01	7.3098e-09	9.7403e-01	2.8735e-02	1.2516e-03	0
	Std	4.8938e-03	3.8901e-01	1.1313e+02	9.2359e+01	1.4678e+00	2.5518e-07	1.2795e+00	3.2186e+02	2.1075e-07	6.7162e+01	1.2661e-08	9.7478e-01	3.0205e-02	1.3500e-03	0
Ackley $n=30$	Best	5.9977e-02	2.9638e+00	1.4677e+01	1.3131e+01	2.4319e+00	3.0749e+00	1.2628e+00	1.7823e+01	1.0520e-06	1.7685e+00	1.6614e-05	4.2893e-01	2.0442e+01	3.9285e-03	1.8451e-06
	Worst	1.2660e-01	5.1890e+00	1.7859e+01	1.6651e+01	3.9909e+00	3.3452e+00	3.7850e+00	1.9858e+01	9.9818e-05	1.3443e+01	3.6098e-05	5.4068e-01	2.0734e+01	6.2015e-03	1.6462e+00
	Mean	8.6900e-02	3.6617e+00	1.6417e+01	1.5443e+01	3.2387e+00	3.2301e+00	2.6671e+00	1.9464e+01	1.8423e-05	1.0493e+01	2.3737e-05	4.8449e-01	2.0600e+01	4.9649e-03	7.1985e-01
	Std	8.8148e-02	3.6841e+00	1.6434e+01	1.5460e+01	3.2601e+00	3.2308e+00	2.7605e+00	1.9468e+01	2.7389e-05	1.0764e+01	2.5307e-05	4.8664e-01	2.0600e+01	5.0350e-03	9.0060e-01

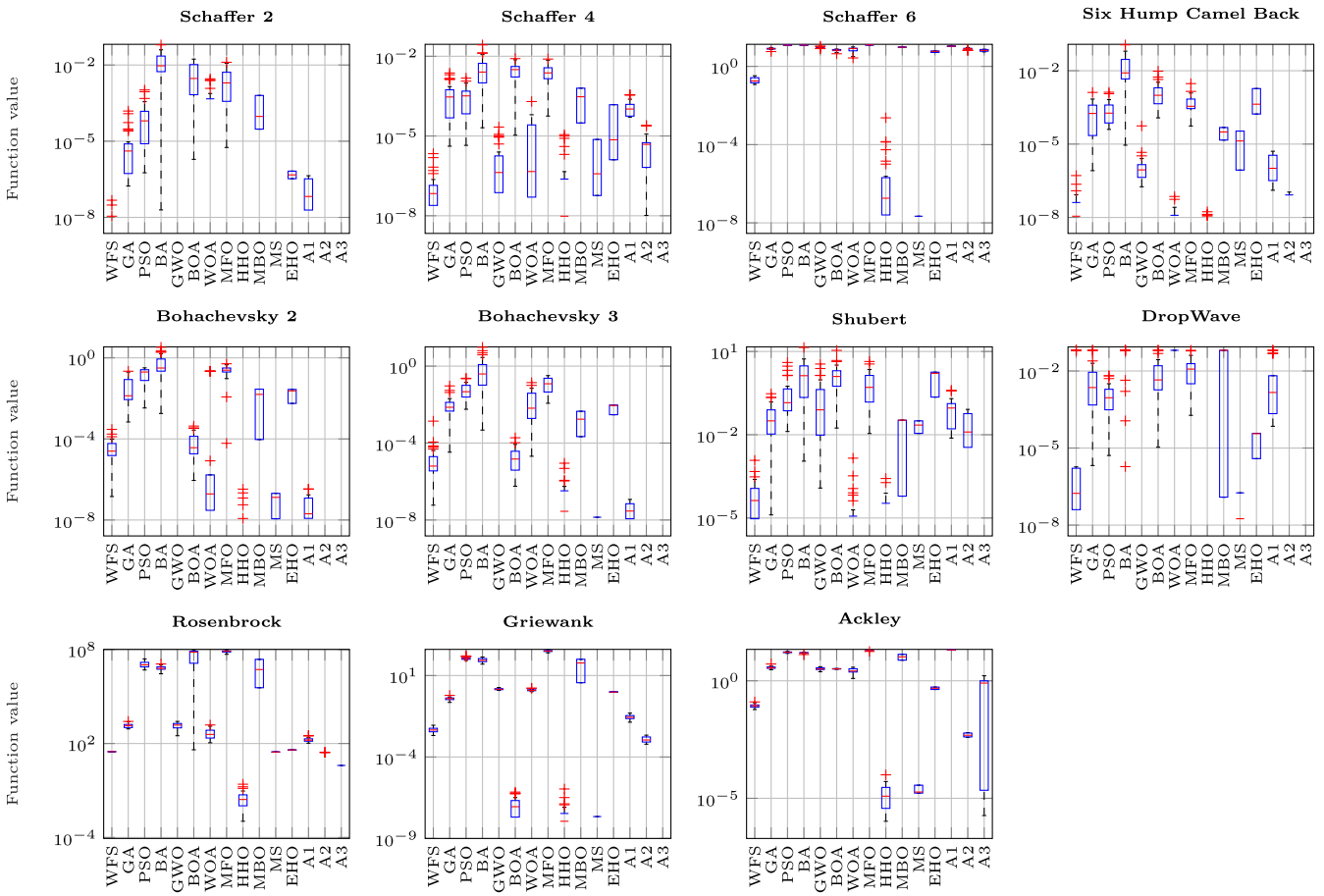

FIGURE 16. Boxplots for multimodal and non-separable classical benchmark functions.

ranked in the top w.r.t. both, classical and CEC 2020 benchmark functions.

As for the algorithm runtimes, Tab. 8 shows the mean runtime in seconds for 30 experiments with Rastrigin function

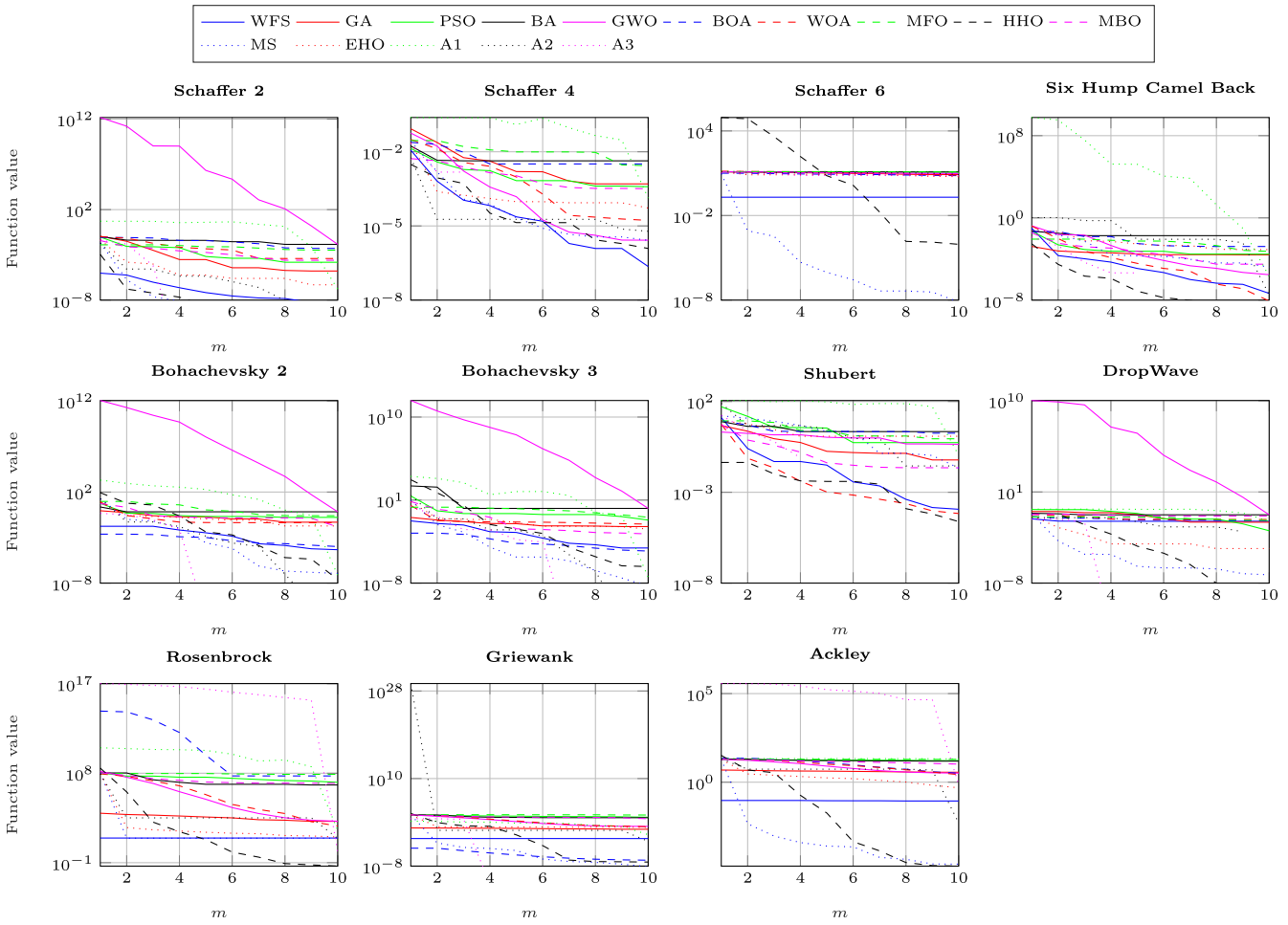

FIGURE 17. Convergence rate comparison for multimodal and non-separable classical benchmark functions.

TABLE 5. Comparative results for CEC 2020 benchmark functions.

Function		WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
Shifted and Rotated Bent Cigar Function $n = 10$	Best	2.2319e+04	1.4161e+07	8.6743e+07	2.6652e+08	4.2953e+06	2.6288e+09	8.9262e+05	1.0532e+09	1.1949e+07	8.2601e+07	8.7668e+07	2.6708e+09	1.5592e+03	6.8800e-03	1.7313e-04
	Worst	8.4253e+06	1.7207e+08	2.4226e+08	4.6786e+09	1.6827e+07	7.2379e+09	1.3635e+08	3.0220e+09	3.7564e+08	2.8759e+08	3.4545e+08	3.5060e+09	5.0991e+04	1.4966e+00	3.7062e-04
	Mean	1.2227e+06	6.2423e+07	1.5782e+08	2.1899e+09	9.6877e+06	4.7184e+09	3.5570e+07	1.9720e+09	1.0758e+08	1.9049e+08	2.2606e+08	3.0287e+09	2.5316e+04	4.2344e-01	3.4512e-04
	Std	2.2557e+06	7.1426e+07	1.6437e+08	2.4065e+09	1.0196e+07	4.9043e+09	5.1182e+07	2.0302e+09	1.3532e+08	2.0820e+08	2.4972e+08	3.0490e+09	2.8078e+04	6.0982e-01	3.4798e-04
Shifted and Rotated Schwefel's Function $n = 10$	Best	4.0434e+01	1.6999e+02	4.7656e+02	9.2162e+02	1.1981e+02	9.7204e+02	3.4050e+02	7.1295e+02	4.4546e+02	2.8091e+02	9.4877e+02	1.3136e+03	4.9219e+02	5.6304e+02	2.5219e+02
	Worst	1.2328e+03	5.7997e+02	1.7359e+03	1.7431e+03	1.6145e+03	1.8151e+03	1.2600e+03	1.5767e+03	1.2914e+03	5.9402e+02	1.3337e+03	1.4050e+03	1.2466e+03	1.0199e+03	8.0954e+02
	Mean	6.0837e+02	3.7943e+02	1.0578e+03	1.4089e+03	5.9012e+02	1.4396e+03	7.8857e+02	1.2834e+03	8.1252e+02	3.9320e+02	1.1145e+03	1.3623e+03	7.9842e+02	7.9520e+02	5.5080e+02
	Std	6.7103e+02	3.9671e+02	1.0948e+03	1.4271e+03	6.9121e+02	1.4509e+03	8.2454e+02	1.2949e+03	8.5374e+02	4.1817e+02	1.1262e+03	1.3628e+03	8.1587e+02	8.0675e+02	5.7243e+02
Shifted and Rotated Lunacek bi-Rastrigin Function $n = 10$	Best	1.8862e+01	2.2500e+01	5.1004e+01	5.8177e+01	2.2714e+01	6.0999e+01	2.7618e+01	1.0442e+02	3.0633e+01	2.7892e+01	5.8624e+01	7.6811e+01	2.6677e+01	2.6371e+01	2.0570e+01
	Worst	5.6147e+01	5.9041e+01	1.0257e+02	1.5861e+02	4.5051e+01	9.7838e+01	1.5023e+02	1.7507e+02	9.2867e+01	6.6050e+01	6.4365e+01	9.4543e+01	5.7235e+01	4.3375e+01	5.2979e+01
	Mean	3.4981e+01	3.8969e+01	7.7160e+01	1.1332e+02	3.3471e+01	8.3002e+01	6.0147e+01	1.4504e+02	6.5748e+01	4.7812e+01	6.1344e+01	8.4313e+01	4.0385e+01	3.5496e+01	3.1223e+01
	Std	3.6330e+01	3.9836e+01	7.8003e+01	1.1588e+02	3.3808e+01	8.3431e+01	6.4671e+01	1.4622e+02	6.7837e+01	5.0300e+01	6.1389e+01	8.4645e+01	4.0861e+01	3.5792e+01	3.1946e+01
Expanded Rosenbrock's plus Griewangk's Function $n = 10$	Best	7.1715e-01	1.9351e+00	3.0989e+00	1.1738e+01	2.0184e+00	4.6390e+00	1.0038e+00	1.9511e+01	1.2149e+00	4.3326e+00	6.6252e+00	2.2851e+03	1.8479e+00	1.6115e+00	9.3202e-01
	Worst	3.8888e+00	5.4059e+00	1.0212e+01	2.3992e+03	3.9949e+00	1.4543e+01	2.5968e+01	5.5488e+02	2.2538e+01	2.3480e+01	1.5653e+01	6.5064e+03	4.0591e+00	3.1685e+00	4.1281e+00
	Mean	2.1591e+00	3.5908e+00	6.1771e+00	5.5717e+02	3.1390e+00	6.9700e+00	5.8684e+00	2.3376e+02	6.0205e+00	1.1801e+01	1.0533e+01	4.3726e+03	2.9494e+00	2.6143e+00	2.5416e+00
	Std	2.2863e+00	3.6968e+00	6.3846e+00	8.4481e+02	3.1751e+00	7.3722e+00	7.7292e+00	2.8178e+02	7.8051e+00	1.4465e+01	1.1192e+01	4.7001e+03	3.0072e+00	2.6336e+00	2.7510e+00
Hybrid Function 1 $n = 10$	Best	8.5245e+02	1.0616e+03	2.2379e+03	6.8047e+03	8.1915e+02	1.5051e+04	1.2789e+03	7.5095e+03	1.0246e+03	1.3502e+04	2.5916e+03	1.0373e+05	1.2274e+02	3.2139e+01	7.1076e+01
	Worst	1.4113e+04	2.9058e+04	5.7881e+04	3.5422e+05	1.1816e+04	3.4133e+05	7.6522e+04	1.6913e+05	5.9045e+04	5.8533e+04	1.5282e+04	1.5989e+05	5.5590e+02	1.5897e+02	6.5992e+02
	Mean	4.8316e+03	9.3596e+03	1.3096e+04	8.6586e+04	3.3767e+03	8.4873e+04	7.7675e+03	5.8889e+04	9.7219e+03	3.7814e+04	7.0507e+03	1.4078e+05	3.3575e+02	8.8656e+01	4.0314e+02
	Std	6.2921e+03	1.1924e+04	1.6388e+04	1.1714e+05	4.3421e+03	1.0496e+05	1.5380e+04	6.8123e+04	1.8210e+04	4.2123e+04	9.1472e+03	1.4319e+05	3.5630e+02	9.4236e+01	4.3781e+02
Hybrid Function 2 $n = 10$	Best	4.7778e-01	4.9039e-01	6.7027e-01	9.4269e-01	5.6899e-01	1.1957e+00	4.7477e-01	1.0935e+00	4.7344e-01	4.8849e-01	9.5191e-01	2.1355e+00	5.6762e-01	5.1837e-01	4.6853e-01
	Worst	1.5933e+00	9.7621e-01	1.4140e+00	1.0429e+01	1.3058e+00	7.0053e+00	1.2813e+00	3.6124e+00	1.5929e+00	6.0446e-01	1.1212e+00	4.4672e+00	1.3109e+00	1.2602e+00	1.0933e+00
	Mean	8.8325e-01	7.6243e-01	1.0682e+00	3.0881e+00	9.3676e-01	3.4527e+00	9.1462e-01	1.8695e+00	1.0290e+00	5.2725e-01	1.0085e+00	3.0139e+00	9.3331e-01	8.3747e-01	6.3698e-01
	Std	9.3127e-01	7.7741e-01	1.0807e+00	3.7769e+00	9.5115e-01	3.8598e+00	9.5342e-01	1.9444e+00	1.0687e+00	5.3006e-01	1.0116e+00	3.1867e+00	9.5022e-01	8.6026e-01	6.6807e-01
Hybrid Function 3 $n = 10$	Best	3.8701e+02	2.4283e+02	5.7971e+02	1.4823e+03	6.0405e+02	1.8396e+03	2.2806e+03	1.6182e+03	6.1656e+02	2.0220e+03	6.2915e+03	1.9756e+03	6.1492e+01	2.6225e+00	6.2973e-01
	Worst	1.0946e+04	4.7560e+03	1.3689e+04	4.7186e+04	1.2679e+04	8.3432e+04	7.8837e+04	2.5439e+04	2.8531e+04	1.3502e+04	8.1311e+03	2.0709e+04	3.0259e+02	6.2968e+01	3.4193e+02
	Mean	2.6924e+03	1.3701e+03	3.0586e+03	1.4111e+04	3.7758e+03	2.6219e+04	2.0893e+04	7.7855e+03	1.0077e+04	5.9838e+03	7.0643e+03	9.2472e+03	1.4748e+02	3.0579e+01	1.5229e+02
	Std	3.7314e+03	1.7568e+03	3.8411e+03	1.8107e+04	4.8643e+03	3.2875e+04	2.7036e+04	9.4776e+03	1.3489e+04	8.0058e+03	7.1072e+03	1.2360e+04	1.5954e+02	3.4220e+01	1.7808e+02
Composition Function 1 $n = 10$	Best	1.8583e+00	3.4553e+01	4.1036e+01	7.4931e+01	7.7003e+00	1.1831e+02	2.7474e+01	1.5150e+02	2.5735e+01	4.2170e+01	5.9626e+01	2.2905e+02	1.0188e+02	1.0001e+02	5.6264e+01
	Worst	1.1335e+02	1.2159e+02	1.4421e+02	3.7901e+02	1.1126e+02	5.0630e+02	1.3598e+02	3.7020e+02	1.3596e+02	1.1952e+02	1.2878e+02	4.4245e+02	1.0795e+02	1.0666e+02	1.0986e+02
	Mean	5.4631e+01	7.2172e+01	9.1611e+01	2.0215e+02	7.6206e+01	2.5426e+02	8.8712e+01	2.4031e+02	8.7931e+01	7.1104e+01	1.0476e+02	3.1210e+02	1.0532e+02	1.0264e+02	9.9949e+01
	Std	6.9240e+01	7.6416e+01	9.7277e+01	2.1867e+02	8.8488e+01	2.7115e+02	9.5646e+01	2.4738e+02	9.4323e+01	7.9009e+01	1.0952e+02	3.2575e+02	1.0534e+02	1.0266e+02	1.0048e+02
Composition Function 2 $n = 10$	Best	1.0057e+02	1.3194e+02	1.3604e+02	1.8378e+02	1.0381e+02	1.7104e+02	1.1369e+02	2.1239e+02	1.2382e+02	1.2122e+02	1.2849e+02	2.7813e+02	1.1201e+02	1.0253e+02	1.0000e+02
	Worst	3.5425e+02	2.0106e+02	3.7776e+02	4.2402e+02	3.6024e+02	4.0936e+02	3.7179e+02	3.9998e+02	3.6037e+02	1.7619e+02	3.8623e+02	3.1647e+02	3.4430e+02	3.2499e+02	1.0000e+02
	Mean	1.5394e+02	1.6793e+02	2.2561e+02	2.6215e+02	2.2189e+02	2.4971e+02	2.1466e+02	3.1127e+02	1.7635e+02	1.4107e+02	2.1566e+02	2.9371e+02	1.6278e+02	1.4614e+02	1.0000e+02
	Std	1.7855e+02	1.6871e+02	2.4434e+02	2.7171e+02	2.5305e+02	2.5518e+02	2.3427e+02	3.1525e+02	1.9030e+02	1.4325e+02	2.4710e+02	2.9417e+02	1.7142e+02	1.5716e+02	1.0000e+02
Composition Function 3 $n = 10$	Best	3.9823e+02	4.1180e+02	4.2680e+02	4.6559e+02	4.0024e+02	4.7826e+02	4.0327e+02	4.9211e+02	4.0960e+02	4.2741e+02	4.1989e+02	6.5828e+02	3.9787e+02	3.9774e+02	3.9774e+02
	Worst	5.2453e+02	4.5368e+02	4.7897e+02	6.3771e+02	4.5118e+02	6.2956e+02	5.5342e+02	5.8954e+02	4.8612e+02	4.5860e+02	4.4857e+02	7.2453e+02	4.0091e+02	4.4393e+02	4.4786e+02
	Mean	4.3260e+02	4.3832e+02	4.5711e+02	5.2907e+02	4.3499e+02	5.4425e+02	4.5340e+02	5.4053e+02	4.5601e+02	4.4622e+02	4.3242e+02	6.8055e+02	3.9890e+02	4.2186e+02	4.0379e+02
	Std	4.3344e+02	4.3854e+02	4.5736e+02	5.3042e+02	4.3537e+02	5.4553e+02	4.5407e+02	5.4124e+02	4.5634e+02	4.4643e+02	4.3259e+02	6.8126e+02	3.9891e+02	4.2242e+02	4.0394e+02

for $n = 30$. Clearly, GWO, MFO and WFS are the fastest algorithms, while MBO, MS and EHO are the slowest ones.

In the context of the trade-off between performance and speed, it is interesting to note that WFS is faster than A2 and

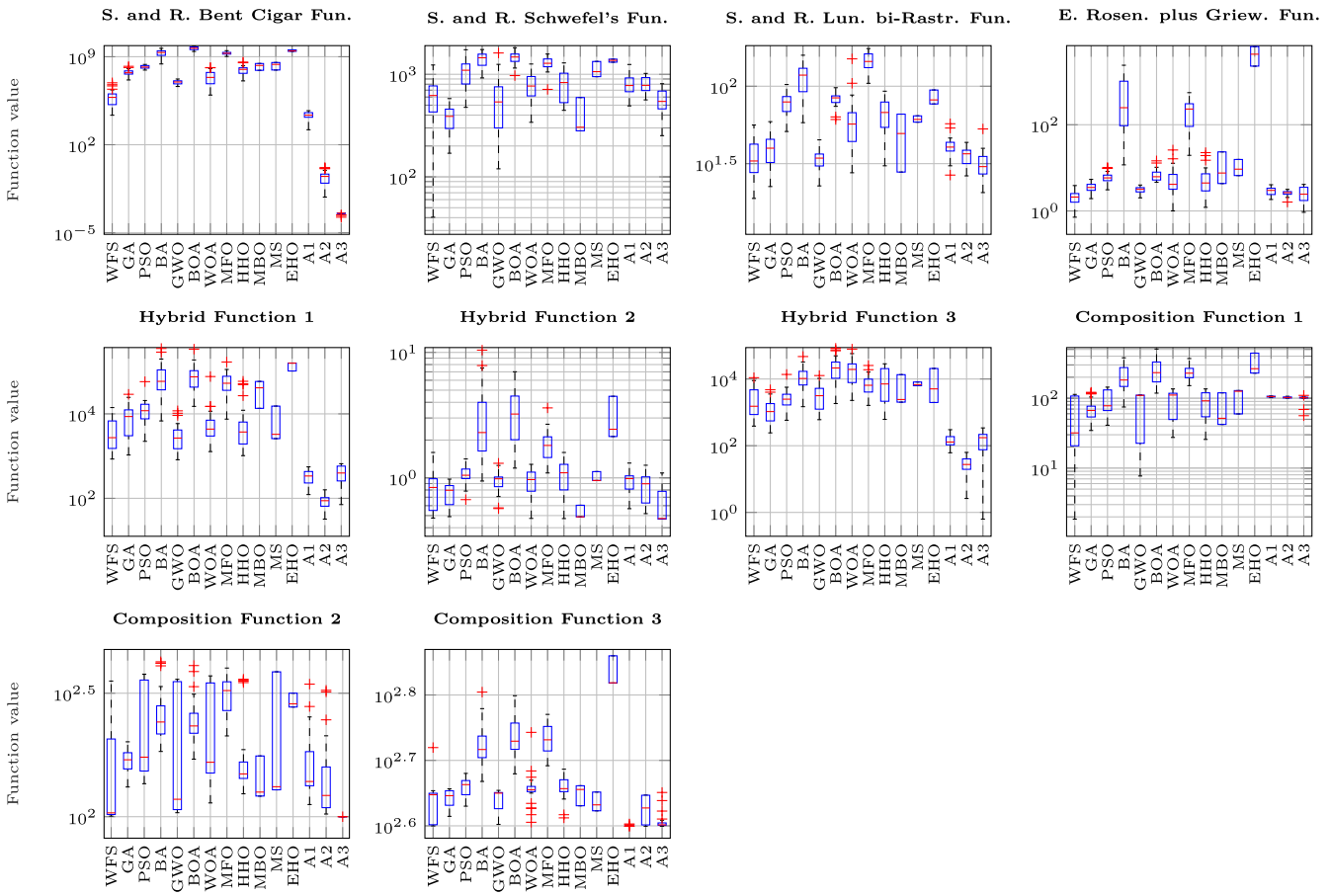

FIGURE 18. Boxplots for CEC 2020 benchmark functions.

TABLE 6. Final ranking for classical benchmark functions.

	WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
Best	134.5	267	360	351	220.5	277	183.5	405	89.5	320	196.5	316	247.5	142	90
Worst	160.5	268	342	410	213.5	299.5	254.5	392	106.5	302	152	261	241	130.5	67
Mean	157.5	268	341	404	208.5	299.5	238.5	398	100.5	309	161	276	236	134.5	68
Std	160.5	266	338	405	209.5	298.5	243.5	394	101.5	311	159	275	237	133.5	68

TABLE 7. Final ranking for CEC 2020 benchmark functions.

	WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
Best	28	61	89	114	51	129	62	130	61	88	112	146	58	48	23
Worst	59	49	96	141	57	134	100	124	97	63	74	116	39	23	28
Mean	38	50	95	133	55	135	77	127	88	63	89	138	49	37	26
Std	40	46	90	133	56	133	85	127	91	65	89	138	46	34	27

A3 more than an order of magnitude. We attribute this to the structural simplicity of WFS. Otherwise, the question about fair comparison of WFS versus A1, A2 and A3 arises due to the fact that they represent improved versions (e.g., A1 and A2 (LSHADE variants) are improved versions of DE (Differential Evolution) algorithm, and A3 (EBOwithCMAR) is improved version od BOA). Moreover, WFS is essentially very simple and "lean" algorithm compared to some competing algorithms equipped with "heavy artillery", which is clearly visible from both runtimes and pseudocode.

G. LIMITATIONS OF WFS ALGORITHM

Generally, the main limitation of WFS algorithm is the lack of guarantee that the computed solution is an optimal one (or even a near optimal), which is the common issue for a wide variety of metaheuristic algorithms. On the other hand, some limitations related to specific implementation of the presented WFS algorithm are as follows:

- The current implementation of WFS supports only box-constrained search spaces. On the other hand, the algorithm is adaptable to treat other constraints, which is the topic of ongoing research.

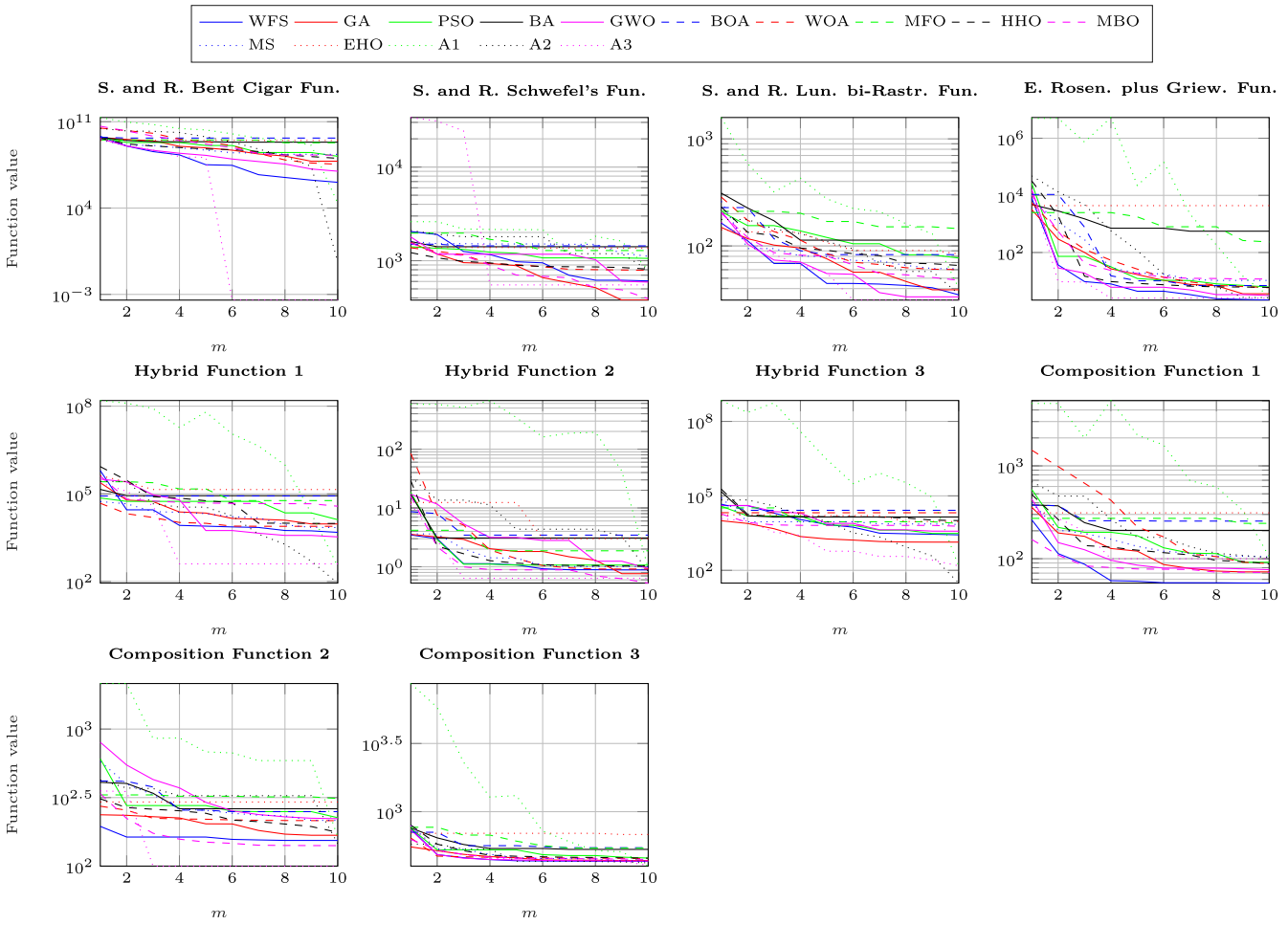

FIGURE 19. Convergence rate comparison for CEC 2020 benchmark functions.

TABLE 8. The mean runtime in seconds for 30 experiments with Rastrigin function for $n = 30$.

WFS	GA	PSO	BA	GWO	BOA	WOA	MFO	HHO	MBO	MS	EHO	A1	A2	A3
4.43	22.83	11.27	15.76	5.21	44.64	15.09	5.10	45.67	779.92	756.44	788.15	196.59	202.09	217.63

- The initial version of WFS algorithm used grid points as candidates for population members. Further experimentation has indicated than Halton points provide better results in general (see Subsection II-E). It remains an open question whether another type of low-discrepancy sequence would provide further performance improvements of the algorithm. However, regardless of the approach (grid, Halton points or any other low-discrepancy sequence), the problem that some parts of the search space are left uncovered remains. This problem becomes even more emphasized when dimensionality of the search space increases.
- Experiments have shown that the best results (on average) are achieved when flier's velocity v is in the range (10, 100) for the most of classical functions. However, there are some test functions for which WFS performs better when v is greater than 100. This calls for the idea to enable online adaptation of parameter v during the algorithm run. This surely remains an open question for future research.
- In some cases, the convergence rate of WFS is considerably slow. This phenomenon is notable in scenarios when the initial population yields solution which is relatively close to optimal one. When this occurs, in order to have the further advancement to the optimal solution, it is necessary to ''wait'' until the discretization step becomes small enough. Unfortunately, this usually implies many more iterations. For instance, this occurs for the following functions: Squares and Sum Squares function in Fig. 11, Schwefel 2.22, Schwefel 1.2 and Dixon-Price function in Fig. 13, Rastrigin function in Fig. 15, and Schaffer 6, Rosenbrock, Griewank and Ackley function in Fig. 17.

V. CONCLUSION AND FUTURE WORK

This paper presented the novel metaheuristic algorithm – Wingsuit Flying Search. Its mathematical model was described in detail along with the pseudocode. Halton sequence was proposed to be the default approach of generating points, though it would be interesting to establish should other low-discrepancy sequences provide better results.

Practically, WFS does not require any parameters, beside setting the population size and maximal number of iterations, which makes it very easy to use. Moreover, its structural simplicity render it very easy for implementation. It was compared to a number of related optimization algorithms. The test suite was chosen to be large enough to include a wide variety of problems, such as unimodal, multimodal, separable, non-separable and multi-dimensional problems. In total, 30 classical and 10 CEC 2020 benchmark functions were used. WFS and HHO have shown the best performance, excluding A2 and A3, w.r.t. best and the worst value, mean solution values, and the standard deviation of the solution values. Convergence rate tests showed that WFS is able to rapidly converge to the global optimum while keeping the exploration of the search space. It is confirmed that WFS has promising capabilities of balancing between exploitation and exploration of the search space. As for average runtimes, WFS has the best performance as well, though this feature may be implementation-dependent. Given WFS structural complexity, and particularly the fact that it is essentially paremater-free, this result is very positive, since WFS clearly successfully competes with related algorithms.

Future work will include a broader numerical study including both engineering applications, and wider range of test functions and search algorithms. Moreover, parallelized computation of WFS, treating other types of constraints, and online adaptation of parameter v should be examined.

REFERENCES

[1] S. Arora and S. Singh, "Butterfly optimization algorithm: A novel approach for global optimization," *Soft Comput.*, vol. 23, no. 3, pp. 715–734, Feb. 2019.

[2] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm," *Comput. Struct.*, vol. 169, pp. 1–12, Jun. 2016.

[3] N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 372–379.

[4] B. Basturk, "An artificial bee colony (ABC) algorithm for numeric function optimization," in *Proc. IEEE Swarm Intell. Symp.*, Indianapolis, IN, USA, 2006, pp. 397–414.

[5] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[6] E. Bonabeau, D. D. R. D. F. Marco, M. Dorigo, G. Théraulaz, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. London, U.K.: Oxford Univ. Press, 1999, no. 1.

[7] P. Civicioglu and E. Besdok, "A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 315–346, Apr. 2013.

[8] K. Deb, *Optimization for Engineering Design: Algorithms and Examples*. New Delhi, India; PHI Learning Pvt., 2012.

[9] W. Deng, J. Xu, and H. Zhao, "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem," *IEEE Access*, vol. 7, pp. 20281–20292, 2019.

[10] M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, "Ant colony optimization and swarm intelligence," in *Proc. 5th Int. Workshop*, vol. 4150. Brussels, Belgium: Springer, Sep. 2006.

[11] S. M. Elsayed, R. A. Sarker, D. L. Essam, and N. M. Hamza, "Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 1650–1657.

[12] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, Jan. 1986.

[13] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Commun. ACM*, vol. 7, no. 12, pp. 701–702, Dec. 1964.

[14] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.

[15] R. Hooke and T. A. Jeeves, "'Direct search'solution of numerical and statistical problems," *J. ACM*, vol. 8, no. 2, pp. 212–229, 1961.

[16] M. Jain, V. Singh, and A. Rani, "A novel nature-inspired algorithm for optimization: Squirrel search algorithm," *Swarm Evol. Comput.*, vol. 44, pp. 148–175, Feb. 2019.

[17] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," 2013, *arXiv:1308.4008*. [Online]. Available: http://arxiv.org/abs/1308.4008

[18] T. Johnson and P. Husbands, "System identification using genetic algorithms," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Berlin, Germany: Springer, 1990, pp. 85–89.

[19] A. Kaveh and A. Dadras, "A novel meta-heuristic optimization algorithm: Thermal exchange optimization," *Adv. Eng. Softw.*, vol. 110, pp. 69–84, Aug. 2017.

[20] R. Kumar, "Directed bee colony optimization algorithm," *Swarm Evol. Comput.*, vol. 17, pp. 60–73, Aug. 2014.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[22] A. Kumar, R. K. Misra, and D. Singh, "Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 1835–1842.

[23] B. Lacevic and E. Amaldi, "Ectropy of diversity measures for populations in Euclidean space," *Inf. Sci.*, vol. 181, no. 11, pp. 2316–2339, Jun. 2011.

[24] X. Li, "A new intelligent optimization artificial fish school algorithm," Ph.D. dissertation, College Comput. Sci. Technol., Univ. Zhejiang, Zhejiang, China, 2003.

[25] C. Yue, K. Price, P. Suganthan, J. Liang, M. Ali, B. Qu, N. Awad, and P. Biswas, "Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization," Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Tech. Rep. 201911, 2019. [Online]. Available: https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark/blob/master/Definitions%20of%20%20CEC2020%20benchmark%20suite%20Bound%20Constrained.pdf

[26] Y. Ling, Y. Zhou, and Q. Luo, "Lévy flight trajectory-based whale optimization algorithm for global optimization," *IEEE Access*, vol. 5, pp. 6168–6186, 2017.

[27] R. Martin and W. Stephen, "Termite: A swarm intelligent routing algorithm for mobilewireless ad-hoc networks," in *Stigmergic Optimization*. New York, NY, USA: Springer, 2006, pp. 155–184.

[28] A. G. Mersha and S. Dempe, "Direct search algorithm for bilevel programming problems," *Comput. Optim. Appl.*, vol. 49, no. 1, pp. 1–15, May 2011.

[29] Z. Michalewicz, J. B. Krawczyk, M. Kazemi, and C. Z. Janikow, "Genetic algorithms and optimal control problems," in *Proc. 29th IEEE Conf. Decis. Control*, Dec. 1990, pp. 1664–1666.

[30] S. Mirjalili, "The ant lion optimizer," *Adv. Eng. Softw.*, vol. 83, pp. 80–98, May 2015.

[31] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowl.-Based Syst.*, vol. 89, pp. 228–249, Nov. 2015.

[32] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[33] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[34] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[35] A. W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi, "LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2017, pp. 145–152.
[36] M. Molga and C. Smutnicki. *Test Functions for Optimization Needs*. Accessed: 2005. [Online]. Available: https://www.robertmarks.org/Classes/ENGR5358/Papers/functions.pdf
[37] D. Molina, A. LaTorre, and F. Herrera, "An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions," *Cognit. Comput.*, vol. 10, no. 4, pp. 517–544, Aug. 2018.
[38] A. Mucherino, O. Seref, O. Seref, O. E. Kundakcioglu, and P. Pardalos, "Monkey search: A novel Metaheuristic search for global optimization," *AIP Conf. Proc.*, vol. 953, pp. 162–173, Nov. 2007.
[39] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Comput.*, vol. 7, no. 1, pp. 65–85, 1988.
[40] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, Jan. 1965.
[41] H. Niederreiter, "Low-discrepancy and low-dispersion sequences," *J. Number Theory*, vol. 30, no. 1, pp. 51–70, Sep. 1988.
[42] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.
[43] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, Jun. 2009.
[44] L. A. Rastrigin, *Systems of Extremal Control*. Moscow, Russia: Nauka, 1974.
[45] Y. Sharafi, M. A. Khanesar, and M. Teshnehlab, "COOA: Competitive optimization algorithm," *Swarm Evol. Comput.*, vol. 30, pp. 39–63, Oct. 2016.
[46] H. Shareef, A. A. Ibrahim, and A. H. Mutlag, "Lightning search algorithm," *Appl. Soft Comput.*, vol. 36, pp. 315–333, 2015.
[47] K. Sörensen, "Metaheuristics—The metaphor exposed," *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.
[48] R. Storn and K. Price, "Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.
[49] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 1658–1665.
[50] S. A. Uymaz, G. Tezel, and E. Yel, "Artificial algae algorithm (AAA) for nonlinear global optimization," *Appl. Soft Comput.*, vol. 31, pp. 153–171, Jun. 2015.
[51] G.-G. Wang, "Moth search algorithm: A bio-inspired Metaheuristic algorithm for global optimization problems," *Memetic Comput.*, vol. 10, no. 2, pp. 151–164, Jun. 2018.
[52] G.-G. Wang, S. Deb, and L. D. S. Coelho, "Elephant herding optimization," in *Proc. 3rd Int. Symp. Comput. Bus. Intell. (ISCBI)*, Dec. 2015, pp. 1–5.
[53] G. G. Wang, S. Deb, and Z. Cui, "Monarch butterfly optimization," *Neural Comput. Appl.*, vol. 31, no. 7, pp. 1995–2014, 2019.
[54] G. G. Wang, S. Deb, and L. D. S. Coelho, "Earthworm optimisation algorithm: A bio-inspired Metaheuristic algorithm for global optimisation problems," *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 1–22, 2018.
[55] M. Xu, X. You, and S. Liu, "A novel heuristic communication heterogeneous dual population ant colony optimization algorithm," *IEEE Access*, vol. 5, pp. 18506–18515, 2017.
[56] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Proc. Int. Symp. Stochastic Algorithms*. Berlin, Germany: Springer, 2009, pp. 169–178.
[57] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, Univ. Cambridge, Cambridge, U.K., 2010.
[58] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization*. Berlin, Germany: Springer, 2010, pp. 65–74.
[59] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Proc. Int. Conf. Unconventional Comput. Natural Comput.* Berlin, Germany: Springer, 2012, pp. 240–249.
[60] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, Dec. 2009, pp. 210–214.
[61] M. Yazdani and F. Jolai, "Lion optimization algorithm (LOA): A nature-inspired Metaheuristic algorithm," *J. Comput. Des. Eng.*, vol. 3, no. 1, pp. 24–36, Jan. 2016.

NERMIN COVIC (Member, IEEE) was born in Suhodol, Sarajevo, Bosnia and Herzegovina, in 1994. He received the B.S. and M.S. degrees (Hons.) from the Department for Control Systems and Electronics, Faculty of Electrical Engineering, University of Sarajevo, in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree. He is also working as a Teaching/Research Assistant with the University of Sarajevo. His research interests include optimization, robotics, automation, control systems, and system identification.

BAKIR LACEVIC (Member, IEEE) received the Dipl.-Ing. and Magister degrees in automatic control from the University of Sarajevo, Sarajevo, Bosnia and Herzegovina, in 2003 and 2007, respectively, and the Ph.D. degree in information technology from Politecnico di Milano, Milano, Italy, in 2011. He is currently an Associate Professor with the Faculty of Electrical Engineering, University of Sarajevo, where he teaches courses in robotics and modeling/identification of dynamic systems. His research interests include robotic motion planning, human-robot interaction, optimization, and machine learning.