Neural Comput & Applic (2018) 30:3117–3144
https://doi.org/10.1007/s00521-017-2881-3

ORIGINAL ARTICLE

Drone Squadron Optimization: a novel self-adaptive algorithm for global numerical optimization

Vinícius Veloso de Melo[1] · **Wolfgang Banzhaf**[2]

Received: 4 April 2016 / Accepted: 10 February 2017 / Published online: 27 February 2017

Abstract This paper proposes Drone Squadron Optimization (DSO), a new self-adaptive metaheuristic for global numerical optimization which is updated online by a hyper-heuristic. DSO is an artifact-inspired technique, as opposed to many nature-inspired algorithms used today. DSO is very flexible because it is not related to natural behaviors or phenomena. DSO has two core parts: the semiautonomous drones that fly over a landscape to explore, and the command center that processes the retrieved data and updates the drones' firmware whenever necessary. The self-adaptive aspect of DSO in this work is the perturbation/movement scheme, which is the procedure used to generate target coordinates. This procedure is evolved by the command center during the global optimization process in order to adapt DSO to the search landscape. We evaluated DSO on a set of widely employed single-objective benchmark functions. The statistical analysis of the results shows that the proposed method is competitive with the other methods, but we plan several future improvements to make it more powerful and robust.

Keywords Global numerical optimization · Hyper-heuristic · Metaheuristic · Self-adaptive · Genetic programming · Coevolution

✉ Vinícius Veloso de Melo
vinicius.melo@unifesp.br

Wolfgang Banzhaf
banzhaf@msu.edu

[1] Institute of Science and Technology, Federal University of São Paulo, São José dos Campos SP, Brazil

[2] Department of Computer Science and Engineering, BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, MI 48864, USA

1 Introduction

Metaheuristics [23, 31] are derivative-free general optimization methods used to solve a variety of combinatorial and numerical optimization problems. They are usually employed when derivatives are costly or impossible to obtain and can deal with problems that are non-differentiable, irregular, noisy, or dynamic. Otherwise, if heuristics or derivatives are available, specialized methods may be orders of magnitude faster and more accurate than metaheuristics, because the former make few or no assumptions about the problem being optimized. Many metaheuristics are nature-inspired, and one of the most popular class of algorithms is the class of Evolutionary Algorithms (EAs) [36], one of whose representatives is the Genetic Algorithm (GA) [24]. GAs are based on Darwinian evolution, where individuals compete against each other to mate and propagate their genes to their offspring. As the comparably fitter individuals survive, the population improves over the generations, getting closer to the optimum solution.

Another important class of metaheuristics is that of the swarm algorithms [10], introduced with the Ant Colony Optimization (ACO) [17] method for combinatorial optimization. Often this can be couched in terms of finding good paths through graphs: Virtual ants move on the graph and deposit virtual pheromone on their path. Initial moves are random, but when ants find pheromone, they intend to follow the trail, consequently reinforcing the deposited pheromone. As shorter paths receive more pheromone, the shortest one will likely be found. Later, the Particle Swarm Optimization (PSO) algorithm [18] was proposed to solve continuous problems. Swarm algorithms like the PSO have a population of agents that interact locally with one another and with the environment, leading to the emergence of

intelligent global behavior commonly seen in bird flocks. PSO has inspired the creation of many nature-inspired algorithms such as the Bee Algorithm [44], Artificial Bee Colony [30], the Gravitational search algorithm [51], Glowworm swarm optimization [32], and more recently the Dragonfly Algorithm [38] and the Human behavior-based optimization [1]. In general, these methods present different recipes for how to combine information from a population, and sometimes other modifications. Fister Jr et al. [19] provide a short review and a list of such methods.

Nature-inspired metaheuristics are developed to follow strict rules regarding the generation of solutions because they aim to mimic some behavior abstracted from a natural system. However, it may be beneficial to consider optimization methods with relaxed rules that can *automatically* improve over time. Such improvement may occur, for instance, when some characteristic of the search space is detected that can lead the search to a promising region.

In order to achieve such flexibility, here we change the source of inspiration. Instead of adopting a natural paradigm, we propose an *artifact-inspired algorithm*,[1] that is, it is inspired by something artificially created (human-made), more specifically here, *drones*. In this paradigm, such an algorithm is not bound by a particular realization at hand; drones are *flexible* machines (in several respects), not biological entities. Instead, it can use a variety of different mechanisms/procedures without losing its core characteristics.

An important aspect of the technique proposed here is that it is self-adaptive regarding *code* modification, not only regarding the parameter configuration. Therefore, the technique can manipulate the procedures that the drones use to generate solutions, meaning that it can itself *to a degree* evolve during the search. Some researchers investigated similar approaches, but there are significant differences, which are discussed later in this paper.

Self-adaptation is one of the characteristics that can provide large improvements in performance [6, 41, 48, 61]. However, most techniques existing today use human-developed adaptation schemes that cannot cover every problem and may be unable to perform well in dynamic situations. Thus, methods that can learn and self-adapt are of great value.

The technique proposed here for single-objective optimization, named Drone Squadron Optimization (DSO), is a hybrid approach that employs coevolution to improve its own code. The key contributions of this paper are:

- the proposal of an artifact-inspired paradigm that is not tied to any natural phenomenon or behavior, but that could automatically act like any of them;
- a novel self-adaptive metaheuristic that can itself evolve on-the-fly and behave similar to an evolutionary or swarm algorithm;
- the introduction of an explicit separation between the controller and the semiautonomous exploration entities of a team approach.

The remaining of the paper is structured as follows: Sect. 2 presents our proposal. In Sect. 3, we present related works. Section 4 demonstrates the strength of the method with computational experiments. Section 5 presents conclusions and some future work.

2 Drone Squadron Optimization

Drones, like submarines or the well-known flying machines, such as balloons, airplanes, helicopters, quadcopters, can navigate autonomously or remotely. They have sensors, can communicate over vast distances, can use solar power, and—one of the most important features—can be upgraded or improved not only concerning hardware but also by changing their software (the firmware). Therefore, since these machines have software (firmware) to control their behavior, researchers are free to add mechanisms to the algorithm as simple software upgrades. Below, we adopt this technique. Later, more details on the components are provided.

Drone Squadron Optimization (DSO) may be related to Particle Swarm Optimization, the Artificial Bee Colony algorithm, or any other swarm algorithm because it is based on the movement of entities in the search space. However, as explained before, the swarm movement is not derived from behavior observed in nature. DSO's approach allows it to automatically choose to use recombination and/or variation of solutions with distinct procedures, making it act as an evolutionary algorithm, swarm algorithm, probabilistic algorithm, or other, according to how it performs in the search landscape. Nevertheless, it is not choosing from pre-coded algorithms [43, 49, 61]; it generates the *actual* code on-the-fly.

The DSO algorithm as presented here is composed of a drone squadron with different teams and a command center. The command center uses information collected by/from the drones to perform two operations (1) to maintain partial control of the search and (2) to develop new firmware for controlling the drones (see Fig. 1). A drone *is not* a solution; it moves to a coordinate (the actual solution). The firmware contains the procedures (codes) and configurations used by teams to search the landscape. In this

[1] The terminology employed in this work is using the artifact as a metaphor which—by way of analogy—can facilitate its understanding.

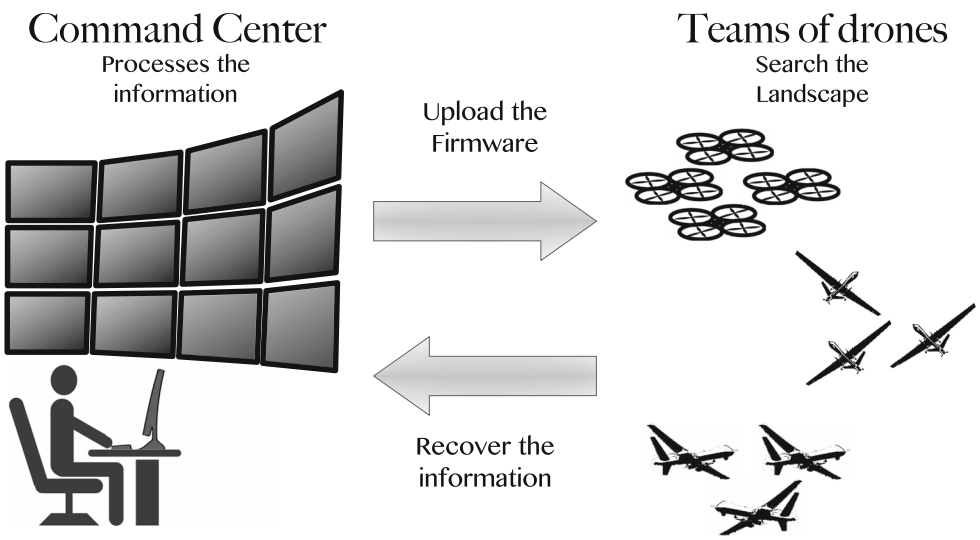

Fig. 1 High-level DSO abstraction showing the two main modules of DSO: the command center and the teams of drones. The command center uses information provided by the teams to update their firmware when necessary

work, the perturbation procedure is an actual source code, a *string* to be parsed and executed by the drone.

Conceptually, the command center is a central place for carrying out orders and for supervising tasks. It receives inputs, processes data, and generates outputs from internal decisions. The command center can update the firmware of drones whenever it decides, dynamically adapting a team's behavior to the problem.

A group of drones is divided into teams of the same size (necessary for the selection mechanism), where each team has its firmware that controls the movement of each drone; there is distinct firmware for each team, which means that *each team has a distinct way of sampling the search space from the same current set of solutions*.[2]

In the analogy, the drones have a search mission (the objective function) to locate a particular target on the landscape, whose value is obtained by the drone's sensors. The teams are not necessarily multiple groups searching distinct and distant regions of the landscape. In fact, all drones move from specific departure points that may be the same for some teams, but not for others. As the teams have distinct firmware, even though they move from the same departure points they may get to distinct coordinates, but may overlap in search regions. Moreover, a team does not have to follow another team, unless the command center encodes such behavior in the firmware.

One may observe in Algorithm 1 that DSO is more complex than traditional nature-inspired methods as it has several components to deal with the command center, the teams, the firmware adaptation, among others. We chose to propose an elaborate algorithm instead of a very simple one that would be slightly enhanced in further researches. More detailed explanations can be seen in "Appendix". For a better understanding, a mapping from DSO to Evolutionary Algorithms (EA) may be useful. We also provide a Glossary with the terms necessary to understand the remaining of the paper (see Table 1).

- **Coordinates:** a numerical solution (array);
- **Scan the landscape:** calculate the objective function; obtain the fitness value;
- **Firmware:** distinct rules/configurations to evolve the population;
- **Team:** group of agents that share a firmware but operate on possibly different data;
- **Squadron:** group of teams with different firmwares.

2.1 Command center

The command center is the "intelligent" part of DSO because it is responsible for producing and giving the orders, while the drones execute the orders to return the results. By using the information collected by the drones, the command center modifies the firmware to update the teams. To do that, the command center uses a hyper-heuristic approach to generate the firmware code.

Drones use new firmware and all data distributed by the command center to control their behavior. Conceptually, cheap data analysis may be performed on the drones due to their limited computing power, while expensive analysis is done at the command center to be accessed by the drones. The command center may be equipped with mechanisms to learn from the search to generate better firmware.

[2] It should be noted that teams are not like species nor niching in evolutionary algorithms.

2.2 The firmware

For the current DSO version, the firmware contains only the mechanism to generate new trial coordinates (TC) through perturbation; thus, this perturbation scheme is the core of the drone's firmware. A new firmware is generated based on the well-known perturbation scheme of a biased random walk:

$$P = Departure + Offset(), \quad (1)$$

$$TC = calculate(P), \quad (2)$$

where *Departure* is a coordinate (a solution point in the search space), *Offset* is a function that returns the actual perturbation movement (a numerical value), and P is the complete perturbation formula that has to be calculated to return the trial coordinates.[3]

Both the *Departure* coordinates and the function that produces the *Offset* are modified, that is, distinct teams may have distinct ways of choosing departure coordinates and how to calculate the offset. As examples, suppose the two following perturbations for teams 1 and 2, where C_1 is a user-defined constant, $G(0, 1)$ is a scalar sampled from a Gaussian distribution with zero mean and unit standard deviation, and $U(0,1)_D$ is an array of D numbers sampled from a uniform distribution with minimum 0 and maximum 1.

$$P_1 : \overrightarrow{GBC} + (C_1 \times (\overrightarrow{GBC} - \overrightarrow{CBC_{drone}})). \quad (3)$$

$$P_2 : \overrightarrow{CBC_{drone}} + (G(0,1) \times (\sqrt{U(0,1)_D} + \overrightarrow{CBC_{drone}})). \quad (4)$$

As one may observe, the two *Departures* are the arrays $\overrightarrow{GBC}$ and $\overrightarrow{CBC_{drone}}$, while the added expressions are the *Offsets*. The pattern of moving from a *Departure* point is important because it avoids shrinking the search space toward the origin.[4] In fact, even if the *Offset* is shrinking to zero, there is still noise applied to the *Departure* coordinates, which results in a neighborhood search.

In this work, we implemented the perturbation scheme using a *tree-structure* representation with terminal and non-terminal nodes, similarly to related works [45, 52]. The scheme always follows the pattern shown in Eq. 1, i.e., a sum of two terms, where the first one is selected from a particular subset of the terminals, and the second one is an expression grown using the available terminals and non-terminals.

At the outset perturbation schemes were randomly generated (arbitrary equations). However, after preliminary experiments, it was realized that a completely random set of initial perturbation schemes shows poor performance, requiring a *warming up* period. To overcome this issue, a set of *reference perturbations* was defined, that is, equations used as *initial* perturbations for the teams, but that may be replaced during the optimization process. For more details, please see "Appendix" section.

Table 1 Glossary

Term	Meaning
μ	Mean
σ	Standard deviation
C	A user-defined constant
CBC	A 2D structure containing the current best coordinates
$\overrightarrow{CBOFV}$	Array of current best coordinates objective function values
$\overrightarrow{Coordinates}$	A solution (an array)
CR	Differential Evolution algorithm crossover rate parameter
D	Number of dimensions (variables) of the objective function
F	Differential Evolution algorithm amplification factor parameter
G	The Gaussian distribution
$\overrightarrow{GBC}$	Global best coordinates, the best solution found so far
$GBOFV$	Global best objective function value
$\overrightarrow{LB}$	Array with the objective function lower bounds
$MaxIt$	Maximum number of iterations
$MaxStagnation$	Maximum number of iterations without improvement
$MVNS$	Multivariate normal sampling
N	Number of drones in each team
P	A perturbation formula
$Pacc$	Probability of accepting a solution worse than the ones in $\overrightarrow{CBC}$
$std-$dev	Standard deviation
t	Number of teams
TC	A 2D structure containing the trial solutions' coordinates
$\overrightarrow{TeamQuality}$	Array containing the quality of the teams
TmC	A 2D structure containing a team's coordinates
$\overrightarrow{TmOFV}$	Array with a team's objective function values
U	The uniform distribution
$\overrightarrow{UB}$	Array with the objective function upper bounds

2.3 Drone movement

The drones use an autonomous system to calculate target positions, move to them, and collect information that is sent back to the command center. The mechanisms

[3] It is important to note that all solutions are one-dimension arrays; therefore, all operations present in this work are element-wise.

[4] Many well-known benchmark functions have their global optimum at the origin, and there are algorithms that exploit this characteristic to achieve high performance.

available to DSO to calculate target positions are employed in various optimization techniques, evolutionary or non-evolutionary. Each mechanism, such as recombination and variation, may have more than one implementation, giving DSO many exploration and exploitation capabilities. More details are shown in "Appendix".

The goal of this step is to generate the target positions of each drone in each team ($TmC_{team,drone}$). After each drone runs the perturbation step and generates trial coordinates (TC_{drone}), it performs either a recombination with the best coordinates found so far to generate TmC or no recombination at all ($TmC = TC$). Currently, the choice is random, and all recombination procedures available to the drones have the same probability of being selected. Also, recombination is performed after perturbation, but changing the order is a perfectly plausible option; this changes the behavior of the method without invalidating the original proposal.

Finally, the drones may be allowed to move only inside a particular perimeter. Therefore, if coordinates in TmC are outside such perimeter (a violation), then a correction must be made. A few correction procedures are available to be chosen and there is no bias in the current DSO to privilege either of them. The violations are calculated as:

$$violation_{team} = \sum_{drone=1}^{N} \sum_{j=1}^{D} \begin{cases} |TmC_{team,drone,j} - UB_j| \\ + \\ |LB_j - TmC_{team,drone,j}| \end{cases}, \quad (5)$$

where N is the number of drones per team, D is the problem's dimension, UB is the upper bounds array, and LB is the lower bounds array. This calculation considers only cases where $TmC_{team,drone,j} > UB_j$ or $TmC_{team,drone,j} < LB_j$, i.e., when there is a violation. Therefore, the violations are accumulated for *each* team, considering all of its drones.

After the drones move and calculate the objective function, their results are sent to the command center to make decisions, such as updating the firmware.

2.4 Firmware update

At this stage, if necessary, the hyper-heuristic takes over. The command center uses two pieces of information to measure the quality of a team: (1) its rank regarding the objective function value (see Table 2) and (2) the degree of out-of-bound coordinates that they generated. Thus, $TeamQuality_i = Rank_i + violation_i$, for $i = 1, \ldots, t$. It is important to take violations into account because good solutions may be generated by the correction procedures just by chance, while TmC had, in fact, large violations. $TeamQuality$ is calculated every iteration; it can be accumulated *or* averaged after a series of iterations to be compared with a threshold.

Table 2 Example of ranking on a *minimization* problem for two teams with three drones: (a) Objective Function Values per Team and (b) Ranks and averages

$TmOFV_1$	$TmOFV_2$	
(a) Values		
4.0	4.0	
2.0	3.0	
8.0	9.0	
	$Rank_1$	$Rank_2$
(b) Average ranks		
	1.0	1.0
	1.0	2.0
	1.0	2.0
$\overline{Rank}$	1.0	1.67

As soon as a firmware update criterion is reached (amount of iterations, for instance), the command center replaces the w worst firmware by variations of the w best firmware, i.e., for $w = 1$ the team with the worst accumulated rank has its firmware updated with a variant (*random sub-tree replacement*) of the best team's firmware. There is no recombination of codes, and the new variant must satisfy the following rules:

1. The size S of the new perturbation P_k, where S is the number of nodes in the tree data structure, and k is the index of the worst of the t teams, has to be $S(P_k) > s_{\min}$ and $S(P_k) < s_{\max}$, where $s_{\min}$ and $s_{\max}$ are user-defined parameters;
2. The new perturbation has to be distinct from the original one, but the current version only detects syntactic differences, not semantic ones;
3. A function is not allowed to receive the same argument for the two parameters, for instance, $sub(Shift, Shift)$;
4. The w reference perturbations must not be replaced; thus, they are *fixed perturbations*;
5. The perturbation scheme of Eq. 1 must hold.

Consequently, one expects a performance improvement after replacing the firmware with the worst results by a variation of the firmware that achieved the best results. Concerning the *fixed* perturbations, this mechanism is to provide at least one firmware with reasonable search capability. A *fixed* perturbation may be one that favors exploration, whereas others may be free to perform exploitation.

2.5 Selection for next iteration, stagnation detection and treatment

When exploiting a particular region of the landscape, the drones may generate identical or almost identical target

coordinates. Convergence avoidance mechanisms, which are not present in most evolutionary algorithms, may help the drones to escape from local optima. The use of scaling (Gaussian, Uniform, etc.), detection and treatment of stagnation, and generation of opposite coordinates enable moves to regions far from the neighborhood. While this may slow down reaching the optimum solution and/or reduce its accuracy, it can expand the exploration capability of the method to locate promising regions in the search space.

After all drones return what they found on the landscape (the objective function values), the command center decides which information is important to keep in the search plan in order to be used in the future. A hard selection mechanism (as shown in Algorithm 1) chooses the best

Algorithm 1 High-level DSO algorithm. The abbreviations are shown in the Glossary.

Input: objective function, problem bounds ($\overrightarrow{LB}$,$\overrightarrow{UB}$), user defined constants, number of teams (t), number of drones per team (N), maximum number of iterations ($MaxIt$), maximum number of iterations without improvement ($MaxStagnation$), options for the firmware
Output: Best solution (GBC) and best Objective function value ($GBOFV$)

Initialize the command center, drones and teams in TmC, where TmC are teams containing the drones' coordinates
Send the drones to scan the landscape at $(t \times N)$ random coordinates
Select the N best coordinates to be CBC

while stopping criteria are not met **do**
▷ Procedures performed by the Teams
  for each $team \in [1, 2, \ldots, t]$ **do**
    for each $drone \in [1, 2, \ldots, N]$ **do**
      Generate TC using the perturbation scheme of the current team
      Choose the recombination method and apply it to combine TC with CBC_{drone}, resulting in the $TmC_{team,drone}$
      Choose out-of-boundary correction method and apply it to $TmC_{team,drone}$, saving the violations that occurred
      Move to the new coordinates ($TmC_{team,drone}$), scan the landscape, and set $TmOFV_{team,drone} \leftarrow$ objective function value
    end for
  end for
▷ Procedures performed by the command center
  Select the best coordinates found by the teams in the current iteration
  $improved \leftarrow false$
  for each $drone \in [1, 2, \ldots, N]$ **do** ▷ Same drone id, but distinct teams
    Rank $TmC_{team,drone}$ according to $TmOFV_{team,drone}$
    Calculate $violation_{team}$
    $bestIdx \leftarrow$ index of the best $TmOFV_{team,drone}$
    if $TmOFV_{drone,bestIdx} < CBOFV_{drone}$ **then**
      $CBOFV_{drone} \leftarrow TmOFV_{drone,bestIdx}$
      $CBC_{drone} \leftarrow TmC_{drone,bestIdx}$
      if $TmOFV_{drone,bestIdx} < GBOFV_{drone}$ **then**
        $improved \leftarrow true$
        $GBOFV \leftarrow TmOFV_{drone,bestIdx}$
        $GBC \leftarrow TmC_{drone,bestIdx}$
      end if
    end if
  end for
  if $improved = false$ **then**
    Check and apply stagnation control
  end if
  Set $TeamQuality$ as the average rank plus boundary violations
  Update the firmware according to $TeamQuality$
end while

between the current solution and the new one, considering the objective function values. However, stagnation must be treated.

Stagnation is detected when the objective function value of the current best solution at a *particular index* remains the same after a certain number of iterations. Then, DSO uses *soft selection* to allow further exploration:

$$\textbf{if } (TmOFV_{drone,bestIdx} < CBOFV_{drone} \textbf{ or } U(0,1) < P_{acc})$$
$$\textbf{then} \overrightarrow{CBC_{drone}} = \overrightarrow{TmC_{drone,bestIdx}},$$

where *bestIdx* is the index of the best drone (rank 1 considering all teams), $U(0, 1)$ is a random number from a uniform distribution between zero and one, and P_{acc} is the probability of accepting a worse solution. Thus, coordinates that resulted in lower-quality solutions can be inserted in the search plan, replacing higher-quality solutions. Elitism concept is applied here to keep the best solution in *CBC*.

As one may note, DSO employs several mechanisms to generate better sampling procedures, to investigate different regions of the search space, to correct solutions, among others. The next section briefly presents some related work that constitute successful attempts at using hyper-heuristics or multiple schemes to improve traditional metaheuristics.

2.6 Example of an iteration

Here we provide a simple example iteration showing how DSO works (see Algorithm 1).

1. Current configuration/state (the abbreviations are shown in the Glossary):
   - Minimize $f(\mathbf{x}) = \sum_{j=1}^{D} x_j$ with $D = 2$ variables;
   - $\overrightarrow{LB} = [0.0 \quad 0.0]$ and $\overrightarrow{UB} = [5.0 \quad 5.0]$;
   - $t = 2$ teams with $N = 2$ drones each;
   - The two firmware have perturbations $P1 = CBC - 1.0$ for team 1 and $P2 = CBC \times 2.0$ for team 2;
   - $\overrightarrow{GBC} = [1.0 \quad 2.0]$ and $GBOFV = 3.0$;
   - $CBC = \begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix}$ and $\overrightarrow{CBOFV} = [3.0 \quad 7.0]$.
2. Generate TC_{team} for each of the two teams:
   $$TC_1 = calculate(P1) = \begin{bmatrix} 0.0 & 1.0 \\ 2.0 & 3.0 \end{bmatrix} \text{ and}$$
   $$TC_2 = calculate(P2) = \begin{bmatrix} 2.0 & 4.0 \\ 6.0 & 8.0 \end{bmatrix}.$$
3. Recombine TC and CBC as $\overrightarrow{TmC_{team,drone}} = [TC_{team,drone,1} \quad CBC_{drone,2}]$, $drone = 1, \ldots, N$:
   $$TmC_1 = \begin{bmatrix} 0.0 & 2.0 \\ 2.0 & 4.0 \end{bmatrix} \text{ and } TmC_2 = \begin{bmatrix} 2.0 & 2.0 \\ 6.0 & 4.0 \end{bmatrix}.$$
4. Check for violations and correct the coordinates according to the problem's bounds ($\overrightarrow{LB}$ and $\overrightarrow{UB}$), also accumulating the violations for later use:
   $$TmC_1 = \begin{bmatrix} 0.0 & 2.0 \\ 2.0 & 4.0 \end{bmatrix}, TmC_2 = \begin{bmatrix} 2.0 & 2.0 \\ \underline{\boldsymbol{5.0}} & 4.0 \end{bmatrix},$$
   $\overrightarrow{violations} = [0.0 \quad 1.0]$.
5. Calculate the objective function values for each drone in the team $TmOFV_{team,drone}$:
   $\overrightarrow{TmOFV_1} = [2.0 \quad 6.0]$ and $\overrightarrow{TmOFV_2} = [4.0 \quad 9.0]$.
6. Calculate the rank offspring each drone in the team $TmOFV_{team,drone}$:
   $\overrightarrow{Rank_1} = [1.0 \quad 1.0]$ and $\overrightarrow{Rank_2} = [2.0 \quad 2.0]$.
7. Update the coordinates CBC_{drone} and $\overrightarrow{GBC}$ with the new best values of each $drone = 1, \ldots, N$, if necessary:
   - $CBC = \begin{bmatrix} 0.0 & 2.0 \\ 2.0 & 4.0 \end{bmatrix}$ and $\overrightarrow{CBOFV} = [2.0 \quad 6.0]$;
   - $\overrightarrow{GBC} = [0.0 \quad 2.0]$ and $GBOFV = 2.0$.
8. Because *GBOFV* has improved, there is no need to apply stagnation control;
9. Calculate *TeamQuality* as the average rank of $\overrightarrow{TmOFV}$ plus the violations:
   $\overrightarrow{TeamQuality} = [1.0 \quad 2.5]$.
10. Update the firmware of the lowest quality team:
    $P2 = CBC - 1.0/2.0$.

3 Related work

In this Section, we compare DSO with well-known popular metaheuristics and with other approaches for automated improvement of metaheuristics.

3.1 Popular population-based metaheuristics

In Particle Swarm Optimization (PSO, [18]), a velocity equation is used to move a single population of particles over the search space. Each particle has a current position and a historically (local) best position. The particle's movement is influenced by both the local (cognitive) and global (social, considering the neighborhood) positions.

The velocity equation shown in Eq. 6, where v_i is the velocity of particle i, c_1 and c_2 are user-defined constants, $U(0, 1)$ is a uniform distribution, x_i is the position of particle i, $local\_best_i$ is the best position found by particle i (memory), $global\_best$ is the best solution found up to the current iteration.

$$v_i = v_i + c_1 \times U(0, 1) \times (x_i - local\_best_i) + c_2 \times U(0, 1) \times (x_i - global\_best). \quad (6)$$

All particles move every iteration, as there is no selection mechanism to move only specific particles. In the canonical PSO, there is a single and fixed equation, a single population, and no mechanism to escape from local optima, although large velocity values may help. On the other hand, small values are necessary for exploitation and fine tuning of solutions. Researchers have proposed velocity equations [25] and mechanisms to adapt PSO configuration on-the-fly, making it adaptive [67] in order to reduce premature convergence and improve solution quality.

Differential Evolution ([47], DE) is a population-based metaheuristic in which two or more solutions are randomly selected to be combined through mutation to result in a trial vector and then recombined with a particular solution of the population. In the mutation strategy in Eq. 7 (*rand*/1), v_i is the trial vector of solution i, F is the user-configured amplification factor, x_{r1}, x_{r2}, and x_{r3} are randomly chosen solutions.

$$v_i = x_{r1} + F \times (x_{r2} - x_{r3}). \quad (7)$$

There are many mutation strategies, each producing a different search behavior, and basically two recombination operators. Mutation strategies and adaptive versions of DE have been proposed over the years, achieving better performance than the original algorithm [12, 14].

Covariance Matrix Adaptation Evolutionary Strategies (CMA-ES, [28]) are also categorized as an Estimation of Distribution Algorithm [33] because offspring is sampled from a model instead of randomly combined with other solutions. In CMA-ES, the model (see Eq. 8) is a multivariate normal distribution with the mean (μ), the step-size (σ), and the covariance matrix (Cov) of the successful individuals in the population (the best ones, according to their fitness value), which is used to generate solution x_i.

$$x_i = \mathcal{N}(\mu, \sigma^2 Cov). \quad (8)$$

The idea behind sampling a model is to increase the probability of successful candidate solutions, reducing the chance of sampling in low-quality regions of the search space. CMA-ES has several parameters that are automatically adjusted by the algorithm using specific equations, making it a powerful adaptive optimization method. Researchers have investigated restarts [3], multiple populations [26], and different parameter adjustments [60].

As can be noticed, the well-known methods presented above use human-made equations to generate new solutions. Following the *No Free Lunch Theorem* [63], we assume that there is no equation and parameter configuration that will achieve the best performance on all optimization problems. For that reason, we developed DSO to try different equations and constants during the search process. Procedures for allowing DSO to escape from local optima and dealing with stagnation are also present in this first version of the algorithm. Other related work is presented next.

3.2 Automated improvement of metaheuristics

Here we consider only the most closely related work, i.e., work that employs some mechanism to evolve a metaheuristic regarding the mutation/perturbation operator. We can separate the research into two basic evolutionary branches: offline and online. In offline evolution, the main method is a hyper-heuristic [11], such as a GP-like method [8, 9], responsible for generating new code for the metaheuristic and running it for a number of repetitions on one or more benchmark problems. The statistical result of the runs is sent back to the hyper-heuristic as the fitness value. In online evolution, on the other hand, the hyper-heuristic and the metaheuristic work in cooperation, which means that there is a single metaheuristic run, not a set of repetitions, and the metaheuristic's code is evolved during the optimization by the hyper-heuristic. Therefore, while in the offline approach the hyper-heuristic tries to find the best static code to solve a single problem or various problems, in the online approach the code is evolved for the particular problem in a particular run.

Regarding the offline approach, GP was used by Poli et al. [45] to evolve optimal velocity updating equations for PSO. The authors tested the approach on 30 random problems taken from two classes of landscapes: the city-block sphere and the Rastrigin function. The problems were evaluated for two and ten dimensions. The conclusion was that GP can quickly evolve better velocity equations than those designed by experts, even though GP used only the four basic arithmetic operations and constants. Being an offline approach, the method is executed to find the optimum equations for the problems used in the objective function. Therefore, a single equation is used in a complete PSO run.

Pavlidis et al. [42] evolved mutation strategies for the Differential Evolution (DE) algorithm, also using GP as a hyper-heuristic. They performed tests on some of the

training benchmark problems used in our experiments. The algorithm used three problems during the training phase and other two on the test phase. In the comparison, the evolved strategies performed better than the classical ones, suggesting that they were not specialized. Thus, this paper provides evidence that the automatic development of algorithms is feasible.

Melo and Carosio [15] extended the work of Pavlidis et al. [42], evolving DE for a single benchmark problem and evaluating it on a set of 20 benchmark functions (the same used here). The evolved strategy significantly outperformed the canonical one (*rand/1/bin*) for the large majority of problems. Also, the evolved strategy was employed for training an MLP neural network to solve regression and classification problems, outperforming the canonical strategy for this task as well.

Hong et al. [29] used GP to automatically design mutation operators (probability distributions) for the Evolutionary Programming (EP) algorithm to replace the classical Gaussian, Cauchy, and Lévy distributions. The approach was tested on ten function classes. The authors report that, for all problems, the probability distributions discovered by GP outperformed both the Gaussian and Cauchy distributions. However, they also state that they can outperform standard mutation operators just by changing σ of the Gaussian distribution, reducing it to fine tune the solutions. Therefore, there is no need to generate a new equation, just adapt σ.

Woodward and Swan [64] employed Iterated Local Search to automatically generate mutation operators for Genetic Algorithms (GA) and tune them to problem instances drawn from a given problem class. Machine-designed mutation operators were tested on seven problem classes and outperformed the human-designed operators. One important aspect of the framework is the actual use of human-designed operators as starting solutions; thus, the proposed method tries to improve an existing heuristic instead of evolving one from scratch.

Miranda and Prudêncio [37] proposed GEFPSO, a framework using Grammatical Evolution (GE) to automatically generate effective PSO designs. GE searches for not only structures but also parameter values (e.g., acceleration constants, velocity equations, and different particles' topology). Tests in 16 well-known benchmark problems showed that GEFPSO achieved competitive solutions when compared to well-succeeded algorithms from the literature, but achieving specialized algorithms. In future works, such specializations can later be used in problems presenting a similar structure (fitness landscape).

Only a few online approaches are reported in the literature. Rashid and Baig [52] proposed PSOGP, an extension of [45] that works online, evolving the code during the optimization. Each particle in the population has its force generating function, which is evolved by GP. This approach gave PSO better exploration abilities by slowing convergence and increasing chances of escaping from local optima. In a similar approach, Si et al. [54] replaced GP by GDE (Grammatical Differential Evolution [40]) and also achieved better performance than the PSO version used in the comparison.

3.3 Other metaheuristics

DSO employs ideas of several distinct frameworks and does not belong to any specific one. DSO can adapt its parameters to produce better performance during the optimization process, as similar adaptive techniques [48, 59, 62, 68]. However, DSO does more than that.

In DSO, the perturbation operator is also modified, not just the parameters. In this case, DSO can be seen as a generative hyper-heuristic approach [11], where code is evolved instead of parameters. However, in a standard hyper-heuristic application, the generation of codes is the main procedure, while in DSO it is a component of the method. Thus, DSO is a hybrid approach with coevolution of solutions to (1) the problem being optimized and (2) the procedures to optimize such problem.

DSO has independent teams operating on current solutions, which is similar to distributed EAs (dEAs) [2, 35, 58]. Nonetheless, most dEAs employ the same algorithm on all subpopulations (either with the same or different control parameters). On the other hand, in DSO distinct teams may use the same *Departure* coordinates (same population), but with distinct ways of calculating the *offset*. At other times, the teams may have the same *offset* formulation, but distinct *Departures*. Also, while one team may use recombination, the other one may not use it. Given that the teams may have distinct perturbation and recombination procedures, resulting in different behaviors, they could be seen as distinct algorithms.

In dEAs the parallel algorithms have their own populations searching distinct regions of the search space, and a migration procedure is adopted to try to escape from local optima and to speedup search. On the other hand, the teams in DSO are not subpopulations. Hence, no migration is required.

Another important argument why migration is not used is because DSO's teams are neither directly coevolving nor cooperating to each other. In fact, the selection mechanism considers the solutions of all teams; therefore, they are primarily competing to each other, even though all teams may improve the same shared information. Nevertheless, a comparison between competition and cooperation is out of the scope of this paper.

By using distinct algorithms at the same time, DSO could be a portfolio [43] of evolving (adapting) evolutionary processes. Most differences for dEAs can be applied to the portfolio framework, but in portfolios and other similar methods, such as AMALGAM [61] and A-Teams [49], distinct populations use distinct algorithms. However, again, these algorithms are previously coded and are not improved during the optimization.

Finally, it is important to recall that DSO's teams are semiautonomous; they must obey the command center. The command center can upgrade the firmware, select the solutions found by the teams, change the amount of teams and drones in each team, restart the drones' positions, among other possibilities.

4 Experimental results

To evaluate the performance of DSO, it is applied to minimize a set of 38 scalable unconstrained (box-constrained) single-objective continuous global optimization benchmark functions that are widely used in the literature. The results are statistically compared with those of well-succeeded evolutionary and swarm optimization methods.

4.1 Benchmark functions

The benchmark functions for both experiments are presented in next subsections.

4.1.1 Experiment 1

A definition of the test problems is presented in Table 3, a short description of their characteristics is presented next, and more details can be found in [65]. The entire set contains 23 functions, but for this contribution we selected only those that can be scaled, thus ignoring the low dimensional problems. For all problems, all methods are allowed to perform a maximum of $D \times 10,000$ objective function evaluations.

1. **Unimodal Functions**
   - (a) Separable, scalable: f_1, f_2, f_5, and f_6. f_5 is the Rosenbrock function which is unimodal for $D = 2$ and 3 but may have multiple minima in high dimension cases;
   - (b) Non-separable, scalable: f_3 and f_4;
   - (c) Non-separable, scalable, narrow valley from local to global optimum: f_7;

Table 3 Benchmark function definitions

Function definitions	Bounds	Optimum value
$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$	0
$f_2(x) = \sum_{i=1}^{D} \lvert x_i\rvert + \prod_{i=1}^{D} \lvert x_i\rvert$	$[-10, 10]^D$	0
$f_3(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]^D$	0
$f_4(x) = \max_i\{\lvert x_i\rvert, 1 \le i \le D\}$	$[-100, 100]^D$	0
$f_5(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$,	$[-30, 30]^D$	0
$f_6(x) = \sum_{i=1}^{D} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^D$	0
$f_7(x) = \sum_{i=1}^{D} i x_i^4 + U(0, 1)$	$[-1.28, 1.28]^D$	0
$f_8(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{\lvert x_i\rvert})$	$[-500, 500]^D$	$-418.9828872724339 \times D$
$f_9(x) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^D$	0
$f_{10}(x) = -20\exp\left(-0.2\sqrt{D^{-1}\sum_{i=1}^{D} x_i^2}\right) - \exp(\frac{1}{D}\sum_{i=1}^{D} \cos 2\pi x_i) + 20 + e$	$[-32, 32]^D$	0
$f_{11}(x) = (1/4000)\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(x_i/\sqrt{i}) + 1$	$[-600, 600]^D$	0
$f_{12}(x) = \frac{1}{D}\pi\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_D - 1)^2\right\}$ $+ \sum_{i=1}^{D} u(x_i, 10, 100, 4)$ $y_i = 1 + (x_i + 1)/4$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a \le x_i \le a, \\ k(-x_i - a)^m, & x_i < -a, \end{cases}$	$[-50, 50]^D$	0
$f_{13}(x) = \left\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]\right\}/10 + (x_D - 1)$ $[1 + \sin^2(2\pi x_D)]/10 + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]^D$	0

D, number of dimensions, was set differently for each sub-experiment

2. **Multimodal Functions**

   (a) Separable, scalable, numerous local optima: f_8–f_{13}.

These problems are currently considered of low difficulty, but remain very popular. Achieving good performance on them is mandatory, but it is also imperative to evaluate the method on a more challenging experiment.

4.1.2 Experiment 2

For this second experiment it selected the 25 functions from the benchmark problem set from CEC' 2005 Special Session in Real Parameter Optimization [57]. Those problems present a diverse set of features, such as multimodality, ruggedness, ill-conditioning, interdependency. In our experiments, all the functions were solved in 10 dimensions, with a maximum of 100,000 objective function evaluations, and all configurations suggested in [57].

4.2 Implementation and configuration of the algorithms

The experiments were executed on an Intel(R) Xeon(R) X5550@2.67 GHz with 8 GB RAM, Linux enterprise 3.14.1-gentoo i686. We implemented and executed DSO in MATLAB R2012b. Its configuration for Experiment 1 is shown in Table 4 and was empirically chosen after a few runs with distinct configurations. Constants C1, C2, and C3 are not used in the reference perturbations, but are available for DSO to compose new perturbations to update the firmware. DSO was not tuned to solve the problems as it should be able to automatically adapt most parameters.

The other methods used in our comparison were configured as follows. The MATLAB source codes were obtained online, mainly on the Web sites of the original authors. We did not tune the parameters of these methods, but used the configuration reported on the literature as suggestion to achieve good performance. Therefore, one must suppose that, based on published reports, with these configurations the methods will achieve high-quality results. The configurations are the same for all problems in this experiment.

- *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)*: default configuration proposed in [27] and in the original source code, with initial $\sigma = 0.25$, $\mu = 50$, $\lambda = 100$.
- *Cuckoo Search (CS)*: default configuration proposed in [20] and in the original source code, $p_a = 0.25$, popsize = 50 (it is a dual-stage algorithm), nests = 25.

Table 4 Run and evolutionary parameter values for DSO in Experiment 1

Parameter	Value
C1, C2, C3	0.5, 0.4, 0.9
Teams	4
Reference perturbation	*rand*{/}1 using $F = C1$, *MVNS* + *Step*
Firmware update	every iteration
w (# of firmware updated)	1
Fixed perturbation	*rand* / 1 using $F = C1$
Stagnation	50 iterations
Elitism when stagnated	1
Prob. acc. worse sol. (P_{acc})	10%
Tree-size (min, max)	5, 20
pBest	25%
CR	*U*(0.4, 0.9)

Table 5 Run and evolutionary parameter values for DSO in Experiment 2, CEC 2005 problems

Parameter	Value
C1, C2, C3	0.5, 0.4, 0.9
Teams	4
Reference perturbation	*rand*/1 using $F = C1$, *MVNS* + *Step*
Firmware update	every iteration
w (# of firmware updated)	1
Fixed perturbation	*rand* / 1 using $F = C1$
Stagnation	50 iterations
Elitism when stagnated	1
Prob. acc. worse sol. (P_{acc})	10%
tree-size (min, max)	5, 20
pBest	25%
CR	*U*(0.4, 0.9)

- *Differential Evolution (DE)*: default configuration proposed in [56], $F = 0.5$, $CR = 0.9$, *rand*/1/*bin*.
- *Gravitational search algorithm (GSA)*: default configuration proposed in [51] and in the original source code.
- *Particle Swarm Optimization (PSO) with Constriction Factor*: default configuration proposed in [13]: $C1 = 2.05$, $C2 = 2.05$, $K = 0.729$, $vMax = 2$, $vMin = -2$.

All methods were configured to generate 100 solutions every iteration. It is important to notice that DSO has 4 teams of 25 members, giving the overall sample size of 100, while the other methods have 100 solutions used to generate 100 new solutions. A larger population may provide better exploration capability and increase the chances of finding the global optimum. Consequently, one could

Table 6 Results for Experiment 1 with $D = 5$

Fun	Measure	DSO	DE	CMA-ES	CS	GSA	PSO
f_1	Median	0.0000E+00	0.0000E+00	9.2242E−31 *	4.2581E−12 *	1.3716E−19 *	1.0888E−35 *
	SR	1	1	1	1	1	1
f_2	Median	0.0000E+00	9.1562E−72 *	1.7870E−15 *	2.8971E−06 *	7.1512E−10 *	1.5416E−19 *
	SR	1	1	1	0	1	1
f_3	Median	0.0000E+00	0.0000E+00 *	8.6736E−30 *	6.4733E−10 *	1.6316E−19 *	1.7250E−29 *
	SR	1	1	1	1	1	1
f_4	Median	0.0000E+00	5.3816E−68 *	6.6692E−16 *	1.4119E−04 *	2.9035E−10 *	2.9066E−17 *
	SR	1	1	1	0	1	1
f_5	Median	1.2173E−09	0.0000E+00	4.0800E−28 *	1.0930E−01 *	1.5620E+00 *	1.2884E−01 *
	SR	0.82	0.62	1	0	0	0
f_6	Median	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
	SR	1	1	1	1	1	1
f_7	Median	4.1239E−04	2.3274E−04 *	7.2166E−03 *	1.6539E−03 *	8.7901E−04 *	2.3624E−04 *
	SR	1	1	0.6	1	1	1
f_8	Median	1.1368E−12	1.1844E+02 *	1.1844E+02	4.6663E+00 *	1.1392E+04 *	2.3688E+02 *
	SR	1	0.32	0.48	0	0	0.08
f_9	Median	0.0000E+00	1.4924E+00 *	0.0000E+00 *	1.4631E−01 *	0.0000E+00 *	0.0000E+00 *
	SR	1	0.18	0.62	0	0.7	0.9
f_{10}	Median	8.8818E−16	4.4409E−15 *	1.9216E+01 *	1.1377E−03 *	7.4869E−10 *	4.4409E−15 *
	SR	1	1	0.44	0	1	1
f_{11}	Median	0.0000E+00	3.9423E−02 *	0.0000E+00 *	3.6040E−02 *	4.0790E−01 *	1.7236E−02 *
	SR	1	0.02	0.58	0	0	0.06
f_{12}	Median	1.1342E−30	9.4233E−32 *	5.7319E−31	9.8027E−09 *	7.6149E−21 *	9.4233E−32 *
	SR	1	1	1	0.5	1	1
f_{13}	Median	1.2191E−29	1.3498E−32 *	1.5137E−30	2.2228E−10 *	1.7697E−20 *	1.3498E−32 *
	SR	1	0.98	1	1	1	1
	$\overline{Rank}$	**1.7692**	2.0769	3.1538	4.7692	4.4615	2.6154
	$\overline{SR}$	**0.9862**	0.7785	0.8246	0.4231	0.7462	0.7723

The bold values indicate the best absolute values

suppose, in advance, that DSO is likely to get stuck into local optima more often than the other methods.

For experiment 2, DSO was slightly tuned to achieve better performance on the harder functions. We did only a few tests to find good parameter settings (see Table 5). Extensive tests could to find configurations able to outperform the other methods, but this is not the goal of this work.

The results are compared to those that were selected on that Special Session and are taken from Table 13 in [21], which considers only the average error for each problem. The optimization methods in the comparison are: Hybrid Real-Coded Genetic Algorithm with Female and Male Differentiation (BLX-GL50 [22]), Real-Coded Memetic Algorithm (BLX-MA [39]), Cooperative Evolution EA (CoEVO [46]), canonical Differential Evolution (DE [53]), Dynamic multi-swarm particle swarm optimizer with local search (DMS-L-PSO [34]), Estimation of Distribution Algorithm (EDA [66]), Covariance Matrix Evolution Strategy and Restarting method (G-CMA-ES [5]), Steady-State Evolutionary Algorithm (K-PCX [55]), Covariance Matrix Evolution Strategy Improved with Local Search (L-CMA-ES [4]), Self-adaptive differential evolution algorithm for numerical optimization (L-SaDE [48]), and Steady-State Genetic Algorithm (SPC-PNX [7]).

4.3 Performance comparison

For experiment 1, results are calculated over 50 independent runs and are organized by the problem's dimension (5, 10, 50, or 100). In the tables, they are presented in two rows per function, in terms of median error (row above) and success rate (row below). Since for some cases the resulting distributions are quite asymmetric, average and standard deviation would not allow a correct interpretation of the behavior. The two bottom lines in the tables are the average ranking ($\overline{Rank}$) of each optimization method and the average success performance ($\overline{SR}$). Error values below

Table 7 Results for Experiment 1 with $D = 10$

Fun	Measure	DSO	DE	CMA-ES	CS	GSA	PSO
f_1	Median	0.0000E+00	0.0000E+00*	4.8851E−30*	2.2845E−13*	4.2917E−19*	2.5748E−54*
	SR	1	0.96	1	1	1	1
f_2	Median	0.0000E+00	4.0618E−23*	5.4302E−15*	4.6081E−06*	2.0108E−09*	1.3797E−29*
	SR	1	0.82	1	0	1	1
f_3	Median	0.0000E+00	9.8852E−15*	1.1092E−28*	1.0214E−06*	8.5368E−19*	2.8589E−28*
	SR	1	0.72	1	0	1	1
f_4	Median	4.6349E−70	2.0988E−02*	1.2377E−15*	1.7950E−03*	4.5770E−10*	2.4699E−21*
	SR	1	0.04	1	0	1	1
f_5	Median	4.9087E−07	8.3357E+00*	2.6740E−27*	8.2580E−01*	5.4257E+00*	5.0054E−01*
	SR	0.34	0	1	0	0	0
f_6	Median	0.0000E+00	0.0000E+00*	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
	SR	1	0.54	1	1	1	1
f_7	Median	7.4231E−04	4.4823E−04*	1.0664E−02*	4.2571E−03*	1.5599E−03*	6.2642E−04
	SR	1	1	0.48	0.98	1	1
f_8	Median	7.7123E−10	7.5012E+02*	2.3688E+02*	5.1762E+02*	1.0836E+04*	9.5054E+02*
	SR	0.78	0	0.08	0	0	0
f_9	Median	0.0000E+00	1.1940E+01*	9.9496E−01*	6.7089E+00*	9.9496E−01*	3.9798E+00*
	SR	1	0	0.22	0	0.24	0.02
f_{10}	Median	8.8818E−16	7.9936E−15*	1.9237E+01*	7.8745E−03*	9.1242E−10*	4.4409E−15*
	SR	1	0.76	0.4	0	1	1
f_{11}	Median	0.0000E+00	1.0703E−01*	0.0000E+00	4.2075E−02*	0.0000E+00*	5.8998E−02*
	SR	1	0	1	0	0.56	0
f_{12}	Median	1.6634E−29	8.2686E−05	1.6592E−30	1.3139E−06*	7.3727E−21*	4.7116E−32*
	SR	1	0.4	1	0	1	1
f_{13}	Median	7.5251E−28	1.3498E−32*	8.1455E−30*	8.2380E−11*	3.7753E−20*	1.3498E−32*
	SR	1	0.82	1	1	1	1
	$\overline{Rank}$	**1.6154**	3.7692	2.9231	4.6923	3.8462	2.6154
	$\overline{SR}$	**0.9323**	0.4662	0.7831	0.3062	0.7538	0.6938

The bold values indicate the best absolute values

$1.0E-100$ were converted to $0.0E+00$. For the success rate calculation, the value-to-reach was $1E-02$ for f_7, and $1.0E-08$ for the remaining problems. The rank for each row is the ascending order of the methods with respect to the median error, where ties consider the minimum rank, i.e., if two methods present the lowest value ($0.0E+00$), then both get rank 1.

We also applied a two-sided Wilcoxon's rank sum test between DSO and each method used in the comparison. The null hypothesis is that the samples compared are independent, but from identical continuous distributions with equal medians. The symbol '*' was added whenever the null hypothesis was rejected at the significance level $\alpha = 0.05$, meaning that the lowest median is the best.

As the reader will notice, for some cases the null hypothesis was not rejected (there is no '*') although the medians (DSO *versus* Another method) were very distinct, meaning that the statistical test did not detect a significant difference between the samples. This is an important issue and the reason why we also present violin/dot plots for a visual analysis of the distributions and comment on the results.

For experiment 2, results are calculated over 36 runs (instead of 25 as the other methods) because DSO was executed on a cluster and we decided to use all available computing nodes. The rank analysis is first done by Friedman rank sum test. Then, when a statistical difference among the methods was detected, we employed the pair-wise post hoc test for Multiple Comparisons of Mean Rank Sums for unreplicated blocked data (Nemenyi's test) to identify differences between DSO (as the control method) and each method used in the comparison.

Computing time is not a criterion to be investigated here. The current DSO is slower than all methods used in comparison as it generates code (perturbation schemes) during the optimization process; the perturbation

Table 8 Results for Experiment 1 with $D = 50$

Fun	Measure	DSO	DE	CMA-ES	CS	GSA	PSO
f_1	Median	0.0000E+00	8.5187E+03*	1.2086E−28*	1.7082E−05*	5.9191E−18*	4.7641E−70*
	SR	1	0	1	0	1	1
f_2	Median	0.0000E+00	4.1127E+01*	8.6894E−14*	1.3322E+02*	1.5728E−08*	1.7298E−20*
	SR	1	0	0.64	0	0	1
f_3	Median	0.0000E+00	8.3081E+03*	7.5232E−26*	1.2226E+03*	4.5695E+01*	5.6020E−04*
	SR	1	0	1	0	0	0
f_4	Median	0.0000E+00	3.6649E+01*	5.8117E−15*	2.2843E+00*	1.1316E−09*	1.1277E−02*
	SR	1	0	1	0	1	0
f_5	Median	2.0293E−08	4.6132E+06*	1.0389E−25*	4.5175E+01*	4.0751E+01*	3.1881E+01*
	SR	0.46	0	0.84	0	0	0
f_6	Median	0.0000E+00	7.5502E+03*	1.3903E−28*	1.5081E−05*	0.0000E+00*	3.0815E−33*
	SR	1	0	1	0	1	1
f_7	Median	6.1744E−04	3.5498E−02*	2.8957E−01*	2.8254E−02*	6.4154E−03*	1.0604E+02*
	SR	1	0.02	0	0	0.96	0.02
f_8	Median	3.5531E+02	1.0351E+04*	8.7375E+03*	6.6746E+03*	1.7177E+04*	1.0677E+04*
	SR	0	0	0	0	0	0
f_9	Median	0.0000E+00	1.6661E+02*	3.7858E+02*	1.4430E+02*	9.9496E+00*	1.1790E+02*
	SR	1	0	0	0	0	0
f_{10}	Median	8.8818E−16	1.2631E+01*	1.9510E+01*	3.9461E+00*	1.4384E−09*	1.0271E+00*
	SR	1	0	0	0	1	0.44
f_{11}	Median	0.0000E+00	7.8454E+01*	0.0000E+00*	3.6353E−04*	0.0000E+00*	9.1868E−02*
	SR	1	0	0.96	0	0.72	0.16
f_{12}	Median	4.9290E−19	5.6869E+05*	2.7799E−29*	6.5139E−01*	2.2864E−20*	3.1101E−02
	SR	1	0	0.92	0	1	0.5
f_{13}	Median	8.4691E−18	7.6647E+06*	6.6538E−28*	6.8833E−03*	5.3259E−19	3.5068E−32
	SR	1	0	0.8	0	1	0.54
	$\overline{Rank}$	**1.692**	5.462	2.923	4.462	2.923	3.308
	$\overline{SR}$	**0.8815**	0.0015	0.6277	0	0.5908	0.3585

The bold values indicate the best absolute values

scheme might be evaluated (parsed and compiled) to provide the desired results. Once compiled, the perturbation scheme is as fast as the other codes. However, if the perturbation scheme is changed often, then a slowdown is clearly noticeable.

4.4 Experiment 1: results and discussion

Results of the experiments are shown in Tables 6, 7, and 8, and Figs. 2, 4, and 6. For $D = 5$, one can notice in Table 6 that DSO performs well for most problems, achieving the best solution for the majority of runs. Many problems, i.e., $f_1 - f_4, f_6, f_9, f_{11}$, reached a very small error. Function f_5 was solved to the optimum only by DE, while CMA-ES achieved 100% of success rate. Function f_7 was the hardest problem, with solutions around $1E-04$, but VTR is $1E-02$; thus, the results were close to the expected precision. DSO achieved both the best $\overline{Rank}$ and $\overline{SR}$. The worst method in both criteria was the Cuckoo Search (CS). One may also notice that significant differences were detected for most cases. When that occurs, but the medians are equal, one should check the success rate and also the violin/dot plot (Fig. 2) to choose the best solution. For instance, for function f_3, both DSO and DE had equal medians and success rate, but the violin/dot plot shows that DSO found lower-quality solutions in some runs, meaning that DE was better than DSO.

Regarding the performance curves in Fig. 3, one may observe that DSO was the fastest method for functions $f_1 - f_4, f_8, f_9$, and f_{11}. A characteristic that must be noted is the noise (peak) present in some of DSO's curves (see f_2 close to generation 200, f_3 close to generation 300). This noise happens because DSO restarts if stagnation is detected. When it restarts, the median solution quality gets worse.

With dimension $D = 10$ (see Table 7), the performance of the metaheuristics deteriorated and became less robust, with a wider distribution of the results (see Fig. 4). DSO achieved very high precision for functions $f_1 - f_4, f_6, f_9$, and

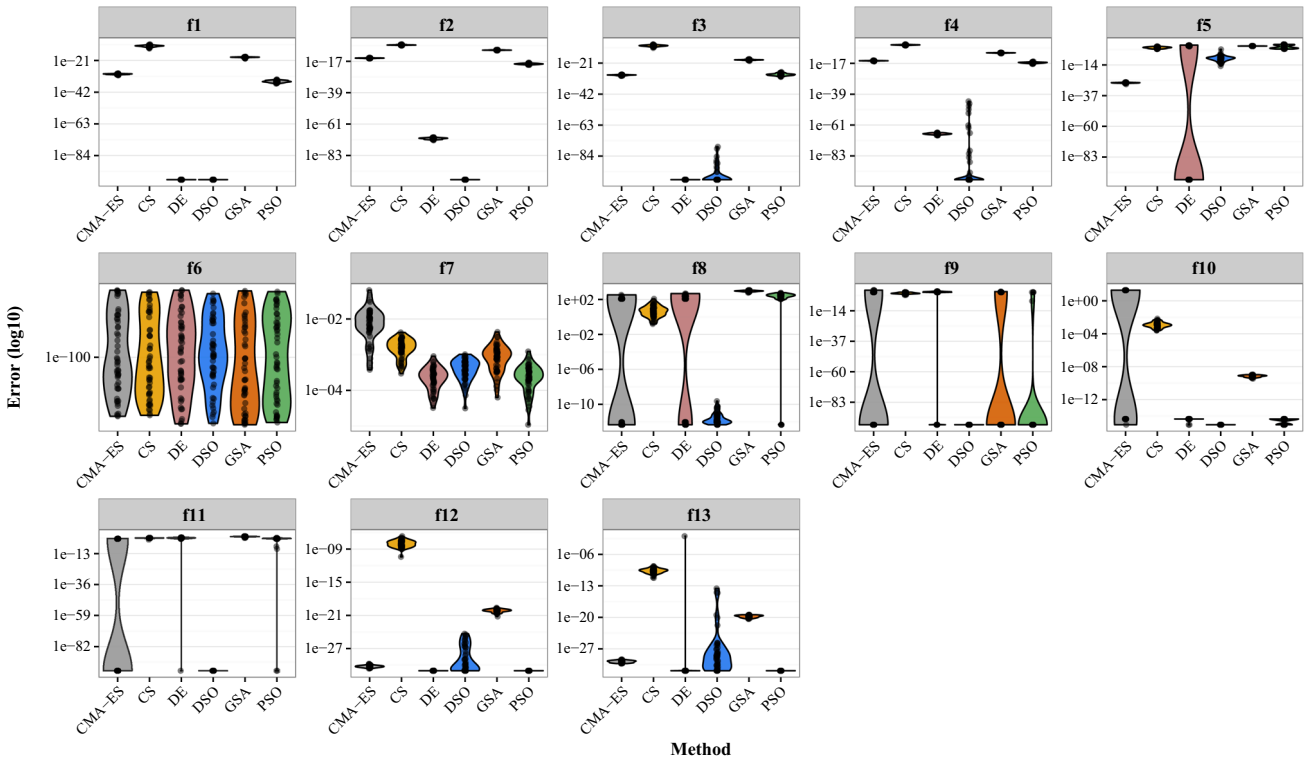

Fig. 2 *Violin/dot plots* of the error distribution (log-scale) for Experiment 1 with $D = 5$. DSO showed very good performance, except for f_5. The variance is DSO is mostly due to the different firmwares generated each run

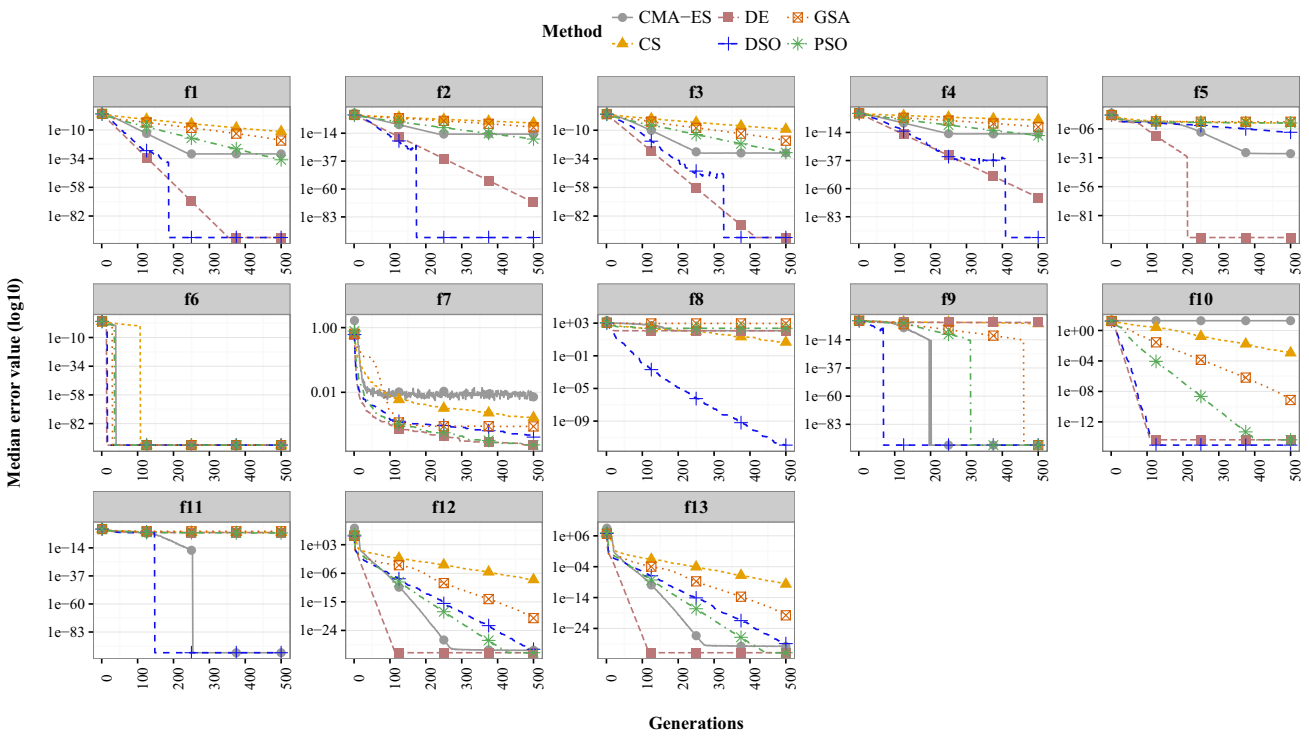

Fig. 3 *Median curves* of the error distribution (log-scale) for Experiment 1 with $D = 5$. DSO was the fastest and most precise method for most functions

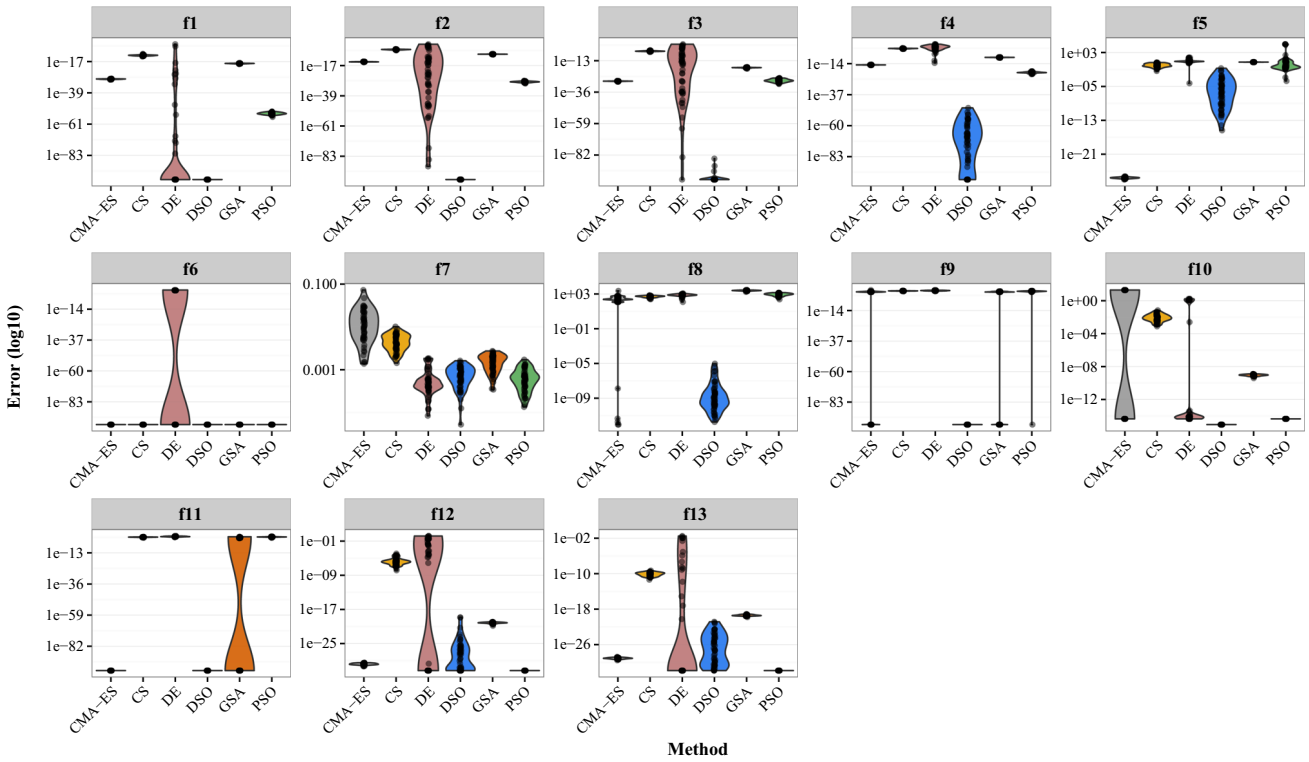

Fig. 4 *Violin/dot plots* of the error distribution (log-scale) for Experiment 1 with $D = 10$. DSO shows less performance degradation than the other methods when compared with $D = 5$

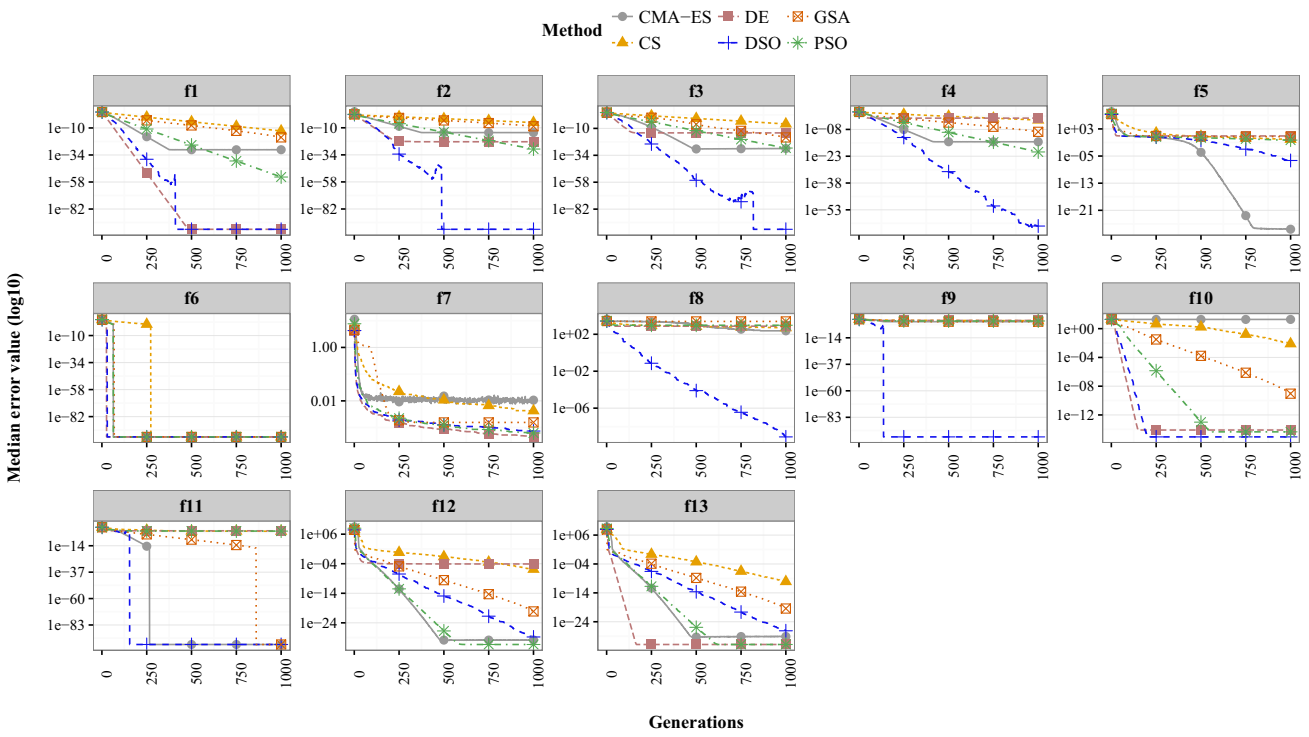

Fig. 5 *Median curves* of the error distribution (log-scale) for Experiment 1 with $D = 10$. DSO still shows the fastest performance for most problems

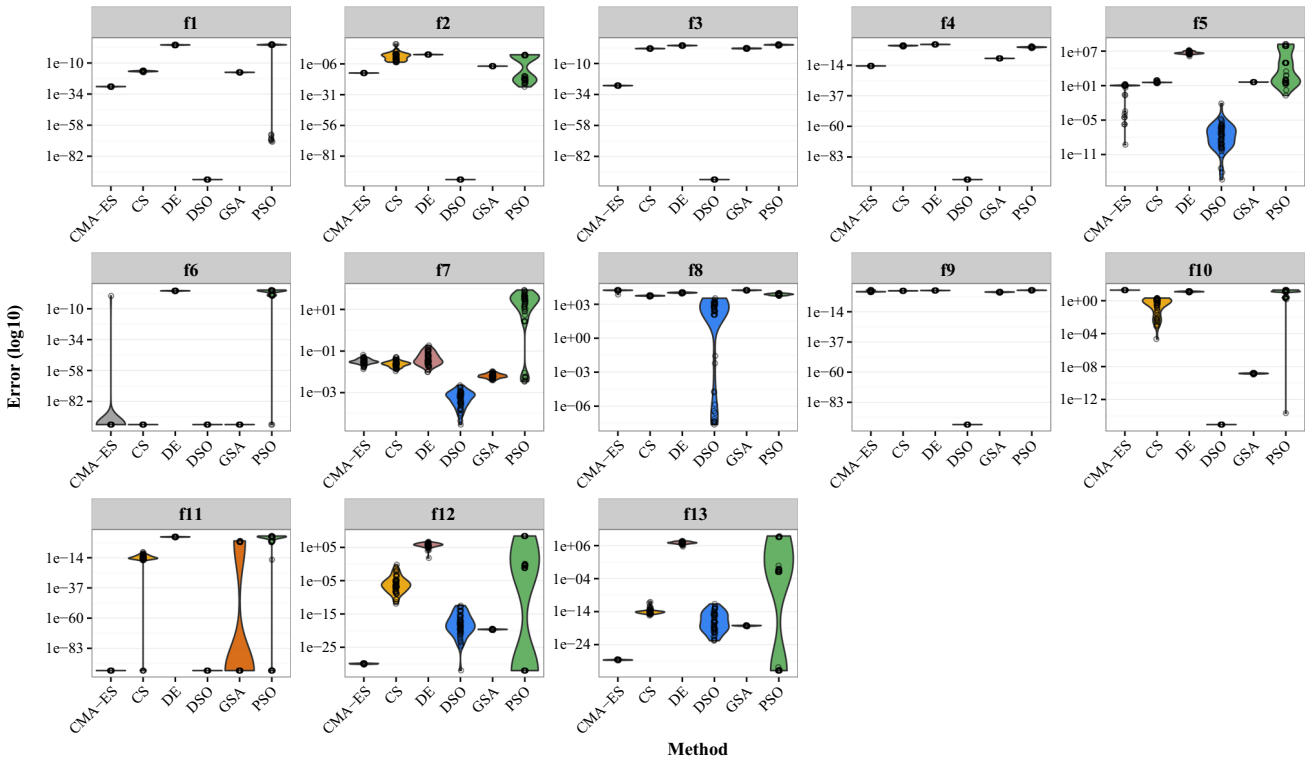

Fig. 6 *Violin/dot plots* of the error distribution (log-scale) for Experiment 1 with $D = 50$. The performance degradation of the other methods is more visible now, with a larger D

f_{11}. f_8 was also best solved by DSO, but with a substantial reduction in quality. Functions f_5 and f_{12} were best solved by CMA-ES, and PSO was the best method for functions f_6 (along with GSA) and f_{13}. Finally, DSO was the best method concerning both $\overline{Rank}$ and $\overline{SR}$.

In the next dimension considered ($D = 50$, Table 8), DSO maintained the quality for f_1–f_4,f_9, and f_{11}, presenting some deterioration in the other problems. However, DSO still showed better than median performance for functions f_5, f_8, f_{12}, and f_{13}. DSO found the best solutions for f_7 and f_{10}. Interestingly, while the other techniques suffered a large impact in f_7, mainly PSO, DSO maintained the best performance. Again, for f_6, the best method was GSA, while DSO was the best overall method w.r.t. both $\overline{Rank}$ and $\overline{SR}$. An issue for functions f_{12} and f_{13} was the comparison of DSO, GSA, and PSO, because the null hypothesis was not rejected even though the medians were clearly distinct. The success rate aids in understanding the phenomena for PSO, which achieved only 50% of success rate; thus, the other half of runs was trapped in low-quality optima. On the other hand, GSA presented an order of magnitude better median solution than DSO. However, DSO's results were spread (larger variance) as shown in the violin/dot plot (Fig. 6), achieving higher precision than GSA.

Finally, for $D = 100$, the median quality of the solutions achieved by DSO for f_1–f_4, f_6, f_7, f_9, f_{10}, and f_{11}, were still on par with those obtained for lower dimensions, that is, DSO was able to properly adapt to those functions. DSO found the best solutions for f_5 and f_{12} in some runs, but CMA-ES was the best for f_{12} and f_{13}. Once more, DSO was the best method when considering both $\overline{Rank}$ and $\overline{SR}$.

As summarized above, DSO was consistently the best method to solve functions f_1-f_4, f_7, f_9, f_{10}, and f_{11}. These functions, as explained before, have distinct characteristics that make some of them harder than the others. It is interesting to notice that even the "easier" separable unimodal problems (f_1 and f_2) were not properly solved by the other methods used in the comparison. That could be an issue of the method's configuration, but recommended settings from literature were employed here, using the open-source code provided by the authors of the methods (Fig. 5).

For many problems, DSO achieved much better results than the other algorithms. One of DSO's characteristics that could be used to explain the results is the convergence

Table 9 Results for Experiment 1 with $D = 100$

Fun	Measure	DSO	DE	CMA-ES	CS	GSA	PSO
f_1	Median	0.0000E+00	4.3174E+04SR	2.9680E−28SR	9.8542E−07SR	1.4943E−17SR	2.2944E−36SR
	SR	1	0	1	0	1	1
f_2	Median	0.0000E+00	1.4794E+02SR	7.5512E−10SR	1.0000E+10SR	3.4861E−08SR	1.2372E−10SR
	SR	1	0	0.58	0	0	0.78
f_3	Median	0.0000E+00	3.7035E+04SR	8.7114E−26SR	9.0556E+03SR	2.9561E+02SR	4.4198E+00SR
	SR	1	0	1	0	0	0
f_4	Median	0.0000E+00	5.0450E+01SR	1.0020E−14SR	5.1268E+00SR	2.0181E−09SR	7.1167E−01SR
	SR	1	0	1	0	1	0
f_5	Median	2.5036E−11	4.9444E+07SR	5.8845E−25SR	9.3117E+01SR	8.6467E+01SR	1.2285E+02SR
	SR	0.86	0	0.82	0	0	0
f_6	Median	0.0000E+00	4.5110E+04SR	3.5339E−28SR	8.6322E−07SR	0.0000E+00SR	4.9612E−31SR
	SR	1	0	1	0	1	1
f_7	Median	4.4336E−04	5.9559E+00SR	4.3644E−01SR	5.5418E−02SR	1.3808E−02SR	9.9994E+02SR
	SR	1	0	0	0	0.04	0
f_8	Median	1.0661E+03	2.5797E+04SR	1.7454E+04SR	1.5811E+04SR	3.6440E+04SR	2.2076E+04SR
	SR	0	0	0	0	0	0
f_9	Median	0.0000E+00	5.3275E+02SR	7.2433E+02SR	3.0480E+02SR	2.2884E+01SR	3.0200E+02SR
	SR	1	0	0	0	0	0
f_{10}	Median	8.8818E−16	1.6148E+01SR	1.9474E+01SR	3.1408E+00SR	1.6017E−09SR	2.6281E+00SR
	SR	1	0	0	0	1	0
f_{11}	Median	0.0000E+00	3.7882E+02SR	0.0000E+00SR	1.8750E−06SR	0.0000E+00SR	1.3338E−01SR
	SR	1	0	0.98	0	0.66	0.22
f_{12}	Median	1.0736E−16	2.0124E+07SR	1.2646E−29SR	4.8271E−01SR	2.9002E−20SR	3.1101E−02
	SR	1	0	0.92	0	0.94	0.48
f_{13}	Median	6.5661E−16	1.0805E+08SR	6.5108E−28SR	9.0035E−04SR	1.4986E−18SR	1.0987E−02SR
	SR	1	0	0.76	0	1	0.34
	$\overline{Rank}$	**1.615**	5.615	2.769	4.308	2.769	3.692
	$\overline{SR}$	**0.9123**	0	0.62	0	0.5108	0.2938

The bold values indicate the best absolute values

avoidance mechanism, the employment of information such as *Opposition*, *matInterval*, and scalings to force exploration. Another reason, though with a smaller impact, is the stagnation control that pauses the hill-climbing approach by accepting solutions that are worse than current ones (Table 9; Fig. 7).

With respect to average ranks, DSO was clearly the best method for $D = 10$, 50, and 100, but for $D = 5$ DE took the first position and DSO was the second best. With the increase in the number of variables, DE became the worst method, followed by CS. The opposite behavior is shown by CMA-ES, achieving better position in the ranks when D increased. An important point here is that DSO was configured with 4 teams, and the first one—that was set as constant, was a DE *rand* / 1, as shown in Table 4. Obviously, there are differences in the parameters (F, CR, crossover method); however, DE *rand*/1/*bin* performance is unlikely to be so largely improved and outperform CMA-ES and DSO. On the other hand, we may also argue that DSO suffered an impact in its performance by using DE *rand*/1 as the main reference team. Still, while the objective of this work is not to find the best reference, the current implementation is good enough to outperform some widely employed metaheuristics (Figs. 8, 9).

4.5 Experiment 2: results and discussion

In this section, we evaluate DSO on the benchmark problem set from CEC' 2005 Special Session in Real Parameter Optimization [57], with focus on the dimension $D = 10$. This problem set is known to be more challenging than the previous one. Thus, techniques that are good at solving that previous problem set may not necessarily perform well on

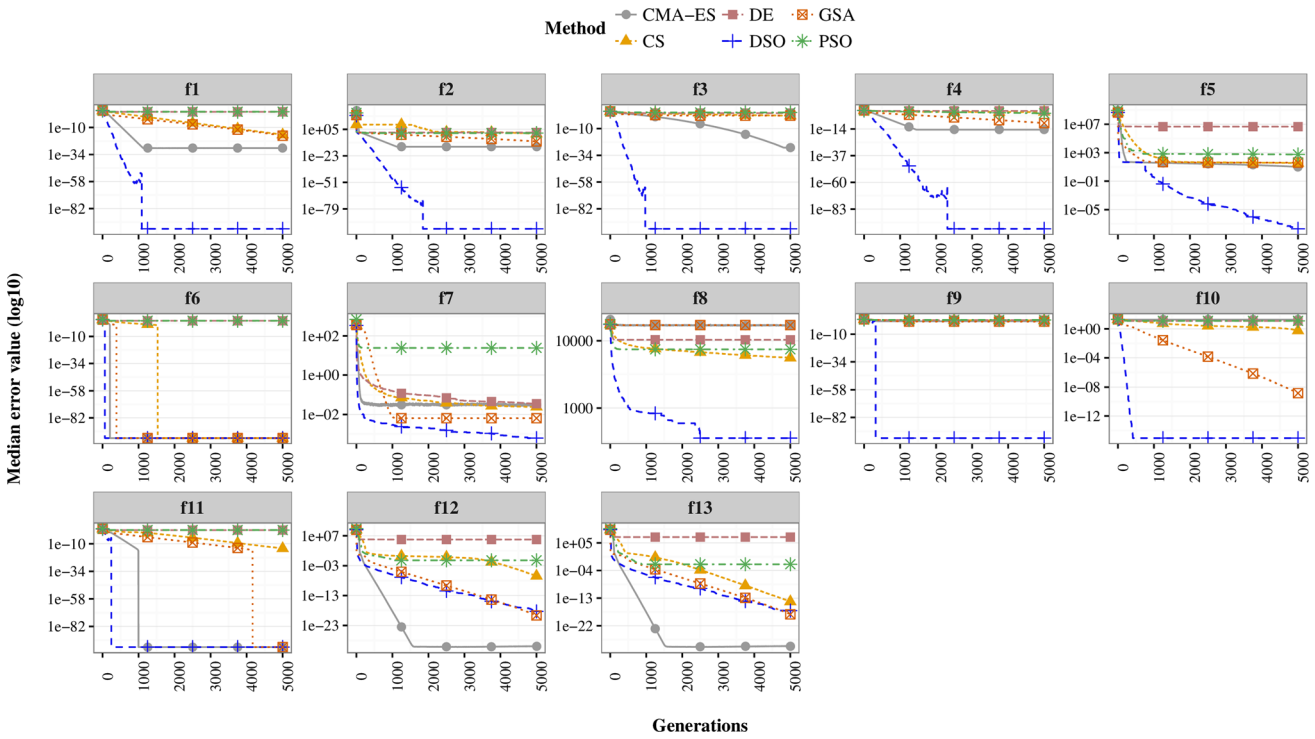

Fig. 7 *Median curves* of the error distribution (log-scale) for Experiment 1 with $D = 50$. DSO achieves even better performance now, losing only in functions f_{12} and f_{13}

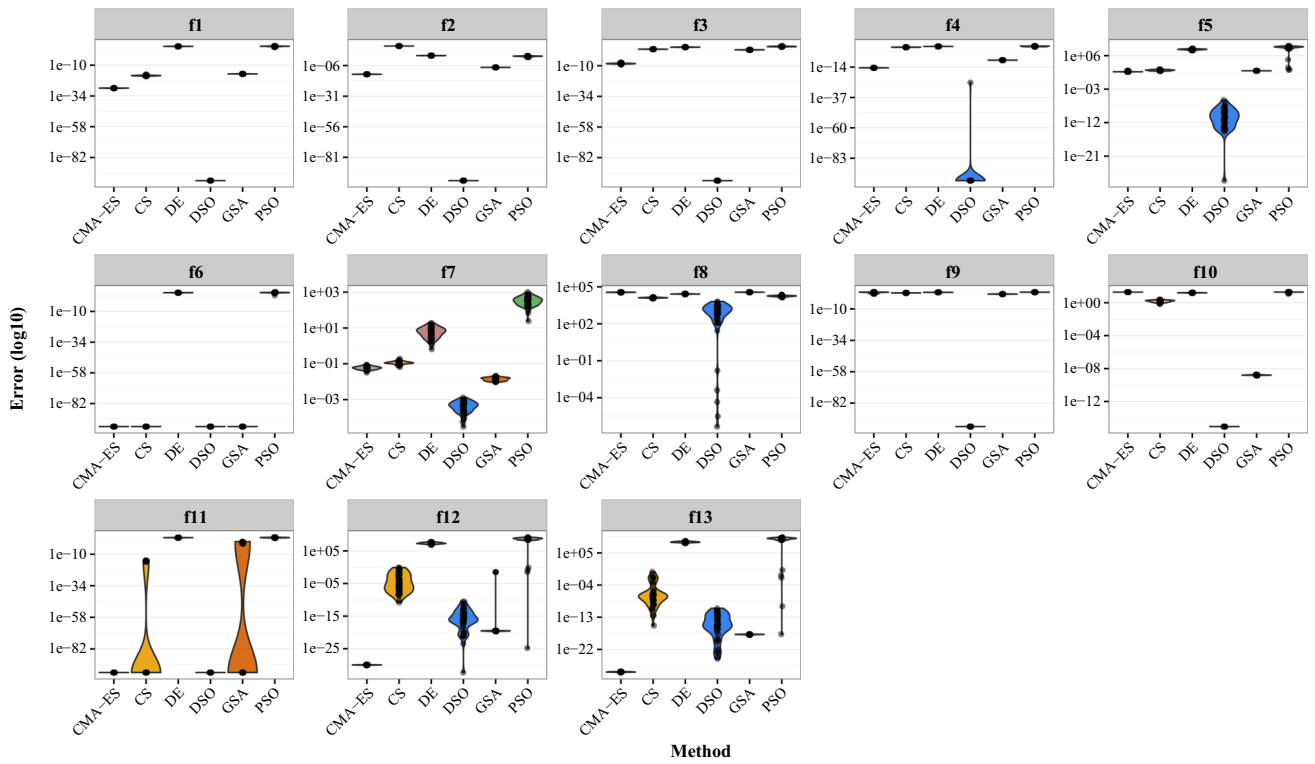

Fig. 8 *Violin plots* of the error distribution (log-scale) for Experiment 1 with $D = 100$. CMA-ES and GSA are still the only methods that outperformed DSO (at least partially), and only for problems f_{12} and f_{13}. The refinement process was considerably better for these two methods than for DSO (larger variance)

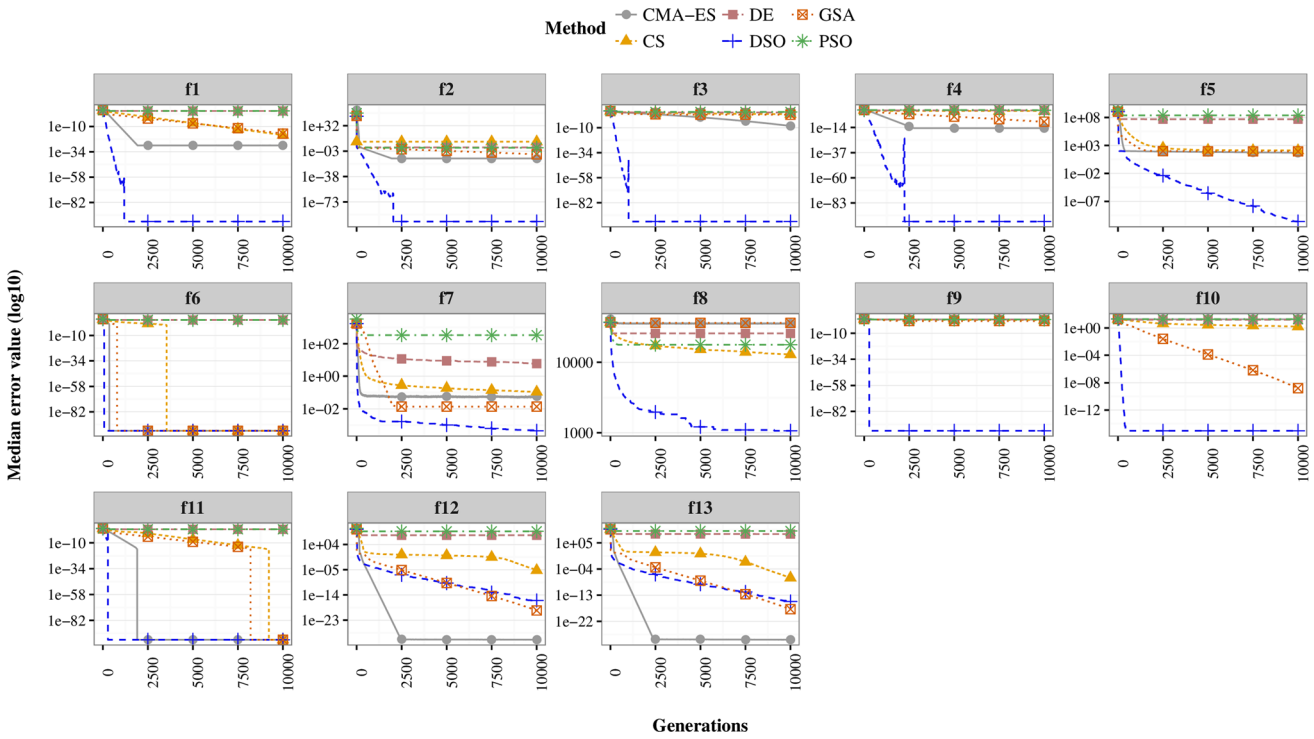

Fig. 9 *Median curves* of the error distribution (log-scale) for Experiment 1 with $D = 100$. Again, CMA-ES was substantially better than all other methods in functions f_{12} and f_{13}, but DSO was the best in all other functions

this one. The results, shown in Table 10, are compared to those selected on that Special Session and are taken from Table 13 in [21], which considers only the average error for each problem. A more detailed statistical description of the results is in "Appendix" (see Table 12).

To analyze the results, and since the original results are not available for running the Wilcoxon's test, we did run the Friedman rank sum test to compare the 12 techniques' averages, assuming significance level $\alpha = 0.05$. As the Friedman test indicates significant difference (χ^2 (11) = 51.8873, $p = 2.858e-07 < \alpha$), it is meaningful to conduct multiple comparisons in order to identify differences between the techniques on the 25 problems. For that purpose, we employed Nemenyi's post hoc test, and the comparisons regarding DSO's wins, ties, and losses, as well as the p value, are presented in Table 11.

Once again, we are investigating the hypothesis whether a method that self-generates perturbation schemes can outperform hand-made specialized methods that participated in a renowned competition. In this case, even ties could be considered a success for DSO as it generates perturbation schemes that produce poor quality (infeasible) solutions, meaning that a considerable number of evaluations is wasted.

On the one hand, concerning the W/T/L values, DSO had more wins than the methods BLX-MA, CoEVO, EDA, K-PCX, and SPC-PNX. On the other hand, as can be noticed by the p values, no significant differences ($p < \alpha$) were detected by the post hoc analysis for $\alpha = 0.05$, even though G-CMA-ES won 20 out of 25 times (lower average error), and DSO won 19 times when compared to CoEVO. The significance for G-CMA-ES versus DSO would be detected for $\alpha = 0.1$. The differences detected by Friedman's test are for G-CMA-ES as the control method, but Table 11 shows only DSO as the control method. Therefore, one may conclude that DSO is as good as the methods selected for that Special Session, although it is based on an online hyper-heuristic which sometimes generates very poor quality expressions that waste precious evaluations, while other methods are hand-made specialized developments.

One may also check the performance curves of DSO in "Appendix" section. Unfortunately, we don't have access to the curves of the other methods for comparison.

5 Summary, conclusions, and future works

Drone Squadron Optimization (DSO), introduced in this paper, is a novel method for global numerical optimization. It is an evolutionary algorithm, but not nature-inspired. Because it is artifact-inspired, it was developed to be a very flexible technique, allowing the insertion and removal of

Table 10 Average error obtained in CEC'2005 Special Session in dimension 10

Algorithm	f1	f2	f3	f4	f5	f6	f7
DSO	1.000E−09	1.000E−09	7.697E+01	1.000E−09	5.968E−03	3.322E−01	6.526E−02
BLX-GL50	1.000E−09	1.000E−09	***5.705E+02***	1.000E−09	1.000E−09	1.000E−09	1.172E−02
BLX-MA	1.000E−09	1.000E−09	***4.771E+04***	***1.997E−08***	***2.124E−02***	***1.490E+00***	***1.971E−01***
CoEVO	1.000E−09	1.000E−09	1.000E−09	1.000E−09	***2.133E+00***	***1.246E+01***	3.705E−02
DE	1.000E−09	1.000E−09	1.940E−06	1.000E−09	1.000E−09	1.590E−01	***1.460E−01***
DMS-L-PSO	1.000E−09	1.000E−09	1.000E−09	***1.885E−03***	1.138E−06	6.892E−08	4.519E−02
EDA	1.000E−09	1.000E−09	2.121E+01	1.000E−09	1.000E−09	4.182E−02	***4.205E−01***
G-CMA-ES	1.000E−09	1.000E−09	1.000E−09	1.000E−09	1.000E−09	1.000E−09	1.000E−09
K-PCX	1.000E−09	1.000E−09	4.150E−01	***7.940E−07***	***4.850E+01***	4.780E−01	***2.310E−01***
L-CMA-ES	1.000E−09	1.000E−09	1.000E−09	***1.760E+06***	1.000E−09	1.000E−09	1.000E−09
L-SaDE	1.000E−09	1.000E−09	1.672E−02	***1.418E−05***	***1.200E−02***	1.199E−08	2.000E−02
SPC-PNX	1.000E−09	1.000E−09	***1.081E+05***	1.000E−09	1.000E−09	***1.891E+01***	***8.261E−02***
Algorithm	f8	f9	f10	f11	f12	f13	f14
DSO	2.005E+01	4.698E−01	1.365E+01	3.830E+00	5.665E+01	4.984E−01	3.042E+00
BLX-GL50	***2.035E+01***	***1.154E+00***	4.975E+00	2.334E+00	***4.069E+02***	***7.498E−01***	2.172E+00
BLX-MA	***2.019E+01***	4.379E−01	5.643E+00	***4.557E+00***	***7.430E+01***	***7.736E−01***	2.030E+00
CoEVO	***2.027E+01***	***1.919E+01***	***2.677E+01***	***9.029E+00***	***6.046E+02***	***1.137E+00***	***3.706E+00***
DE	***2.040E+01***	***9.550E−01***	1.250E+01	8.470E−01	3.170E+01	***9.770E−01***	***3.450E+00***
DMS-L-PSO	2.000E+01	1.000E−09	3.622E+00	***4.623E+00***	2.400E+00	3.689E−01	2.360E+00
EDA	***2.034E+01***	***5.418E+00***	5.289E+00	***3.944E+00***	***4.423E+02***	***1.841E+00***	2.630E+00
G-CMA-ES	2.000E+01	2.390E−01	7.960E−02	9.340E−01	2.930E+01	***6.960E−01***	3.010E+00
K-PCX	2.000E+01	1.190E−01	2.390E−01	***6.650E+00***	***1.490E+02***	***6.530E−01***	2.350E+00
L-CMA-ES	2.000E+01	***4.490E+01***	***4.080E+01***	3.650E+00	***2.090E+02***	4.940E−01	***4.010E+00***
L-SaDE	2.000E+01	1.000E−09	4.969E+00	***4.891E+00***	4.501E−07	2.200E−01	2.915E+00
SPC-PNX	***2.099E+01***	***4.020E+00***	7.304E+00	1.910E+00	***2.595E+02***	***8.379E−01***	***3.046E+00***
Algorithm	f15	f16	f17	f18	f19	f20	f21
DSO	2.491E+02	1.222E+02	1.171E+02	5.983E+02	5.638E+02	6.324E+02	5.237E+02
BLX-GL50	***4.000E+02***	9.349E+01	1.090E+02	4.200E+02	4.490E+02	4.460E+02	***6.893E+02***
BLX-MA	***2.696E+02***	1.016E+02	***1.270E+02***	***8.033E+02***	***7.628E+02***	***8.000E+02***	***7.218E+02***
CoEVO	***2.938E+02***	***1.772E+02***	***2.118E+02***	***9.014E+02***	***8.445E+02***	***8.629E+02***	6.349E+02
DE	***2.590E+02***	1.130E+02	1.150E+02	4.000E+02	4.200E+02	4.600E+02	4.920E+02
DMS-L-PSO	4.854E+00	9.476E+01	1.101E+02	***7.607E+02***	***7.143E+02***	***8.220E+02***	5.360E+02
EDA	***3.650E+02***	***1.439E+02***	***1.568E+02***	4.832E+02	5.644E+02	6.519E+02	4.840E+02
G-CMA-ES	2.280E+02	9.130E+01	***1.230E+02***	3.320E+02	3.260E+02	3.000E+02	5.000E+02
K-PCX	***5.100E+02***	9.590E+01	9.730E+01	***7.520E+02***	***7.510E+02***	***8.130E+02***	***1.050E+03***
L-CMA-ES	2.110E+02	1.050E+02	***5.490E+02***	4.970E+02	***5.160E+02***	***4.420E+02***	***4.040E+02***
L-SaDE	3.200E+01	1.012E+02	1.141E+02	***7.194E+02***	7.049E+02	7.130E+02	4.640E+02
SPC-PNX	***2.538E+02***	1.096E+02	***1.190E+02***	4.396E+02	3.800E+02	4.400E+02	***6.801E+02***

Algorithm	f22	f23	f24	f25
DSO	7.316E+02	6.664E+02	2.083E+02	3.784E+02
BLX-GL50	***7.586E+02***	6.389E+02	2.000E+02	***4.036E+02***
BLX-MA	6.709E+02	***9.267E+02***	***2.240E+02***	***3.957E+02***
CoEVO	***7.789E+02***	***8.346E+02***	***3.138E+02***	2.573E+02
DE	7.180E+02	5.720E+02	2.000E+02	9.230E+02

Table 10 continued

Algorithm	f22	f23	f24	f25
DMS-L-PSO	6.924E+02	***7.303E+02***	***2.240E+02***	3.657E+02
EDA	***7.709E+02***	6.405E+02	2.000E+02	3.730E+02
G-CMA-ES	7.290E+02	5.590E+02	2.000E+02	3.740E+02
K-PCX	6.590E+02	***1.060E+03***	***4.060E+02***	***4.060E+02***
L-CMA-ES	***7.400E+02***	***7.910E+02***	***8.650E+02***	***4.420E+02***
L-SaDE	***7.349E+02***	6.641E+02	2.000E+02	3.759E+02
SPC-PNX	***7.493E+02***	5.759E+02	2.000E+02	***4.060E+02***

Results ***worse*** than DSO's (in absolute values) are in italic and bold-face

Table 11 Comparison of the averages: the values for Wins/Ties/Losses are for DSO versus the method

	BLX-GL50	BLX-MA	CoEVO	DE	DMS-L-PSO	EDA
W/T/L	9/3/13	18/2/5	19/3/3	7/3/15	8/2/15	12/3/10
p value	0.999	0.986	0.297	1.000	0.986	1.000
	G-CMA-ES	K-PCX	L-CMA-ES	L-SaDE	SPC-PNX	
W/T/L	2/3/20	15/2/8	10/2/13	7/2/16	13/3/9	
p value	0.083	1.000	1.000	0.969	1.000	

The p value was calculated by Nemenyi's post hoc test

components without disrupting the original concept. For instance, DSO may or may not apply recombination, and any kind of data recombination is permitted. On the other hand, a Genetic Algorithm without crossover may not be considered an actual Genetic Algorithm by some. More importantly, DSO is self-adaptive from its first version, because this is the role played by the command center.

In the first experiment, DSO was evaluated on 13 well-known continuous benchmark problems in dimensions $D = 5$, 10, 50, and 100 using four methodologies to analyze the results: 1) median of the final values with comparison via hypothesis test, 2) rank, 3) success rate given a value-to-reach, and 4) visual analysis of the distributions via violin/dot plots. In summary, while DSO's performance stays high for most problems when the problem's dimension D is increased, other methods used in this comparison showed a substantial reduction in the amount of good solutions found. Finally, DSO was the best approach for the majority of tests.

In the second experiment, DSO was evaluated on 25 well-known continuous benchmark problems (CEC 2005) in dimensions $D = 10$ and the results were compared to those obtained by the papers selected to appear in the special session. In the direct rank comparison, DSO was better than five out of 11 techniques. After the hypothesis test, no significant difference was found, meaning the DSO showed competitive performance. However, it is important to remember that DSO is a technique that itself evolves during the optimization, meaning that it will generate poor perturbation schemes and waste a substantial amount of objective function evaluations.

The DSO method introduced here uses random distributions to generate scalings, to select methods for recombination and bound correction, to select lower-quality solutions when stagnation occurs, to update the firmware, among other aspects. We intend to employ learning procedures to reduce such randomness avoid non-promising regions of the search space. Other future work includes the investigation of other procedures to generate solutions and the inclusion of the recombination procedures into the firmware.

In this paper, DSO was evaluated in single-objective unconstrained optimization problems. We intend to investigate improvements to develop a multi-objective version of the method, probably starting with the Pareto front strategy, but looking for innovative procedures. Regarding constrained optimization, we will evaluate DSO's performance with the adaptive penalty function proposed in a previous work for the CMA-ES algorithm [16].

Acknowledgements This paper was supported by the Brazilian Government CNPq (Universal) Grant (486950/2013-1) and CAPES (Science without Borders) Grant (12180-13-0) to V.V.M., and Canada's NSERC Discovery Grant RGPIN 283304-2012 to W.B.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix 1: More detailed explanation on DSO

Departure points

1. *CBC*: the matrix of current best solutions found so far;
2. *PermutedCBC*: the current best solutions found so far, but permuted every iteration. Because of the permutation, *CBC* can be combined with other solutions even using the same firmware;
3. CBC_{pBest}: the *pBest* solutions found, where p is a user-defined percentage parameter. The selected solutions are sampled with repetition to create a matrix with N solutions;
4. Multivariate normal sampling (MVNS): new random solutions sampled using the average and covariance matrix of the *pBest* solutions found;
5. *Opposition*(*CBC*): the opposed coordinates of the Current Best ones, calculated as proposed in [50].

Offset

The movements from the departure points are generated applying scaling and functions to other coordinates and information. These items are presented below.

1. Constants: $\overrightarrow{matInterval} = (\overrightarrow{UB} - \overrightarrow{LB})$ and user-defined values C1, C2, and C3, where $\overrightarrow{matInterval}$ is an array of size D;
2. Random weights: $U(0, 1)$, $U(0.5, 1)$, $G(0, 1)$, $abs(G(0.5, 0.1))$, and $abs(G(0, 0.01))$, where U is the uniform distributions, and G the Gaussian distribution;
3. Calculated weights: $\text{std} - \text{dev}(CBC)$, $\text{std} - \text{dev}(CBC_{pBest})$, and $Step(CBC) = \sigma * G(0, 1)_{N,D} * \overrightarrow{matInterval} * U(0, 0.5)$, as used in [16], where N is the number of drones in a team;
4. Two-parameter functions: *plus*, *times*, *sub*, *protected division*, *average*, where protected division returns $Numerator/((1e-15) + Denominator)$.
5. *TmC*: the best positions found by the teams after calculating the target coordinates;
6. *Shift*: the difference between *TmC* and *CBC*, that is, how much the drones have to move;
7. $\overrightarrow{GBC}$: the best solution found so far;
8. $Opposition(CBC_{pBest})$: the opposed position of the *pBest* current best coordinates.

Reference perturbation

The command center is instructed to set the initial firmware with at least one reference perturbation. This directive is to avoid starting with teams using completely random perturbation. The two reference perturbations available for this DSO are:

1. $\overrightarrow{CBC_{r1}} + c_1 * (\overrightarrow{CBC_{r2}} - \overrightarrow{CBC_{r3}})$, and
2. $MVNS + Step(CBC)$;

where r_1, r_2, and r_3 are random and distinct solutions. Therefore, the two reference perturbations are 1) *rand* / 1 from the Differential Evolution (but not linked to a particular crossover), and 2) *inspired* by the CMA-ES technique, but employing only sample generation and step calculation with $\sigma = 0.04 \times \mu_{eff} \times ||\mu||$. This formula is from the CMA-ES author's source code and was not tuned to be used in DSO.

Recombination

After the perturbation step generates new coordinates, a recombination with the current coordinates, representing the best coordinates found so far, may be done. Three possibilities are available:

1. No recombination;
2. Uniform crossover [GA] / Binomial recombination [DE];
3. One or two-point crossover [GA] / Exponential recombination [DE].

In the current DSO, recombination is performed after perturbation, but changing the order is also an option. That will change the behavior of the method without invalidating the original inspiration.

Coordinates correction (bounds)

The drones may be allowed to move only inside a particular perimeter. Therefore, if the new target coordinates (x) are outside the perimeter then a correction must be made. Three techniques are available:

1. The coordinate is re-positioned exactly over the bound;
2. The coordinate gets a new random value inside the feasible bounds;
3. The coordinate gets the remainder of the excess, that is $LB_j + remainder$ or $UB_j - remainder$, for $j = 1, \ldots, D$.

Appendix 2: More detailed results for CEC'05

See Figs. 10, 11 and Table 12.

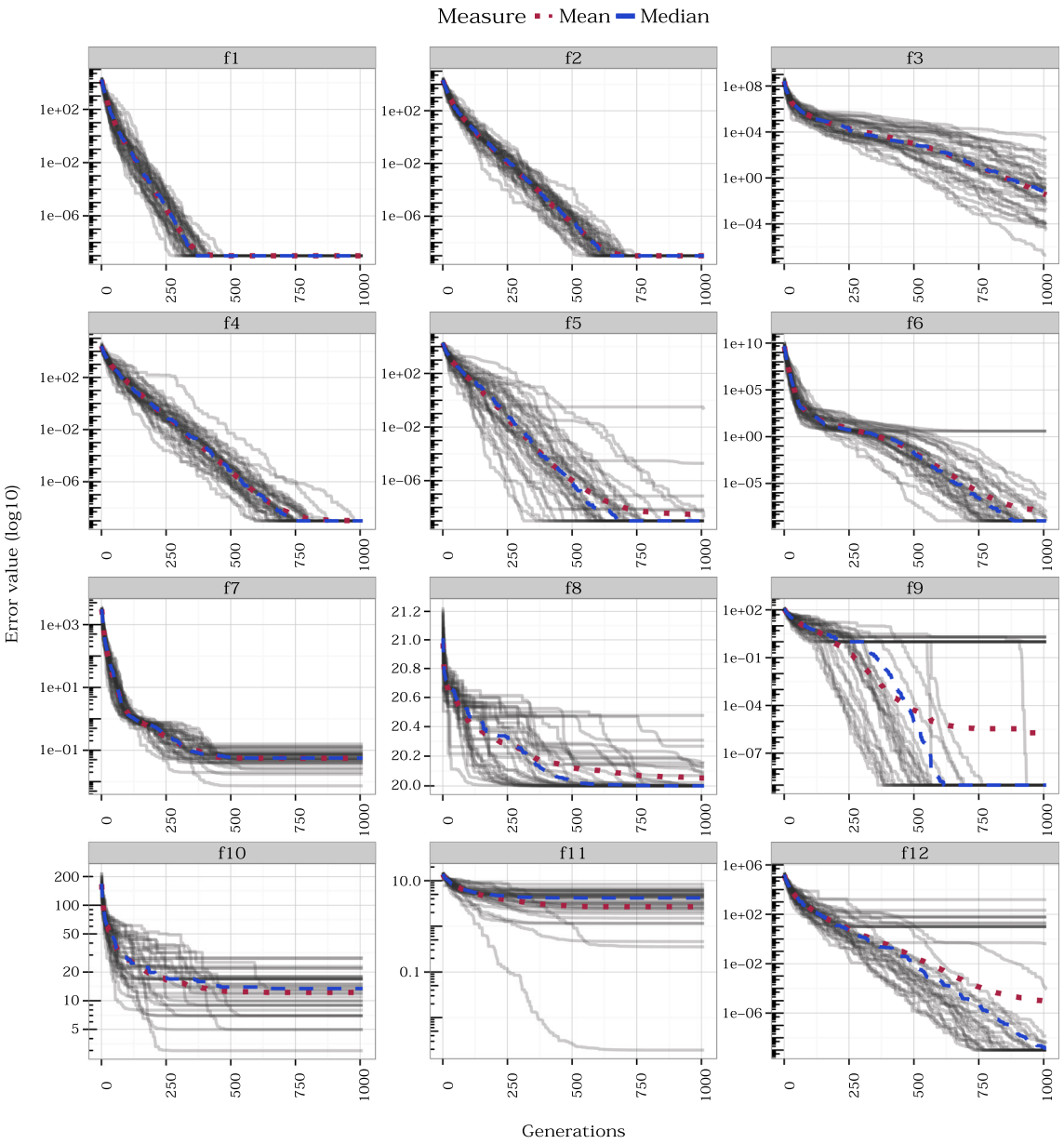

Fig. 10 DSO curves for functions f1–f12 (CEC'05)

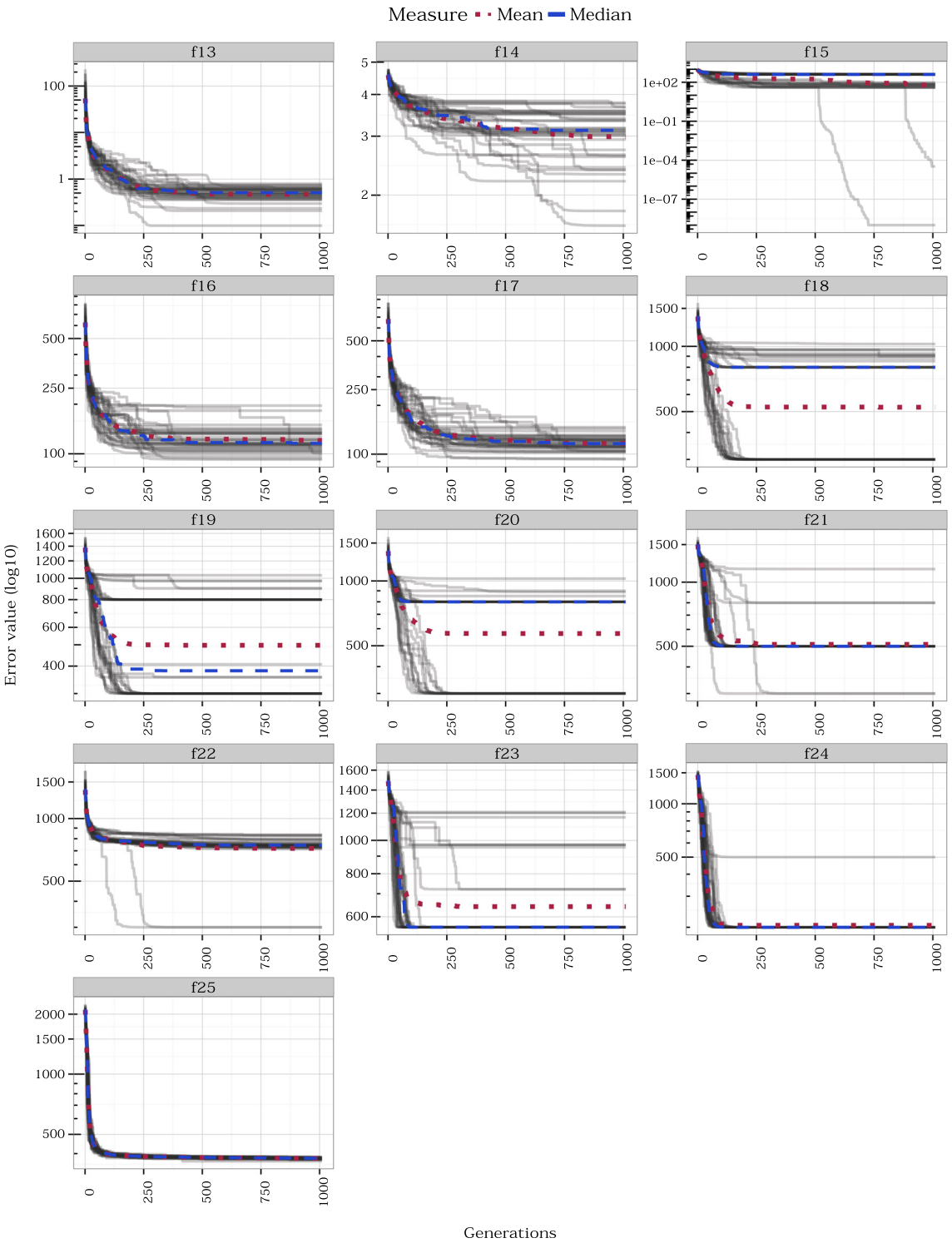

Fig. 11 DSO curves for functions f13–f25 (CEC'05)

Table 12 DSO's results obtained in CEC'2005 Special Session in dimension 10, over 36 independent trials

Algorithm	f1	f2	f3	f4	f5	f6	f7
Min	0.000E+00	7.086E−21	1.903E−07	9.389E−17	0.000E+00	1.037E−19	7.396E−03
Median	0.000E+00	4.061E−16	4.994E−02	5.241E−14	0.000E+00	4.470E−12	5.662E−02
Max	4.077E−19	6.021E−14	2.415E+03	1.512E−10	2.148E−01	3.987E+00	1.573E−01
Mean	1.132E−20	4.027E−15	7.697E+01	5.723E−12	5.968E−03	3.322E−01	6.526E−02
Std-dev	6.795E−20	1.191E−14	4.027E+02	2.574E−11	3.580E−02	1.117E+00	3.435E−02
SR	1	1	0.028	1	0.944	0.917	0.028
Algorithm	f8	f9	f10	f11	f12	f13	f14
Min	2.000E+01	0.000E+00	2.985E+00	1.920E−03	1.673E−15	9.897E−02	1.620E+00
Median	2.000E+01	0.000E+00	1.343E+01	4.180E+00	1.148E−09	5.084E−01	3.125E+00
Max	2.048E+01	1.990E+00	2.786E+01	8.340E+00	1.557E+03	7.971E−01	3.780E+00
Mean	2.005E+01	4.698E−01	1.365E+01	3.830E+00	5.665E+01	4.984E−01	3.042E+00
Std-dev	1.060E−01	6.929E−01	6.164E+00	2.014E+00	2.599E+02	1.554E−01	5.519E−01
SR	0	0.639	0	0.028	0.639	0	0
Algorithm	f15	f16	f17	f18	f19	f20	f21
Min	0.000E+00	9.231E+01	9.287E+01	3.000E+02	3.000E+02	3.000E+02	3.000E+02
Median	4.000E+02	1.158E+02	1.161E+02	8.000E+02	3.818E+02	8.000E+02	5.000E+02
Max	4.282E+02	1.960E+02	1.468E+02	1.025E+03	1.033E+03	1.025E+03	1.152E+03
Mean	2.491E+02	1.222E+02	1.171E+02	5.983E+02	5.638E+02	6.324E+02	5.237E+02
Std-dev	1.798E+02	2.214E+01	1.236E+01	2.909E+02	2.764E+02	2.569E+02	1.378E+02
SR	0.056	0	0	0	0	0	0

Algorithm	f22	f23	f24	f25
Min	3.000E+02	5.595E+02	2.000E+02	3.662E+02
Median	7.442E+02	5.595E+02	2.000E+02	3.790E+02
Max	8.352E+02	1.208E+03	5.000E+02	3.838E+02
Mean	7.316E+02	6.664E+02	2.083E+02	3.784E+02
Std-dev	1.113E+02	2.074E+02	5.000E+01	3.694E+00
SR	0	0	0	0

Success rate considers the following precision accuracy: $1.0E-06$ for f1–f6, $1.0E-02$ for f6–f16, and $1.0E-01$ for f17–f25. Results in this table were not cut in the $1E-09$ threshold as in Table 10

References

1. Ahmadi S-A (2016) Human behavior-based optimization: a novel metaheuristic approach to solve complex optimization problems. Neural Comput Appl 27:1–12
2. Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462
3. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC2005). Canberra, 8-12. IEEE Press, New York, pp 1769–1776
4. Auger A, Hansen N (2005) Performance evaluation of an advanced local search evolutionary algorithm. In: The 2005 IEEE congress on evolutionary computation, vol 2. IEEE, pp 1777–1784
5. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC2005), vol 2. IEEE, pp 1769–1776
6. Bäck T (1998) An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundam Inf 35(1–4):51–66
7. Ballester PJ, Stephenson J, Carter JN, Gallagher K (2005) Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC2005), pp 498–505
8. Banzhaf W (2001) Artificial intelligence: Genetic programming. In: Smelser NJ, Baltes PB (eds) International encyclopedia of the social & behavioral sciences. Pergamon, Oxford, pp 789–792
9. Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming—an introduction: on the automatic evolution of computer programs and its applications. Dpunkt–Verlag and Morgan Kaufmann, New York
10. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Number 1. Oxford University Press, Oxford
11. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR (2010) A classification of hyper-heuristics approaches. In: Handbook of metaheuristics, volume 57 of international series in

operations research and management science, 2nd edn, chap 15. Springer, Berlin, pp 449–468
12. Chakraborty UK (2008) Advances in differential evolution. Springer Publishing Company Incorporated, New York
13. Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6(1):58–73
14. Das S, Nagaratnam Suganthan P (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
15. de Melo VV, Luiza CCG (2013) Automatic generation of evolutionary operators: a study with mutation strategies for the differential evolution. In: Proceedings of the 28th annual ACM symposium on applied computing, SAC '13, Coimbra, Portugal, March 18–22, 2013, pp 188–193
16. De Melo VV, Iacca G (2014) A modified covariance matrix adaptation evolution strategy with adaptive penalty function and restart for constrained optimization. Expert Syst Appl 41(16):7077–7094
17. Dorigo M (1992) Optimization, learning and natural algorithms. Ph. D. Thesis, Politecnico di Milano, Italy
18. Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science, vol 1. New York, pp 39–43
19. Fister I Jr, Yang X-S, Fister I, Brest J, Fister D (2013) A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186
20. Gandomi A, Yang X-S, Alavi A (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 29(1):17–35. doi:10.1007/s00366-011-0241-y
21. García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. J Heuristics 15(6):617–644
22. García-Martínez C, Lozano M (2005) Hybrid real-coded genetic algorithms with female and male differentiation. In: The 2005 IEEE congress on evolutionary computation, vol 1. IEEE, pp 896–903
23. Glover Fred W, Kochenberger Gary A (2006) Handbook of metaheuristics, vol 57. Springer Science & Business Media, New York
24. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
25. Haklı H, Uğuz H (2014) A novel particle swarm optimization algorithm with levy flight. Appl Soft Comput 23:333–345
26. Hansen N (2009) Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In: Proceedings of the 11th annual conference companion on genetic and evolutionary computation: late breaking papers. ACM, New York, pp 2389–2396
27. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
28. Hansen N, Müller SD, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evol Comput 11(1):1–18
29. Hong L, Woodward J, Li J, Özcan E (2013) Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In: Krzysztof K, Alberto M, Ting H, Sima E-UA, Bin H (eds) genetic programming, vol 7831 of Lecture Notes in Computer Science. Springer, Berlin, pp 85–96
30. Dervis K, Bahriye B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 39(3):459–471
31. Kelly JP (1996) Meta-heuristics: theory and applications. Kluwer, Norwell
32. Krishnanand KN, Ghose D (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intell 3(2):87–124
33. Larrañaga P, Lozano JA (2001) Estimation of distribution algorithms: a new tool for evolutionary computation. Kluwer, Norwell
34. Liang JJ, Suganthan PN (2005) Dynamic multi-swarm particle swarm optimizer with local search. In: The 2005 IEEE congress on evolutionary computation, vol 1, pp 522–528. IEEE
35. Lin Shyh-Chang, Punch WF, III, Goodman Erik D (1994) Coarse-grain parallel genetic algorithms: categorization and new approach. In: Sixth IEEE symposium on parallel and distributed processing, 1994. Proceedings, pp 28–37. IEEE
36. Michalewicz Z, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4:1–32
37. Miranda PB, Prudêncio RB (2015) Gefpso: a framework for PSO optimization based on grammatical evolution. In: Proceedings of the 17th annual conference on genetic and evolutionary computation. ACM, New York, pp 1087–1094
38. Mirjalili S (2016) Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput Appl 27(4):1053–1073
39. Molina D, Herrera F, Lozano M (2005) Adaptive local search parameters for real-coded memetic algorithms. In: The 2005 IEEE congress on evolutionary computation, vol 1. IEEE, pp 888–895
40. O'Neill M, Brabazon A (2006) Grammatical differential evolution. In: IC-AI, pp 231–236
41. Osaba E, Diaz F, Onieva E, Carballedo R, Perallos A (2014) Amcpa: a population metaheuristic with adaptive crossover probability and multi-crossover mechanism for solving combinatorial optimization problems. Int J Artif Intell 12(2):1–23
42. Pavlidis NG, Tasoulis DK, Plagianakos VP, Vrahatis MN (2006) Human designed vs. genetically programmed differential evolution operators. In: Proceedings of the 2006 IEEE congress on evolutionary computation (CEC2006), pp 1880–1886
43. Peng F, Tang K, Chen G, Yao X (2010) Population-based algorithm portfolios for numerical optimization. IEEE Trans Evol Comput 14(5):782–800
44. Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S, Zaidi M (2006) The bees algorithm—a novel tool for complex optimisation problems. In: Proceedings of the 2nd virtual international conference on intelligent production machines and systems (IPROMS 2006), pp 454–459
45. Poli R, Langdon WB, Holland O (2005) Extending particle swarm optimisation via genetic programming. In: Proceedings of the 8th European conference on genetic programming, EuroGP'05. Springer, Berlin, pp 291–300
46. Pošik P (2005) Real-parameter optimization using the mutation step co-evolution. In: The 2005 IEEE congress on evolutionary computation, vol 1. IEEE, pp 872–879
47. Price KV, Storn RM, Lampinen JA (2005) Differential evolution a practical approach to global optimization. Natural computing series. Springer, Berlin
48. Qin AK, Suganthan PN (2005) Self-adaptive differential evolution algorithm for numerical optimization. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC2005), vol 2. IEEE, pp 1785–1791
49. Rachlin J, Goodwin R, Murthy S, Akkiraju R, Wu F, Kumaran S, Das R (1999) A-teams: an agent architecture for optimization and decision-support. In: Intelligent agents V: agents theories, architectures, and languages. Springer, Berlin, pp 261–276
50. Rahnamayan S, Tizhoosh HR, Salama MMA (2008) Opposition-based differential evolution. IEEE Trans Evol Comput 12(1):64–79

51. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2009) Gsa: a gravitational search algorithm. Inf Sci 179(13):2232–2248
52. Rashid M, Baig AR (2008) Adaptable evolutionary particle swarm optimization. In: 3rd international conference on innovative computing information and control, 2008. ICICIC '08, pp 602–602
53. Rônkkônen J, Kukkonen S, Price KV (2005) Real-parameter optimization using the mutation step coevolution. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC 2005), pp 506–513
54. Si T, De A, Bhattacharjee AK (2014) Grammatical swarm based-adaptable velocity update equations in particle swarm optimizer. In: Proceedings of the international conference on frontiers of intelligent computing: theory and applications (FICTA) 2013. Springer, Berlin, pp 197–206
55. Sinha A, Tiwari S, Deb K (2005) A population-based, steady-state procedure for real-parameter optimization. In: The 2005 IEEE congress on evolutionary computation, vol 1. IEEE, pp 514–521
56. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Opt 11(4):341–359
57. Suganthan PN, Hansen N, Liang JJ, Deb K, Chen Y-P, Auger A, Tiwari S (2005) Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL Report, 2005005
58. Tasoulis DK, Pavlidis NG, Plagianakos VP, Vrahatis MN (2004) Parallel differential evolution. In: Congress on evolutionary computation, 2004. CEC2004, vol 2. IEEE, pp 2023–2029
59. Thierens D (2005) An adaptive pursuit strategy for allocating operator probabilities. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation, pages 1539–1546. ACM, New York
60. Voß T, Hansen N, Igel C (2010) Improved step size adaptation for the MO-CMA-ES. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. ACM, New York, pp 487–494
61. Vrugt JA, Robinson BA, Hyman JM (2009) Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans Evol Comput 13(2):243–259
62. Whitacre JM, Pham TQ, Sarker RA (2006) Credit assignment in adaptive evolutionary algorithms. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. ACM, New York, pp 1353–1360
63. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
64. Woodward JR, Swan J (2012) The automatic generation of mutation operators for genetic algorithms. In: Proceedings of the 14th annual conference companion on genetic and evolutionary computation. ACM, New York, pp 67–74
65. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3(2):82–102
66. Yuan B, Gallagher M (2005) Experimental results for the special session on real-parameter optimization at CEC 2005: a simple, continuous EDA. In: Proceedings of the 2005 IEEE congress on evolutionary computation (CEC2005), vol 2005. IEEE, pp 1792–1799
67. Zhan Z-H, Zhang J, Li Y, Chung HS-H (2009) Adaptive particle swarm optimization. IEEE Trans Syst Man Cybern Part B 39(6):1362–1381 (Cybernetics)
68. Zhang J, Sanderson AC (2009) Jade: adaptive differential evolution with optional external archive. IEEE Trans Evol Comput 13(5):945–958