Soft Computing
https://doi.org/10.1007/s00500-024-09896-5

OPTIMIZATION

A co-evolutionary algorithm with adaptive penalty function for constrained optimization

Vinícius Veloso de Melo[1] · Alexandre Moreira Nascimento[2] · Giovanni Iacca[3]

Accepted: 8 May 2024

Abstract
Several constrained optimization problems have been adequately solved over the years thanks to the advances in the area of metaheuristics. Nevertheless, the question as to which search logic performs better on constrained optimization often arises. In this paper, we present Dual Search Optimization (DSO), a co-evolutionary algorithm that includes an adaptive penalty function to handle constrained problems. Compared to other self-adaptive metaheuristics, one of the main advantages of DSO is that it is able auto-construct its own perturbation logics, i.e., the ways solutions are modified to create new ones during the optimization process. This is accomplished by co-evolving the solutions (encoded as vectors of integer/real values) and perturbation strategies (encoded as Genetic Programming trees), in order to adapt the search to the problem. In addition to that, the adaptive penalty function allows the algorithm to handle constraints very effectively, yet with a minor additional algorithmic overhead. We compare DSO with several algorithms from the state-of-the-art on two sets of problems, namely: (1) seven well-known constrained engineering design problems and (2) the CEC 2017 benchmark for constrained optimization. Our results show that DSO can achieve state-of-the-art performances, being capable to automatically adjust its behavior to the problem at hand.

Keywords Co-evolutionary algorithm · Swarm intelligence, Constrained optimization · Self-adaptation · Engineering design

1 Introduction

Constrained optimization, i.e., the search for an optimal solution to a given optimization problem, subject to one or more constraints, is a key element in many applications. Examples of constrained optimization can be found in various areas of engineering (De Melo and Carosio 2013; Kashan 2011; Zhang et al. 2008; Wang et al. 2009), where in most cases the systems at hand have specific physical or economical requirements that may limit the feasibility of the possible solutions.

Without loss of generality, in this kind of problems the goal is to find $\mathbf{x}$ that minimizes (or maximizes) a given $f(\mathbf{x})$, subject to:

$$h_i(\mathbf{x}) = 0 \; i = 1 \ldots m$$
$$g_i(\mathbf{x}) \leqslant \epsilon \; i = 1 \ldots p$$

where $f(\mathbf{x})$ is the objective function to be optimized, $\mathbf{x} \in \mathbb{R}^D$ is a solution vector $\mathbf{x} = [x_1 \ldots x_D]^{\mathrm{T}}$, ϵ is the tolerance factor (that in this paper we set to zero, unless specified otherwise), and $h_i(\mathbf{x})$ and $g_i(\mathbf{x})$ are the equality and inequality constraints, respectively, with m and p being their number. In addition to these constraints, each variable x_k, $k = 1 \ldots D$ is usually bounded by lower and upper bounds $LB_i \leq x_k \leq UB_k$, which define the search space. In general, the solution may contain both integer and continuous variables, and the equality and inequality constraints can be either linear or non-linear. It is important to note that, in general, the presence of constraints makes the optimization process more difficult,

✉ Giovanni Iacca
giovanni.iacca@unitn.it

Vinícius Veloso de Melo
vinicius.melo@nasdaq.com

Alexandre Moreira Nascimento
alexandre.nascimento@sloan.mit.edu

[1] Nasdaq Verafin, Windsor, ON, Canada

[2] Institute of Science and Technology, Universidade Federal de São Paulo, UNIFESP, São José dos Campos, São Paulo 12231-280, Brazil

[3] Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 9, 38123 Povo, TN, Italy

Published online: 26 July 2024

since the region of the search space where the feasible solutions lie can very narrow compared to the total volume of the search space, and as such hard to find.

Much research effort in the last three decades has been put in the attempt of defining efficient Computational Intelligence algorithms for solving constrained optimization problems. Broadly speaking, this research has focused on two directions, namely: (1) the definition of specific constraint handling techniques (CHTs), such as those surveyed in (Coello Coello 2002; Mezura-Montes and Coello Coello 2011; Michalewicz 1995; Salcedo-Sanz 2009) and (2) the incorporation of such CHTs into various algorithmic schemes falling under the broad umbrella of Evolutionary Computation and Swarm Intelligence, or hybrids thereof. A complete survey of this vast research area is out of the scope of this paper, however in the following we briefly summarize some of the main achievements in the field, to provide the context of this work.[1]

Evolutionary Computation: The first works in this area date back to the late '90 and early 2000, when research focused especially on Genetic Algorithms (GA) (Coello Coello and Mezura-Montes 2002; Coello Coello 2000; Michalewicz et al. 1996; Rasheed 1998). More recently, while some researchers have focused on specific CHTs for GA (Long 2014) and applications thereof, such as in Gutérrez-Antonio et al. (2011), most of the research attention has shifted towards Differential Evolution (DE) Das et al. (2016) and Evolution Strategies (ES) Mezura-Montes and Coello Coello (2005). Among DE-based algorithms, it is worth mentioning two proposals by Mezura-Montes et al. (2006, 2004), who introduced the use of feasibility rules and diversity preservation mechanisms specifically designed for constrained optimization. Another powerful algorithm, named ε-DE, was introduced and further extended in Takahama and Sakai (2006, 2010, 2012); Takahama and Sakai (2013). This algorithm ranks the solutions such that infeasible solutions are considered only if they violate the constraints at most by a given ε-value. A number of works have devised self-adaptive variants of DE (De Melo and Carosio 2013; Huang et al. 2006; Mezura-Montes et al. 2010; Mezura-Montes and Palomeque-Ortiz 2009a, b; Zou et al. 2011) or adaptive penalty functions (Ali and Zhu 2013; de Melo and Carosio 2012). As for ES, the current literature focuses mostly on Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and variants thereof. For instance, adaptive penalty functions for CMA-ES have been proposed in (Collange et al. 2010; de Melo and Iacca 2014; de Melo and Iacca 2014), whereas Arnold and Hansen (2012) proposed a (1+1)-CMA-ES that, starting from a feasible solution, adapts the covariance matrix to decrease the likelihood of sampling infeasible solutions. Other algorithms are based on multiple rankings (Kusakci and Can 2013), or surrogate models (Gieseke and Kramer 2013; Kramer et al. 2009, 2013) specifically meant for expensive optimization. Finally, a recent work has evaluated the effectiveness of a diversity-driven evolutionary algorithm, named MAP-Elites, for solving constrained optimization problems (Fioravanzo and Iacca 2019).

Swarm Intelligence: Most of the works falling in this category are based on Particle Swarm Optimization (PSO) and several of its variants (Aguirre et al. 2007; Cagnina et al. 2008; Coelho, L.d.S, 2010; He and Wang 2007; Hu et al. 2003; Pant et al. 2009; Machado-Coelho et al. 2017; Guedria 2016; Mazhoud et al. 2013). However, more recently also other Swarm Intelligence paradigms have been applied to solve constrained optimization, for instance Ant Colony Optimization (ACO) Leguizamón and Coello Coello (2009), Cuckoo Search (CS) Yang and Deb (2010), Artificial Bee Colony (ABC) (Akay and Karaboga 2012; Brajevic et al. 2011), Social Spider Optimization (SSO) Cuevas and Cienfuegos (2014), Artificial Immune System (AIS) Zhang et al. (2014), Crow Search Algorithm (CSA) Askarzadeh (2016), Gravitational Search Algorithm (GSA) Purcaru et al. (2013), and Simulated Societies (Ray and Liew 2003).

Hybrid algorithms: Lastly, it is worth mentioning the recent research on constrained optimization tackled by hybrid algorithms, for instance Memetic Computing (MC) algorithms (Barkat Ullah et al. 2011; Noman and Iba 2008; Pescador Rojas and Coello 2012). Some hybrid algorithms combine GA and AIS (Bernardino et al. 2007), PSO and Simulated Annealing (SA) He and Wang (2007), PSO and GA (Takahama et al. 2005), PSO and DE (Liu et al. 2010), or (1+1)-CMA-ES and DE (Maesani et al. 2016). Other studies have proposed the use of gradient-based information (Hamza et al. 2014; Handoko et al. 2010; Sun and Garibaldi 2010), which however can only be applied when the mathematical formulation of the optimization problem is available and its functions are differentiable. Of note, this idea is also used in one of the most complex hybrid algorithms from the literature, which combines gradient-based mutations with PSO, DE, CMA-ES, and ε-constrained optimization (Bonyadi et al. 2013).

Here, we contribute to this research field by proposing a self-adaptive metaheuristic we refer to as Dual Search Optimization (DSO), which couples a co-evolutionary algorithm with an adaptive penalty function for solving constrained optimization problems. The main idea of the proposed algorithm is that a population of candidate solutions evolves to solve the optimization problem, however this evolution is not conducted based on predefined perturbation logics (i.e., how solutions are modified to generate new ones, e.g. mutation and crossover in Evolutionary Algorithms, or position update rules in Particle Swarm Optimization), but rather it is based

[1] A constantly updated list of references on the topic, maintained by Carlos A. Coello Coello, is available at: http://www.cs.cinvestav.mx/~constraint/.

on evolved ones. In particular, the algorithm evolves groups of solutions (encoded as vectors of integer/real values), and each group is associated to a perturbation logic (encoded as a Genetic Programming tree), which in turn evolves at runtime. This form of co-evolution of solutions and perturbation logics allows the algorithm to explore the search space according to different dynamic strategies, that self-adapt to the problem at hand. This provides a powerful means for solving black-box constrained optimization problems. From this point of view, this work pushes forward the study of self-adaptive (and auto-constructive) algorithmic structures, one of the most recent trends in the metaheuristics research field. Finally, the adaptive penalty function allows the DSO algorithm to handle constraints very effectively, yet with a minor algorithmic overhead.

As shown in the experimental section of the paper, we have evaluated DSO on seven well-known constrained engineering design problems, as well as the CEC 2017 benchmark for constrained optimization (Wu et al. 2017). For both problem sets, we performed a comparative analysis which includes a large body of computational intelligence optimization algorithms from the most recent literature. Our results show that DSO can achieve in most cases state-of-the-art performances, with the additional advantage of being capable to automatically adjust its behavior to the problem at hand.

The paper is organized as follows: In Sect. 2, we introduce the proposed algorithm. Section 3 describes the experimental setup, i.e., the tested problems, the configuration of the algorithm, and the computational environment. Section 4 presents the numerical results. Finally, in Sect. 5 we draw the conclusions of the paper and we mention some possible future research directions.

2 Proposed algorithm

The common idea of co-evolutionary frameworks is to have two (or more) kinds of entities that evolve jointly in order to solve a certain problem. This concept has been explored so far in various ways in the Evolutionary Computing community. For instance, an early paper by Schmidt and Lipson introduced the idea of co-evolving solutions and *fitness predictors*, i.e., a kind of dynamic fitness model used to approximate especially expensive fitness computations (Schmidt and Lipson 2008); more recently, Sipper et al. introduced a framework named Solution and Fitness Evolution (SAFE), in the attempt to co-evolve solutions and fitness functions (Sipper et al. 2019). In this direction, the same authors have made another step forward, co-evolving representations and encoding (i.e., a genotype-phenotype mapping) (Sipper and Moore 2019). Co-evolution has also been applied to agent-based reinforcement learning, as a means to create a *curriculum* of increasingly more difficult environments in order to evolve the controller of a bipedal walker (Wang et al. 2019). Lately, a first attempt to co-evolve solutions and perturbation logics was made in de Melo and Banzhaf (2017), who devised an "artifact-inspired" algorithm for solving unconstrained optimization problems. Motivated by these successes, here we continue the research line initiated in de Melo and Banzhaf (2017), by introducing an adaptive penalty function into a co-evolutionary algorithm specifically meant for constrained optimization. The goal is to co-evolve solutions and perturbation logics in a way that the algorithm solves a certain constrained optimization problem *while it learns to do so*.

It should be noted that so far co-evolutionary concepts have been adopted in various forms in the field of constrained optimization. However, to the best of our knowledge, no prior work has proposed a co-evolution of solutions and perturbation logics for solving constrained optimization problems. The related works on co-evolutionary concepts applied to constrained optimization have indeed explored quite different directions. Some works focused on problem decomposition: this idea led, for instance, to the surrogate-assisted memetic algorithm with random decomposition proposed in Goh et al. (2011), and to the Comprehensive Learning Particle Swarm Optimizer (CLPSO) hybridized with Sequential Quadratic Programming (SQP) proposed in Liang et al. (2010). Algorithms based on problem decomposition have been devised also for solving specific problems, such as train timetabling (Kwan and Mistry 2003) and Dynamic Constraint Satisfaction Problems (DCSPs) Handa et al. (1999). More recently, a decomposition-based approach has been proposed in the area of large-scale constrained optimization (Xu et al. 2021), while in Cai et al. (2022) a knowledge-based dynamic variable decomposition has been presented. Another interesting recent work (Hu et al. 2023) proposed combining deep learning with DE, with the deep neural network supporting the DE for the choice of suitable parents and corresponding archives for mutation.

Other works focused on the co-evolution of populations optimizing objective function and constraints separately. This is the case for instance, of CCS (Co-evolutionary approach to Constraint Satisfaction) proposed in Paredis et al. (1994), or the DE-based co-evolutionary approach proposed in Liu et al. (2007) and later extended in Liu et al. (2007) (with the inclusion of a memetic component based on Gaussian mutation), where two independent populations, one evaluated only in terms of objective functions, and one in terms of constraints, are evolved and mixed through migrations. A similar DE-based dual population scheme has been explored also in Gao et al. (2014). Older attempts at using dual population schemes were focused instead on evolving separately one population handling linear constraints and another one handling all (including non-linear) constraints (Michalewicz and Nazhiyath 1995). In more recent works (Liu et al. 2022; Bao et al. 2023), dual population approaches have been

proposed instead for solving Constrained multi-objective optimization problems (CMOPs), while other attempts have been made at using more than two populations (Ming et al. 2022).

Other authors have introduced custom co-evolutionary mutations (Kou et al. 2009), or proposed algorithms based on co-evolving sub-populations (Guo et al. 2007; Sergienko and Semenkin 2010). However, the largest part of the related literature focuses on co-evolution of solutions and penalty factors, e.g. based on PSO (He and Wang 2007; He et al. 2008) or DE (Fz et al. 2007; Fan and Yan 2012), or co-evolution of solutions and Lagrangian multipliers, achieved e.g. through GA (Barbosa 1999; Tahk and Sun 2000), PSO (Krohling and dos Santos Coelho 2006; Mohemmed et al. 2010), or DE (Ghasemishabankareh et al. 2016), and applied e.g. to constrained min-max problems, as in Kim and Tahk (2001).

The proposed algorithm, that we dub as Dual Search Optimization (DSO), is based on a Genetic Programming (GP) algorithm (Koza 1993). In DSO, a structured (i.e., divided in groups) population of candidate solutions evolves during the optimization process. Furthermore, each group of solutions is associated to a perturbation logic, that is a GP individual, which is in turn evolved during the optimization process.

All solutions within each group are updated with the same perturbation logic, while different groups are updated with different perturbation logics. The perturbation logic encodes the procedure to perform the search, i.e., each group has its own particular way of "moving" from the current set of solutions to new ones (target solutions). At each iteration of the algorithm, each solution in each group calculates its target solution, based on its own group's perturbation logic, and collects the information to be used in the GP part of the algorithm, i.e., a computation of the objective function and the constraint violations at the target solution. It should be noted that even though the solutions within each group share the same perturbation logic, they are, in general, different, so that they can reach different target solutions. On the other hand, even though some solutions from different groups may be the same, due to the different perturbation logics they may reach different target solutions.

The update of the perturbation logics can be activated based on different criteria, such as with a certain periodicity (as we did in our experiments, see below), a convergence or diversity metric, or some other condition on the population. In any case, the most important characteristic of DSO is that its GP part generates the code of the perturbation logics on-the-fly. Furthermore, the GP part of the algorithm can learn from the search (by using the information on the objective function and constraint violations), in order to generate better perturbation logics at each iteration.

Let us now discuss more in detail the algorithm, whose complete pseudo-code is shown in Algorithm 2. The main structures that the DSO maintains are the set of the current best solutions (i.e., the best solution found by each element of the population), **CBC**, and the related objective functions and aggregated constraint violations, respectively **CBOFV** and **CBCV**. It then uses the perturbation logic associated to each group to generate a set of trial solutions (which are not evaluated), **TC**, which in turn are used to generate the target solutions, **TmC**, for which the associated objective functions and aggregated constraint violations, respectively **TmCV** and **TmOFV**, are calculated. All the symbols mentioned in what follows are summarized in Table 1.

The generation of the target solutions in **TmC** proceeds as follows. Each solution in each group computes its trial solution (**trial**) by applying its group's perturbation logic (P), which is based on the following general scheme:

$$P = \textbf{base} + Offset() \tag{1}$$

$$\textbf{trial} = calculate(P) \tag{2}$$

where **base** is a *base solution* and $Offset()$ is a function that returns the actual perturbation movement. The GP part of DSOmodifies the base solution and the $Offset()$ function (thus a perturbation logic P) differently for each group, to enable different ways of performing the search. For instance, two possible perturbation logics might be:

$$P_1 : \textbf{GBC} + c \times (\textbf{GBC} - \textbf{CBC}_{group,solution}) \tag{3}$$

$$P_2 : \textbf{CBC}_{group,solution} + G(0,1) \times (\sqrt{U(0,1,N,D)} + \textbf{CBC}_{group,solution}) \tag{4}$$

where $\textbf{CBC}_{group,solution}$ indicates the best solution of the focal element in the focal group, c is a user-defined constant, $G(0,1)$ is a scalar value sampled from a Gaussian distribution with zero mean and unit standard deviation, and $U(0,1,N,D)$ returns an $N \times D$ matrix of numbers sampled from a uniform distribution in $[0,1]$. This added noise due to $G()$ and $U()$ is used to avoid that the search converges towards the origin of the search space. Here it can be seen that the two expressions differ in the **base** solutions, respectively **GBC** and $\textbf{CBC}_{group,solution}$, and in the $Offset()$ functions.

Since generating effective perturbations from scratch is hard (as also noted in de Melo and Banzhaf (2017)), the GP part of the algorithm is initialized with a set of predefined *reference perturbations* (in our experiments we used just one, see Table 6) that are used as initial perturbations for all groups, instead of randomly generated expressions. During the optimization process, these perturbations are then modified by means of GP to better adapt to the problem at hand.

To further improve the exploitation capabilities of DSO, after each solution applies its perturbation logic to generate

Table 1 Main symbols and acronyms used in the paper

Term	Meaning
D	Number of dimensions (variables) of the optimization problem
FES	Function evaluations
FES_{curr}	Current number of FES
FES_{max}	Maximum number of FES
$GBCV$	Global best constraint violation value
$GBOFV$	Global best objective function value
$Iter_{curr}$	Current number of iterations
$Iter_{max}$	Maximum number of iterations
$Iter_{stagnation}$	Maximum number of iterations without improvement
N	Number of solutions in each group
P	Perturbation logic
P_{acc}	Probability of accepting a solution worse than the ones in **CBC**
std	Standard deviation
t	Number of groups
w	Number of worst perturbation logics to be replaced
BoundsDiff	Array obtained by $\mathbf{UB} - \mathbf{LB}$ (size: $1 \times D$)
BoundsSum	Array obtained by $\lvert\mathbf{UB} + \mathbf{LB}\rvert$ (size: $1 \times D$)
CBC	Tensor containing the current best solutions (size: $t \times N \times D$)
CBCV	Matrix containing the aggregated constraint violation of each solution in **CBC** (size: $t \times N$)
CBOFV	Matrix containing the objective function values of each solution in **CBC** (size: $t \times N$)
GBC	Array containing the global best solution, i.e., the best solution found so far (size: $1 \times D$)
LB	Array containing the problem lower bounds (size: $1 \times D$)
TC	Tensor containing the trial solution for each solution in each group (size: $t \times N \times D$)
TQ	Array containing the quality of the groups (size: $1 \times t$)
TmC	Tensor containing the target solution for each solution in each group (size: $t \times N \times D$)
TmCV	Matrix containing the aggregated constraint violation of each solution in **TmC** (size: $t \times N$)
TmOFV	Matrix containing the objective function values of each solution in **TmC** (size: $t \times N$)
UB	Array containing the problem upper bounds (size: $1 \times D$)

the trial solution ($\mathbf{TC}_{group,solution}$), it generates the target solution $\mathbf{TmC}_{group,solution}$ by performing either a recombination of $\mathbf{TC}_{group,solution}$ with the global best solution found so far (GBC), or no recombination at all (in this case $\mathbf{TmC}_{group,solution} = \mathbf{TC}_{group,solution}$). Three recombination procedures (including no recombination) are available, with the same probability of being selected:

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

2.1 Boundary constraint handling

Another important aspect of DSO is the way the boundary violations are handled. In fact, as recently outlined in Nordmoen et al. (2020); Kononova et al. (2020), this is a fundamental aspect that can heavily affect the performance of a search algorithm. In DSO, the target solutions **TmC** are corrected by a given pool of correction techniques, randomly selected with the same probability, whenever there are boundary violations (i.e., the target solutions fall outside the lower or upper bounds). Three techniques are available:

1. The out-of-bound j-th element of the solution gets the value of the (closest) corresponding bound, that is: if $\mathbf{x}_j > \mathbf{UB}_j$, then $\mathbf{x}_j = \mathbf{UB}_j$; if $\mathbf{x}_j < \mathbf{LB}_j$, then $\mathbf{x}_j = \mathbf{LB}_j$.
2. The out-of-bound j-th element of the solution gets a new random value uniformly sampled inside the corresponding bounds, that is: if $\mathbf{x}_j > \mathbf{UB}_j$ or $\mathbf{x}_j < \mathbf{LB}_j$, then $\mathbf{x}_j = U(\mathbf{LB}_j, \mathbf{UB}_j)$.
3. The out-of-bound j-th element of the solution gets the value of the (closest) corresponding bound, plus the remainder of the division of the element by $|\mathbf{UB}_j - \mathbf{LB}_j|$, that is: if $\mathbf{x}_j > \mathbf{UB}_j$, then $\mathbf{x}_j = \mathbf{UB}_j - rem(\mathbf{x}, |\mathbf{UB}_j - \mathbf{LB}_j|)$; if $\mathbf{x}_j < \mathbf{LB}_j$, then $\mathbf{x}_j = \mathbf{LB}_j + rem(\mathbf{x}, |\mathbf{UB}_j - \mathbf{LB}_j|)$.

Then, the group-wise boundary violations, i.e., the sum of the boundary violations of all the solutions in each group, are computed as:

$$\mathbf{violation}_{group} = \sum_{solution=1}^{N} \sum_{j=1}^{D} \langle \mathbf{TmC}_{group,solution,j} - \mathbf{UB}_j \rangle + \langle \mathbf{LB}_j - \mathbf{TmC}_{group,solution,j} \rangle \quad (5)$$

where $\mathbf{TmC}_{group,solution,j}$ indicates the j-th element of the focal target solution in the focal group. The $\langle\rangle$ notation indicates that the calculation considers only violations (i.e., either $\mathbf{TmC}_{group,solution,j} > \mathbf{UB}_j$, or $\mathbf{TmC}_{group,solution,j} < \mathbf{LB}_j$).

2.2 Constraint handling

The technique employed to handle the (non-boundary) constraints can also directly affect the search, as it should guide the optimization to the feasible region (**FR**). As briefly discussed in the introduction, various techniques exist to handle the constraints. For instance, an infeasible solution may be simply removed from the population, or it might be repaired to become a feasible one. However, the most common CHT are the penalty-based techniques (Michalewicz and Schoenauer 1996). These simply consist in applying a penalty function to the infeasible solutions, in order to generate a low-quality objective function value:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbf{FR} \\ f(\mathbf{x}) + penalty() & \text{otherwise} \end{cases}$$

The penalty may be static, i.e., set to a fixed value, or dynamic, i.e., it can change during the optimization process. Either way, the idea is that due to the penalty any infeasible solution found during the search *should* be removed from the population during the selection phase. Such simple yet effective technique was adopted also in this work. In particular, here we use an adaptive (i.e., dynamic, with feedback from the search itself) penalty function similar to the one introduced in de Melo and Iacca (2014). According to this scheme, first an infeasibility level proportional to the magnitude of the constraint violations is calculated for each solution, as follows:

$$infeasibility(\mathbf{x}) = \sum g_i(\mathbf{x}) \text{ if } g_i(\mathbf{x}) > \epsilon, i = 1 \ldots m+p \quad (6)$$

It is important to highlight that with this scheme equality constraints are treated as inequality constraints, with a given tolerance factor ϵ (that we set to zero). Thus, $h_i(\mathbf{x}) = 0$ is converted into $|g_i(\mathbf{x})| \leqslant \epsilon$. After this calculation, the penalized objective function of each solution is calculated as the global best function value, plus the (absolute) difference between its function value and the current global best function value, plus $infeasibility()$ divided by the maximum infeasibility value in the current population. This form of adaptation requires then the evaluation of the entire population, in order to obtain the maximum infeasibility value. The pseudo-code of the adaptive penalty function is shown in Algorithm 1.

One important characteristic of this technique is that an infeasible solution may present a better objective function value than a feasible solution, if the former has small constraint violations. However, if the best solution in the population is feasible, then all infeasible solutions must be penalized to have a worse objective function value than that of the best feasible solution. This way, the CHT allows infeasible solutions to get close to an optimum solution located near the border of the feasible region.

2.3 GP update

After each solution computes its target solution, the calculation of the objective function and constraint violations is performed, and the constraints are handled as discussed above, the information on the function value and constraint violations obtained at the target solutions are used in the GP part. More specifically, at each iteration the quality of each group is computed as:

$$\mathbf{TQ}_{group} = \mathbf{Rank}_{group} + \mathbf{violation}_{group} \quad (7)$$

where, in addition to the rank regarding the objective function value (see Table 2 for an example of how $\mathbf{Rank}_{group}$ is calculated), the group violation calculated according to Eq. (5) is considered. This is important because good solutions may be generated by the correction techniques just by chance, while **TmC** had, in fact, large violations. **TQ** values represent, in practice, the GP solutions' quality, i.e., the "fitness" of the GP individuals corresponding to the perturbation logics associated to all groups.

The perturbation logics update process is then triggered when a certain criterion is met: in our experiments, this is done periodically (see the parameter T_{GP} in Table 5). The update replaces the w worst perturbation logics (i.e., the ones displaying the highest **TQ** values) by variations of the w best perturbation logics. In particular, we set $w = 1$, i.e., the group with the worst rank has its perturbation logic updated with a variation of the best group's perturbation logic.

As said earlier, in order to evolve the perturbation logics we employ GP, where variations of the perturbations are obtained by random replacing a sub-tree of the best group's perturbation logic and applying the following rules:

Algorithm 1 Pseudo-code of the adaptive penalty function.

```
▷ The code below starts at row 10 in Algorithm 2
GBOFV = maxPenalty                                   ▷ An arbitrary big constant
maxInfeasibility = 0                                 ▷ Maximum infeasibility in the current population
hasFeasible = False                                  ▷ True if one feasible solution has been found so far, false otherwise

▷ The code below starts at row 26 in Algorithm 2
for group = 1...t do
    for solution = 1...N do
        TmCV_{group,solution} = infeasibility(TmC_{group,solution}), see Eq. (6)
        if TmCV_{group,solution} > ε then            ▷ Infeasible solution
            TmOFV_{group,solution} = TmCV_{group,solution}
            if hasFeasible = False and TmOFV_{group,solution} < GBOFV then
                GBOFV = TmOFV_{group,solution}
                GBCV = TmCV_{group,solution}
                GBC = TmC_{group,solution}
            end if
            if TmCV_{group,solution} > maxInfeasibility then
                maxInfeasibility = TmCV_{group,solution}
            end if
        else                                         ▷ Feasible solution
            TmOFV_{group,solution} = f(TmC_{group,solution})
            if hasFeasible = False then
                hasFeasible = True
            end if
            if TmOFV_{group,solution} < GBOFV or TmCV_{group,solution} < GBCV then
                GBOFV = TmOFV_{group,solution}
                GBCV = TmCV_{group,solution}
                GBC = TmC_{group,solution}
            end if
        end if
    end for
end for

for group = 1...t do
    for solution = 1...N do
        if TmCV_{group,solution} > ε then            ▷ Penalize infeasible solution
            TmOFV_{group,solution} = GBOFV + |GBOFV − TmOFV_{group,solution}| +
                                     TmCV_{group,solution} / maxInfeasibility
        end if
    end for
end for

▷ Continue the content starting at row 27 of Algorithm 2
```

1. The size of the new perturbation, i.e., the number of nodes in the corresponding GP tree, must be in the interval (s_{min}, s_{max}), where s_{min} and s_{max} are user-defined parameters;
2. The new perturbation has to be distinct (in terms of syntactic difference) from the original one;
3. A function is not allowed to receive the same value for its two arguments;
4. The reference perturbations must not be replaced, i.e., they are *fixed perturbations* needed to make sure that at any step of the algorithm there is at least one working perturbation;
5. The perturbation scheme of Eq. (1) must hold.

As a consequence, an improvement is expected when the perturbation logics associated to the worst ranked groups are replaced by a variation of the ones that achieved the best results.

2.4 Stagnation control

Aiming to help solutions escape from local optima, DSO includes different convergence avoidance mechanisms, in particular generation of opposite solutions (see Table 6), stagnation control and restart. These mechanisms are meant to allow the search to move to regions far from the current neighborhood, expanding the overall exploration capability of the algorithm at the cost of possibly slowing down the

Table 2 Example of ranking on a *minimization* problem for two groups with three solutions each

$TmOFV_1$	$TmOFV_2$		$Rank_1$	$Rank_2$
5.0	5.0		1.0	1.0
7.0	3.0		2.0	1.0
10.0	1.0		2.0	1.0
		Rank	1.67	1.0
(a) Values		(b) Ranks		

The (unsorted) objective function values of the solutions in each group are compared, pairwise, and a rank of 1 is given to the lower-fitness solution. In case of tie, the same (lower) rank is assigned

speed of reaching the optimum solution and/or reducing its accuracy (as we will show in Sect. 4).

Stagnation control is implemented as follows. Normally, a hard selection mechanism (see Algorithm 2, row 35) is used to select the best between the current solution and the new one, considering their objective function values. However, a stagnation mechanism detects if the objective function value of the global best solution remains the same for a certain number of iterations ($Iter_{stagnation}$). If this happens, DSO uses soft selection to allow further exploration: **if** ($\mathbf{TmOFV}_{bestGroup,bestSolution} < \mathbf{CBOFV}_{group,solution}$ **or** $U(0,1) < P_{acc}$) **then**
$\mathbf{CBC}_{group,solution} = \mathbf{TmC}_{bestGroup,bestSolution}$
end if
where $(bestGroup, bestSolution)$ are the indexes of the best solution among all groups, $U(0,1)$ is a random number sampled from a uniform distribution in [0, 1], and P_{acc} is the probability of accepting a worse solution. Thus, lower-quality solutions can be inserted in the population, replacing higher-quality solutions. However, elitism is applied in order to keep the global best solution.

Finally, a simple restart mechanism is used to detect if the variable-wise standard deviation of the solutions in **CBC** becomes smaller than a given threshold $threshold_{restart}$. In this case, the entire algorithm and all its data structures – including the GP trees– are reinitialized (keeping however the global best solution) and continued for the remaining number of FES.

3 Experimental setup

In this section, firstly we briefly describe the optimization problems, namely seven engineering design problems and the whole suite of CEC 2017 benchmark problems, used to assess the DSO's performance. The chosen problems pose a broad range of optimization challenges, in terms of number and types of constraints as well as problem dimensionality, thus providing indications on the possible applicability and limitations of the proposed method. Then, we present the details of the configuration of DSO and the computational environment used for the experimentation.

3.1 Engineering design problems

For this part of the experimentation, we selected seven well-known real-world constrained engineering design problems, which have been used as benchmark problems in a number of research papers in the field of engineering optimization. These seven problems have been chosen as they have been used in several recent papers on metaheuristics for constrained optimization and given the availability of benchmark results for comparison purposes. For each of these problems, we also report the best known value found in the literature, that we consider the value-to-reach (VTR).

1. **Design of a three-bar truss**
   This problem consists in minimizing the volume of a three-bar planar truss structure, subject to constraints related to the stress on each truss member. The problem presents two continuous design variables x_1 and x_2, with ranges $0 \leqslant x_1 \leqslant 1$ and $0 \leqslant x_2 \leqslant 1$ (the detailed problem formulation can be found in Kashan (2011); Ray and Liew (2003); Liu et al. (2010); Gandomi et al. (2011)). This problem has a linear objective function and three non-linear inequality constraints. The best known value is currently 263.895843376468 (Liu et al. 2010).
2. **Design of a speed reducer**
   This problem consists in the minimization of the weight of a speed reducer, subject to constraints related to the bending stress of the gear teeth, the surface stress, the transverse defections of the shafts, and the stresses in the shaft. The problem presents seven design variables $(x_1 \ldots x_7)$, with ranges $2.6 \leqslant x_1 \leqslant 3.6$, $0.7 \leqslant x_2 \leqslant 0.8$, $17 \leqslant x_3 \leqslant 28$, $7.3 \leqslant x_4 \leqslant 8.3$, $7.3 \leqslant x_5 \leqslant 8.3$, $2.9 \leqslant x_6 \leqslant 3.9$, and $5 \leqslant x_7 \leqslant 5.5$. All variables are continuous except x_3, that is integer (the detailed problem formulation can be found in (Zhang et al. 2008; Mezura-Montes et al. 2006; Cagnina et al. 2008; Brajevic et al. 2011; Ray and Liew 2003; Liu et al. 2010)). This problem has a non-linear objective function and 11 non-linear inequality constraints. The best known *rounded* value reported in the literature is 2, 994.471066 (Zhang et al. 2008).

Algorithm 2 Pseudo-code of the DSO algorithm.

Input: objective function, problem bounds (**LB**,**UB**), user-defined constants,
GP configuration (see Table 6), number of groups (t), number of solutions per group (N),
maximum number of iterations ($Iter_{max}$), maximum number of iterations without
improvement ($Iter_{stagnation}$), options for the perturbation logics (see Table 6)
Output: Best solution (**GBC**) and best objective function value ($GBOFV$)

Initialize GP with reference perturbations (see Table 6)
Initialize solutions (randomly sampled in [**LB**,**UB**])
Initialize groups (assigning each solution randomly to a group)
Initialize adaptive penalty data (see row 1 in Algorithm 1)

Compute objective function and constraint violations at the initial solutions
Select the N best solutions to be **CBC**

while stopping criteria are not met **do**
  for $group = 1 \ldots t$ **do**
    for $solution = 1 \ldots N$ **do**
      Generate $\mathbf{TC}_{group,solution}$ using the perturbation scheme of the current group, see Eq. (1) and Eq. (2)
      Choose randomly the recombination technique and apply it to combine
      $\mathbf{TC}_{group,solution}$ with $\mathbf{CBC}_{group,solution}$, resulting in $\mathbf{TmC}_{group,solution}$
      Choose randomly the out-of-bounds correction technique and apply it to $\mathbf{TmC}_{group,solution}$,
      saving the violations that occurred
      $\mathbf{TmCV}_{group,solution}$ = constraint violations at $\mathbf{TmC}_{group,solution}$
      $\mathbf{TmOFV}_{group,solution}$ = objective function value at $\mathbf{TmC}_{group,solution}$
    end for
  end for

  Apply adaptive penalty function (see row 6 in Algorithm 1)

  $(bestGroup, bestSolution)$ = indexes of the best solution in **TmOFV**
  Rank groups according to **TmOFV**, see e.g. Table 2
  $improved = false$
  for $group = 1 \ldots t$ **do**
    Calculate $\mathbf{violation}_{group}$, see Eq. (5)
    Update **TQ**, see Eq. (7)
    for $solution = 1 \ldots N$ **do**
      if $\mathbf{TmOFV}_{bestGroup,bestSolution} < \mathbf{CBOFV}_{group,solution}$ **then**
        $\mathbf{CBOFV}_{group,solution} = \mathbf{TmOFV}_{bestGroup,bestSolution}$
        $\mathbf{CBC}_{group,solution} = \mathbf{TmC}_{bestGroup,bestSolution}$
        if $\mathbf{TmOFV}_{bestGroup,bestSolution} < GBOFV$ **then**
          $improved = true$
          $GBOFV = \mathbf{TmOFV}_{bestGroup,bestSolution}$
          $\mathbf{GBC} = \mathbf{TmC}_{bestGroup,bestSolution}$
        end if
      end if
    end for
  end for

  if $improved = false$ **then**
    Apply stagnation control (soft selection) if no. of iterations without improvement $> Iter_{stagnation}$
  end if
  if $Std(\mathbf{CBC}) < threshold_{restart}$ **then**
    Restart algorithm (reinitialize GP, solutions, and groups, and adaptive penalty data)
  end if
  if GP activation condition triggered **then**
    Update the perturbation logics based on **TQ** (replace worst w logics with variations of the w best)
  end if
end while

3. **Design of a pressure vessel**
   This problem consists in minimizing the fabrication cost of a cylindrical pressure vessel with two hemispherical heads, subject to constraints derived from the American Society of Mechanical Engineers (ASME) standards on pressure vessels. The problem presents four contin-

uous variables ($x_1 \ldots x_4$), with ranges $1 \leqslant x_1 \leqslant 99$, $1 \leqslant x_2 \leqslant 99$, $10 \leqslant x_3 \leqslant 200$, and $10 \leqslant x_4 \leqslant 200$ (the detailed problem formulation can be found in Coello Coello (2000); Mezura-Montes et al. (2006); Coelho, L.d.S, (2010); He and Wang (2007); Ray and Liew (2003); Shen et al. (2009)). This problem has a non-linear objective function and four inequality constraints, of which three are linear and one is non-linear. It should be noted that in the original formulation (Coello Coello 2000) x_1 and x_2 are integer variables (multiplied by 0.0625), while x_3 and x_4 are continuous. However, we noticed that there are inconsistencies in the literature: some authors ignore that x_1 and x_2 are integer variables and achieve results close to 5885, 57, such as Dhiman and Kumar (2017). Other authors use instead different bounds for the problem, and achieve even lower results (Maesani et al. 2016). Here, we employ the most used formulation, with the above mentioned bounds and x_1 and x_2 treated as integer variables, for which the best known value is 6, 059.701660 (Mezura-Montes et al. 2006), although it should be noted that this value contrasts with a recent work from Yang at al., who proved analytically that the global optimum is 6, 059.714335048436 (Yang et al. 2013).

4. **Design of a tension/compression spring**
   This problem consists in the minimization of the weight of a tension/compression spring, subject to constraints on the shear stress, the minimum deflection, the surge frequency. The problem presents three continuous design variables (x_1, x_2, x_3) with ranges $0.25 \leqslant x_1 \leqslant 1.3$, $0.05 \leqslant x_2 \leqslant 2.0$, and $2 \leqslant x_3 \leqslant 15$ (the detailed problem formulation can be found in (Coello Coello 2000; Mezura-Montes et al. 2006; Coelho, L.d.S, 2010; He and Wang 2007; Ray and Liew 2003; Shen et al. 2009)). This problem has a non-linear objective function and four inequality constraints, of which one is linear and three are non-linear. The best known *rounded* value is 0.012665 (Coelho, L.d.S, 2010), and the closed-form optimum is 0.0126652327883 (Celik and Kutucu 2018).
5. **Design of a welded beam**
   This problem consists in minimizing the fabrication cost of a welded beam, subject to constraints on the shear stress, the bending stress in the beam, the blocking load on the bar, the end deflection of the beam, and side constraints. The problem has four continuous design variables ($x_1 \ldots x_4$), with ranges $0.1 \leqslant x_1 \leqslant 2.0$, $0.1 \leqslant x_2 \leqslant 10.0$, $0.1 \leqslant x_3 \leqslant 10.0$ and $0.1 \leqslant x_4 \leqslant 2.0$ (the detailed problem formulation can be found in (Coello Coello 2000; Mezura-Montes et al. 2006; He and Wang 2007, ?; Coello Coello and Becerra 2004). This problem has a non-linear objective function and seven inequality constraints, of which two are linear and five are non-linear. The best known *rounded* value is 1.724852 (Mezura-Montes et al. 2006; He and Wang 2007; Coello Coello and Becerra 2004)).
6. **Design of a rolling element bearing**
   This problem consists in *maximizing* the dynamic load carrying capacity of a rolling element bearing, subject to constraints based on kinematic and manufacturing considerations. This problem has a non-linear objective function and 11 inequality constraints, of which seven are linear and four are non-linear. The problem presents 10 decision variables ($x_1 \ldots x_{10}$), which are all continuous except x_3 (the detailed problem formulation can be found in Eskandar et al. (2012)). However, similarly to what we observed on the pressure vessel problem, we noted some inconsistencies in the literature: in particular, the most recent works addressing this problem report results obtained assuming x_3 as a continuous variable (Eskandar et al. 2012; Dhiman and Kumar 2017). The best known value is 85, 538.48 (Eskandar et al. 2012), although under the assumption of continuity of x_3. Under the same assumption, we found that DSO is able to find a feasible solution with a fitness value of 85, 539.07823091, that is thus better than the best solution previously found in the literature. Considering x_3 as an integer variable, on the other hand, leads to a fitness value of 85, 533.18278573.
7. **Design of a multiple disk clutch brake**
   This problem consists in minimizing the mass of a multiple disk clutch brake, subject to mechanical constraints. It is important to state that, contrarily to all the previous problems, for this problem the constraint values must be *positive*. The problem presents five integer design variables, namely inner radius ($r_i = 60, 61, 62, \ldots 80$), outer radius ($r_o = 90, 91, 92, \ldots 110$), disk thickness ($t = 1, 1.5, 2, 2.5, 3$), actuating force ($F = 600, 610, 620, \ldots 1000$), and number of friction surfaces ($Z = 2, 3, \ldots, 9$) (the detailed problem formulation can be found in Balande and Shrimankar (2017); Eskandar et al. (2012)). This problem has a non-linear objective function and a total of eight inequality constraints, of which one is linear and seven are non-linear. The best known value is 0.313656 (Balande and Shrimankar 2017).

For the reader's convenience, a summary of the main details of the seven engineering design problems is reported in Table 3.

3.2 CEC 2017 benchmark

In addition to the engineering design problems, we considered the 28 scalable problems included in the CEC 2017 benchmark on constrained optimization (Wu et al. 2017), in four dimensionalities: 10D, 30D, 50D and 100D. In the benchmark, all variables are assumed continuous. A sum-

Table 3 Details of the engineering design problems

Problem	Variables	Objective	Constraints	
			E	I
1. Three-bar truss	2C	L	0	3L
2. Speed reducer	6C 1I	N	0	11N
3. Pressure vessel	2C 2I	N	0	3L 1N
4. Tension/compression spring	3C	N	0	1L 3N
5. Welded beam	4C	N	0	2L 5N
6. Rolling element bearing	9C 1I	N	0	7L 4N
7. Multiple disk clutch brake	5I	N	0	1L 7N

I is the number of inequality constraints, E is the number of equality constraints. The symbols "L" and "N", indicate respectively linear and non-linear functions, while "C" and "I" indicate respectively continuous and integer variables

mary of the main features of the benchmark functions is reported in Table 4. Please note that all problems in this benchmark are formulated as *minimization* problems. In the table, it can be seen that the benchmark functions differ mainly as for what concerns the kind of objective function,[2] and the number and kind of equality and inequality constraints. We refer the interested reader to Wu et al. (2017) for the complete mathematical details of the benchmark.

3.3 Configuration of the algorithm

The list of DSO's parameters used in the experiments is shown in Table 5. We should remark that no specific parameter tuning was performed: instead, we set the parameter values empirically based on previous observations reported in de Melo and Banzhaf (2017). In all the algorithm runs, all groups start with one reference perturbation, that is the "rand/1" mutation scheme used in DE (Das et al. 2016). For the engineering design problems, the maximum number of FES (FES_{max}) was chosen to be compatible with the literature and present competitive performance, see the next section for details. For the CEC 2017 benchmark, it was set as indicated in Wu et al. (2017), i.e., $20,000 \times D$, where D is the problem dimensionality. In all the experiments, we increased the function evaluation counter by one whenever a newly generated solution was evaluated either in terms of objective function or constraints.[3] For each engineering design problem, we performed 50 independent runs of DSO. For the CEC 2017 problems, we performed 25 independent runs, as indicated in Wu et al. (2017). For reproducibility reasons, we started the random seed of each run with its own progressive id; thus, run number 1 used seed 1, run number 2 used seed 2, and so on.

As for the GP part of the algorithm, its main settings (i.e, the terminal and non-terminal nodes from which the perturbation logics can be composed, see Eq. (1), as well as the "rand/1" reference perturbation) are shown in Table 6. With reference to the table, the non-terminal nodes include both single and double argument functions, namely:

- $square()$: returns the element-wise squared **arg**;
- $plus()$, $minus()$, $times()$, $avg()$: return the element-wise sum, difference, multiplication and average of $\mathbf{arg}_1$ and $\mathbf{arg}_2$, respectively.

The terminal nodes include various constants, (matrices of) random numbers, and a diverse set of calculated weights, base vectors and differential vectors, namely:

- $U(min, max, N, D)$: returns a matrix of numbers sampled from a uniform distribution in $[min, max]$ (size: $N \times D$);
- $G(mean, variance, N, D)$: returns a matrix of numbers sampled from a Gaussian distribution with given $mean$ and $variance$ (size: $N \times D$);
- $Sigma(\cdot)$: returns $\sigma \times G(0, 1, N, D) \times \mathbf{BoundsDiff}^{\mathrm{T}} \times U(0, 0.5)$ (saturated within the problem bounds), as used in de Melo and Iacca (2014), where $\sigma = 0.04 \times \mu_{eff} \times \|\boldsymbol{\mu}\|$ (with $\boldsymbol{\mu}$ being the mean calculated on the best 50% solutions in **CBC** and μ_{eff} being its variance-effective size), as in de Melo and Banzhaf (2017) (size: $N \times 1$);
- $RandDiag(\cdot)$: returns a matrix containing N rows randomly selected (with possible duplicates) from a $D \times D$ identity matrix (size: $N \times D$);
- $Std(\cdot)$: returns a tensor containing the variable-wise standard deviation calculated within each group in **CBC** (size: $t \times N \times D$);
- $MVNS(\cdot)$: returns new random solutions sampled from a multivariate Gaussian distribution with mean and covari-

[2] In this regard, it should be noted that the benchmark characterizes the objective and constraint functions as being separable, non-separable, or rotated, i.e., obtained via the application of a rotation transformation to another given function, rather than being linear or non-linear.

[3] One possible optimization could be to pre-calculate the constraint violations before evaluating the objective function, thus excluding the evaluation of infeasible solutions from the FES.

Table 4 Details of the CEC 2017 benchmark functions

Problem	Objective	Constraints	
		E	I
C01	N	0	1S
C02	N,R	0	1N,R
C03	N	1S	1S
C04	S	0	2S
C05	N	0	2N,R
C06	S	6S	0
C07	S	2S	0
C08	S	2N	0
C09	S	1N	1N
C10	S	2N	0
C11	S	1N	1N
C12	S	0	2S
C13	N	0	3 S
C14	N	1S	1S
C15	S	1N	1S
C16	S	1N	1S
C17	N	1N	1S
C18	S	1N	2N
C19	S	0	2N
C20	N	0	2S
C21	R	0	2R
C22	R	1R	1R
C23	R	1R	1R
C24	R	1R	1R
C25	R	1R	1R
C26	R	1R	1R
C27	R	1R	2R
C28	R	0	2R

I is the number of inequality constraints, E is the number of equality constraints. The symbols "S", "N", and "R" indicate respectively separable, non-separable and rotated functions. All variables are assumed continuous. All problems are scaled for $D = 10, 30, 50,$ and 100 variables

Table 5 Configuration of DSO used in the numerical experiments

Parameter	Value
Constants	$C_1 = 0.5, C_2 = 0.3, C_3 = 0.7$
Groups (t)	D (problem dimensionality)
Solutions per group (N)	10
Perturbation logics to replace (w)	1
Update GP period (T_{GP})	2 iterations
Prob. of accepting a worse solution (P_{acc})	0.9
Stagnation period ($Iter_{stagnation}$)	10% $Iter_{max}$
Convergence threshold to restart ($threshold_{restart}$)	1e-6
Crossover rate (CR)	$U(0.1, 0.6)$
Maximum number of FES (FES_{max})	Depending on the problem
Maximum number of iterations ($Iter_{max}$)	$FES_{max}/(t \times N)$

ance matrix calculated on the best 50% solutions in either **CBC** or **TmC** (size: $N \times D$);
- $Opposition(\cdot)$: returns a tensor containing the opposite solutions w.r.t. to the problem bounds, i.e., **BoundsSum**$-$**CBC** or **BoundsSum** $-$ **TmC** (size: $t \times N \times D$);
- $Mean(\cdot)$: returns a tensor containing the variable-wise means calculated within each group in **CBC** (size: $t \times N \times D$);
- $Median(\cdot)$: returns a tensor containing the variable-wise medians calculated within each group in **CBC** (size: $t \times N \times D$);
- $Shift(\cdot)$: returns a tensor containing the variable-wise differences between **TmC** and **CBC** (size: $t \times N \times D$).

3.4 Computational environment

The code[4] was developed and tested in MATLAB R2015a (8.5.0.197613) 64-bit. The experiments were performed on an Intel i9-7940x workstation running Ubuntu Linux 19.10.

4 Results and discussion

First, we provide an overall analysis of DSO's performance alone, i.e., without comparing it to other algorithms. Then, for each problem, we compare our results with those from the literature.

4.1 Engineering design problems

As a preliminary experiment, we tried to address the question as to whether evolving the perturbations via GP is actually beneficial for solving constrained optimization problems. In fact, the entire rationale of the proposed DSO is that, since there are several possible ways to conduct the search (which means, in other words, to apply perturbations to existing solutions in order to generate new ones), but it is hard to state *a priori* which algorithm may work better on a certain problem (or set of problems), using a self-constructing algorithm can be an advantage. In our proposal, the self-constructing capability is implemented by GP. To verify if the use of GP provides a benefit, we considered the seven engineering design problems described above and executed for each of them 50 runs of the full version of DSO (i.e., including GP) against a version of DSO where the GP part is disabled and a different perturbation is randomized for each group of solutions at the beginning of the algorithm. We believe that this simple comparison can shed some light on our intuition regarding the advantage of using GP.

As we expected, our results show that the version of the algorithm using GP to optimize the perturbation statistically outperforms, in terms of fitness function, the version with randomized perturbation. Table 7 shows the median of the best fitness values (including the adaptive penalty, i.e., some of the corresponding solutions may be infeasible) found across 50 runs of DSO vs DSO with randomized perturbation. The Wilcoxon rank-sum test ($\alpha = 0.05$) rejects the null hypothesis of statistical equivalence of the two algorithms for all the seven problems. The symbol "+" in the table indicates a statistical superiority of the median fitness value obtained by DSO w.r.t. the one obtained by DSO with randomized perturbation. This superiority can be seen also in Fig. 1, which shows the distribution of the best fitness values found in each of the 50 independent runs for the two DSO with and without GP. For all the problems, it can be immediately noticed that the distributions obtained by DSO with randomized perturbation are much larger, and skewed towards worse values. Thus, we can conclude that using GP to automatically evolve the perturbations can be beneficial.

The next step of our analysis is to analyze in depth the results obtained by the full version (i.e., with GP) of DSO. In the following, we will consider only DSO with GP and omit to specify this. Table 8 shows the descriptive statistics of DSO results on the seven engineering design problems, and the VTR reported in the previous section. Of note, the best results found by DSO are very close to the VTR for all problems, being even superior to the VTR on problem 6 (considering x_3 as a continuous variable). Furthermore, the Median and Mean values are close to the best value (Min or Max, depending on the problem) value, which indicates a high robustness of the algorithm. In order to quantify this aspect, we calculated std/Mean to obtain a rate on the same scale for all problems. It can be noted that the largest std/Mean occurred on problem 5, followed by problem 3. On the other hand, the lowest std/Mean values were obtained on problems 1 and 2, where DSO was very effective in all runs, except a few runs in which the algorithm got trapped in local optima. This can be seen again in Fig. 1, where it can be observed that DSO presented robust performances also on problems 4 to 6. The lowest robustness (i.e., highest variation across 50 runs) was shown on problem 3 and 7: this might be due to the presence of multiple integer variables (two in problem 3 and five in problem 7), which makes the problem harder. To increase the algorithm robustness on these problems, one possible solution is to include additional elements for creating perturbation logics more specific for this case (see Table 6). For the sake of completeness, we report the best results obtained by DSO in Table 9. It can be observed that all the best solutions are feasible ($g(\mathbf{x}) \leq 0$ for problems 1 to 6 and $g(\mathbf{x}) \geq 0$ for problem 7).[5]

[4] The source code is available upon request.

[5] Note that the solution reported for problem 6 assumes x_3 as a continuous variable. With x_3 considered as an integer variable, the best solution found by DSO is: $\mathbf{x} = \{125.72271843, 21.42330097, 11, 0.515,$

Table 6 Information made available to the GP part of the algorithm to generate perturbation logics

Setting	Value
Single argument functions	$square(\mathbf{arg})$
Double argument functions	$plus(\mathbf{arg}_1, \mathbf{arg}_2), minus(\mathbf{arg}_1, \mathbf{arg}_2),$ $times(\mathbf{arg}_1, \mathbf{arg}_2), avg(\mathbf{arg}_1, \mathbf{arg}_2)$
Constants	C_1, C_2, C_3, **BoundsDiff**, **BoundsSum** $(1.0/Iter_{curr}), (1.0/FES_{curr})$
Random numbers	$U(0, 1, N, D), U(0.5, 1, N, D),$ $G(0, 1, N, D), G(0, 0.1, N, D),$ $G(0, Iter_{curr}, N, D), G(0, FES_{curr}, N, D)$
Calculated weights	$Sigma(\mathbf{CBC}, \mathbf{CBOFV}, N, \mathbf{BoundsDiff}),$ $RandDiag(N, D),$ $Std(\mathbf{CBC})$
Base vectors	$MVNS(\mathbf{CBC}) + Sigma(\mathbf{CBC}, \mathbf{CBOFV}, N, \mathbf{BoundsDiff}),$ $Opposition(\mathbf{CBC}, \mathbf{BoundsSum}),$ $Mean(\mathbf{CBC}), Median(\mathbf{CBC}), \mathbf{GBC}, \mathbf{CBC}$
Differential vectors	$MVNS(\mathbf{CBC}) + Sigma(\mathbf{CBC}, \mathbf{CBOFV}, N, \mathbf{BoundsDiff}),$ $MVNS(\mathbf{TmC}) + Sigma(\mathbf{CBC}, \mathbf{CBOFV}, N, \mathbf{BoundsDiff}),$ $Shift(\mathbf{TmC}, \mathbf{CBC}), \mathbf{TmC}_{r1}, \mathbf{TmC}_{r2}, \mathbf{TmC}_{r3},$ $\mathbf{TmC}_{r4}, \mathbf{TmC}_{r5}, \mathbf{CBC}_{r1}, \mathbf{CBC}_{r2},$ $\mathbf{CBC}_{r3},$ $\mathbf{CBC}_{r4}, \mathbf{CBC}_{r5}, \mathbf{GBC}, Opposition(\mathbf{CBC}, \mathbf{BoundsSum}),$ $Opposition(\mathbf{TmC}, \mathbf{BoundsSum})$
Reference perturbation	$\mathbf{CBC}_{r1} + C_1 \times (\mathbf{CBC}_{r2} - \mathbf{CBC}_{r3})$

$r1$, $r2$, etc. are indices of solutions chosen randomly. For more details, see the symbols in Table 1 and the explanation in the text

Table 7 Engineering design problems: median of the best fitness values found across 50 runs of DSO vs DSO with randomized perturbation (i.e., with the GP part disabled)

DSO	DSO (random)	W
263.8958	264.7022	+
2994.4713	3050.9444	+
6635.7096	10400.7578	+
0.0127	0.022021	+
1.8673	3.1048	+
84,831.3275	56649.7132	+
0.3263	0.34091	+

Problem 6 is a maximization problem (the values reported in the table are the actual function values)

Table 8 Engineering design problems: descriptive statistics of the DSO results

Prob.	Min	Median	Mean	Max	std	std/Mean	VTR
1	263.89584338	263.89584630	263.89705299	263.93992706	6.30e-03	2.387e-05	263.895843376468
2	2994.47106770	2994.47132983	2994.69165169	3004.42909435	1.406	4.697e-04	2994.471066
3	6059.71433565	6635.70963326	6713.56036966	7544.49251821	4.777e+02	7.116e-02	6059.701660
4	0.01266539	0.01274725	0.01288959	0.01407721	3.245e-04	2.517e-02	0.012665
5	1.72485332	1.86735435	2.00059370	3.18904036	3.401e-01	1.700e-01	1.724852
6	74,345.08450340	84,831.32750597	83,173.11537690	85,539.07823091	3.154e+03	3.792e-02	85,538.48
7	0.31365661	0.32634990	0.32881603	0.38815006	1.450e-02	4.409e-02	0.313656

Problem 6 is a maximization problem (the values reported in the table are the actual function values)

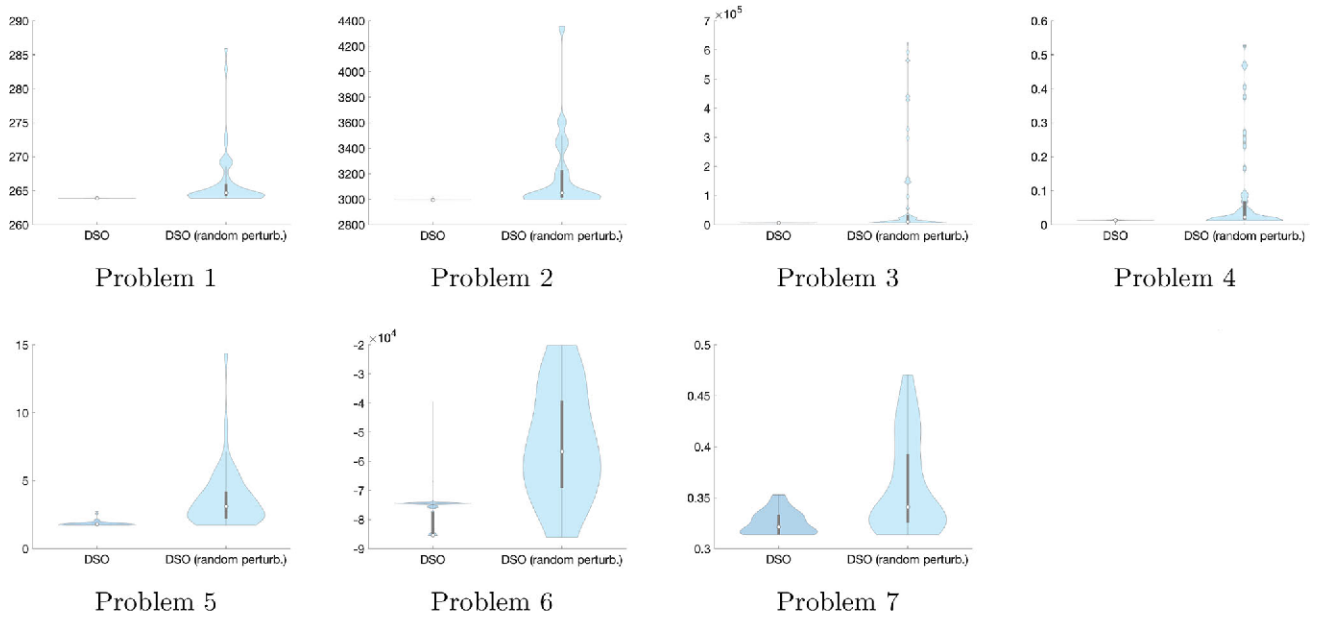

Fig. 1 Engineering design problems: violin plots of the best fitness values found across 50 runs of DSO vs DSO with randomized perturbation (i.e., with the GP part disabled). Note: problem 6 is a maximization problem (the values shown in the plot are negated)

The performance of DSO in terms of best objective function value at each step of the algorithm (including the adaptive penalty, i.e., at each step the best function value may correspond to an infeasible solution) in each of the 50 runs on each problem is shown in Fig. 2. In Fig. 2, DSO plateaus quite fast on problems 1 to 4. However, it can also be noticed that DSO keeps refining the solutions throughout the entire computational budget, and that on some problems it suddenly "jumps" to better solutions. This behavior has two explanations: on the one hand, as noted earlier problem 7 is fully integer (with 5 integer variables), while problem 3 is mixed-integer (with 2 integer variables and 2 continuous ones). The presence of multiple integer variables (all other problems have at most one integer variable) thus justifies these discontinuities in the fitness trends. On the other hand, this behavior also depends on how the solutions are located in the search space, and what are the current perturbation logics evolved by DSO. In fact, it can happen that a certain perturbation logic becomes effective only in some specific search conditions.

Another thing that can be noted in Fig. 2 is that DSO shows a quite visible variation across the 50 runs on problems 5 to 7. This indicates that either those fitness landscapes contain multiple optima, or that the feasible regions are hard to find, thus requiring a continuous refinement of the search. Finally, one notable aspect of the figure is that on problems 2 and 5, $f(\mathbf{x})$ increases in the first iterations and then starts to decrease, until it plateaus. Such behavior is caused by the adaptive penalty function. At the beginning of the search, usually all solutions are infeasible, thus the one presenting the lowest constraint violation is chosen as the best solution and all the other solutions are penalized according to the adaptive penalty function. Then, when a feasible solution is found, it becomes the best solution and gets the best $f(\mathbf{x})$. Therefore, all the infeasible solutions are penalized according to this best $f(\mathbf{x})$ value, and their $f(\mathbf{x})$ is increased to be worse than that. On the other hand, any other feasible solution present in the population is not penalized, as explained earlier. Still, some infeasible solutions may have better $f(\mathbf{x})$ than feasible solutions, which allows them to get close to an optimum solution near the border of the feasible region.

Next, we compare the results obtained by DSO on each problem with those from the literature. The list of the compared algorithms is presented in Table 10. To make a fair comparison, for each algorithm we considered the original results reported in the corresponding paper. However, we included in the comparisons only those algorithms for which the paper reports at least the descriptive statistics (in terms of Best, Mean, Worst and std) and the FES used in the original experiments. It should also be noted that the results reported in the literature often have different precision (while our results include 8 decimal digits), thus we report them here as they appear in the original papers, with zero-padding on Best, Mean, and Worst for a better visualization.

0.515, 0.48494889, 0.68389258, 0.3, 0.02711788, 0.63056305}, with $f(\mathbf{x}) = 85,533.18278573$.

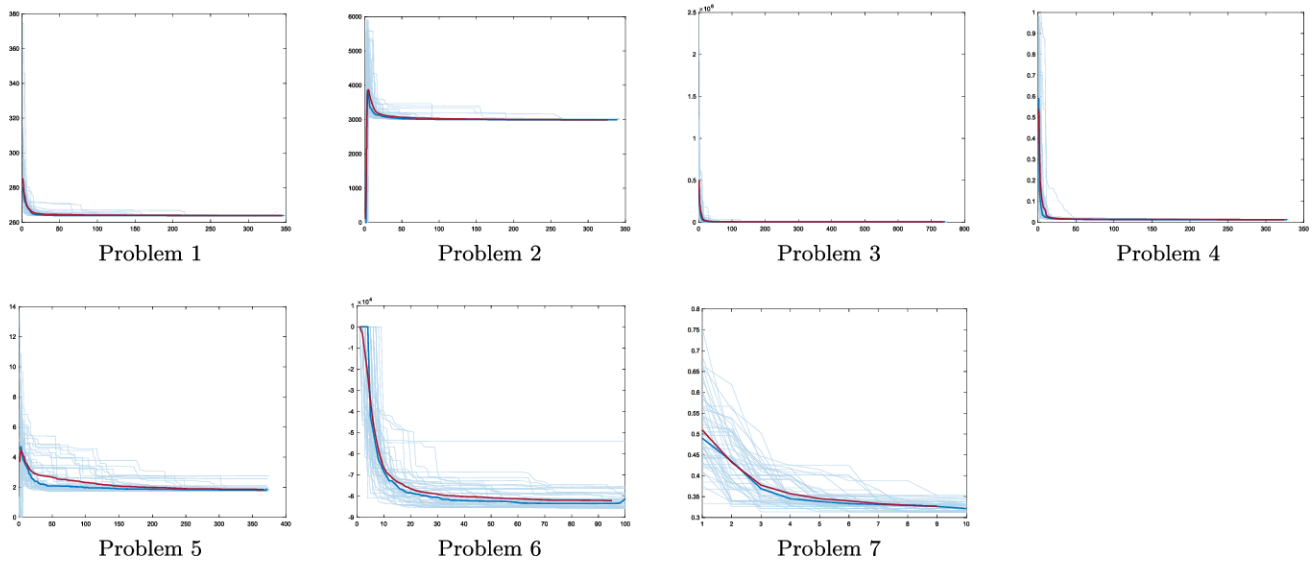

Fig. 2 Engineering design problems: fitness trends of DSO. The x-axis and y-axis show, respectively, the number of function evaluations and the fitness value (including the adaptive penalty). Every plot shows the trends of 50 runs, their mean (thick red line) and median (thick blue line). Note: problem 6 is a maximization problem (the values shown in the plot are negated)

Table 9 Engineering design problems: best feasible solutions ($g(\mathbf{x}) \leq 0$ for problems 1 to 6 and $g(\mathbf{x}) \geq 0$ for problem 7) found by DSO

	Problem						
	1	2	3	4	5	6	7
$f(\mathbf{x})$	263.89584338	2994.47106770	6059.71433565	0.01266539	1.72485332	85,539.07823091	0.31365661
$\mathbf{x}$	0.78867591	3.5	0.8125	0.05162531	0.20572949	125.72268595	70
	0.4082461	0.7	0.4375	0.35518461	3.47049516	21.42328816	90
		17	42.09844559	11.37947206	9.03662185	11.00115487	1
		7.3	176.63659588		0.20572974	0.515	780
		7.71531991				0.51500014	3
		3.35021467				0.46931813	
		5.28665447				0.66235054	
						0.30000015	
						0.03817113	
						0.6031354	
$g(\mathbf{x})$	0	− 0.07391528	0	− 5.4e-07	− 0.00767042	− 6.78e-06	0
	− 1.46410411	− 0.19799853	− 0.03588083	− 2.98e-06	− 0.00024919	− 9.99430708	24
	− 0.53589589	− 0.49917225	− 8.858e-05	− 4.05072819	− 2.5e-07	− 3.51796116	0.92241197
		− 0.9046439	− 63.36340412	− 0.72879339	− 3.43298284	− 3.32922616	9.83665365
		0			− 0.08072949	− 0.72268595	7.89469659
		0			− 0.23554032	− 8.82009599	0.16908322
		− 0.7025			− 0.00747431	− 2.337e-05	34.0875
		0				0	14.83091678
		− 0.79583333				− 1.4e-07	
		− 0.05132575				− 6.6e-06	
		0				− 3.6e-06	

Problem 6 is a maximization problem (the fitness reported in the table is the actual function value)

4.1.1 Problem 1—three-bar truss

Table 11 shows the results of DSO and the compared metaheuristics from the literature. We configured DSO to run for the same maximum number of FES as MVDE (De Melo and Carosio 2013). DSO found a near-optimum solution, outperforming several state-of-the-art algorithm while using only 7, 000 FES, compared to more than 10, 000 required by the others. The Mean and Worst results are also competitive, indicating that DSO is very effective at solving this problem, reaching high-quality solutions in a low computing time.

4.1.2 Problem 2—speed reducer

Table 12 reports the comparison with the state-of-the-art algorithms. Again, DSO had the same maximum number of FES as MVDE (De Melo and Carosio 2013). With 24, 000 FES, the proposed algorithm achieved a Mean value of 2, 994.69165169, outperforming most algorithms included in the comparison. Even though DSO got trapped in suboptimal solutions in a few runs, the compared algorithms show in general much larger Mean and Worst values, indicating that DSO was more effective than the other algorithms at escaping local optima.

4.1.3 Problem 3—pressure vessel

As seen earlier, this problem presents two integer variables. However, several authors treat them as continuous, which makes the problem relatively easier. We report here also these results, for the reader's reference. In Table 13, it can be observed that none of the algorithms (including DSO) achieved the VTR reported in Mezura-Montes et al. (2006). This was a hard problem for DSO: although using 30, 000 FES, DSO failed to find high-quality solutions in most of the runs, resulting in a low average performance. DSO would probably need a specific configuration for this problem w.r.t. the number of groups and the number of solutions per group in order to improve the optimization performance.

4.1.4 Problem 4—tension/compression spring

For this problem, DSO was configured to perform a maximum of 10, 000 FES as MVDE (De Melo and Carosio 2013). In Table 14, it can be seen that the VTR (up to 6 decimal places, as reported in the literature) was reached at least once, while many algorithms failed. DSO showed better Mean and Worst values than several algorithms, such as SC, PSO, MFO, MVO and GSA. Furthermore, we noted that the Mean value of DSO is biased because of a few outliers, which may have happened because DSO used less than half FES compared to most of the other algorithms. When one takes FES into consideration, the results of DSO get even more competitive.

4.1.5 Problem 5—Welded beam

As it can be seen in the statistics reported in Table 15, DSO achieved the VTR using only 15, 000 FES (set as for MVDE (De Melo and Carosio 2013)), outperforming most of the compared algorithms. However, it was unable to obtain an adequate average performance. Nonetheless, DSO outperformed other algorithms such as PSO, GSA and GA. In the more detailed descriptive statistics shown in Table 8, it can be noticed that the Median value was considerably lower than the Mean value because three runs did not find good solutions.

4.1.6 Problem 6—rolling element bearing

As discussed earlier, the results available in the literature show a rather serious inconsistency in how x_3 is treated, with the most recent works that seem to relax it as a continuous variable, rather than integer. Here, for completeness we report the results of DSO obtained in both cases, i.e., assuming x_3 either as an integer or a continuous variable. For this problem, there is an important finding: considering x_3 continuous, DSO found a better solution than the one reported in the literature as VTR (see Table 16). In fact, DSO outperformed in terms of Best function value all the other 11 algorithms (which also make an assumption of continuity on x_3), even though this new VTR was found only once. Furthermore, it should be noticed that DSO was tested with a much lower number of FES than the majority of algorithms.

Considering instead x_3 as an integer variable, DSO was still able to find a feasible solution, whose objective value is actually very close to the VTR reported in Eskandar et al. (2012). However, clearly these two values are not directly comparable due to the different handling of x_3, which makes the problem different.

4.1.7 Problem 7—multiple disk clutch brake

Even though this problem is well-known in the Evolutionary Computation research community, it has been tested less often than the other engineering design problems. Thus, there are only a few results in the literature. Here, we set the maximum number of FES as the same value used by WCA (Eskandar et al. 2012). Of note, this is a very small value, since with the tested settings DSO has only 10 iterations to run, during which also the perturbation strategies have to evolve. Nevertheless, as presented in Table 17, DSO succeeded in reaching the VTR several times and showed the lowest Mean value. DSO resulted also quite robust, although its std was several orders of magnitude higher than that of WCA, due to a single outlier run corresponding to the Worst value. To summarize, this problem is useful to show

Table 10 Engineering design problems: algorithms included in the comparisons (in alphabetic order) and references from which the corresponding results were taken

Short name	Long name
AATM	Accelerating adaptive trade-off model (Wang et al. 2009)
CPSO	Co-Evolutionary Particle Swarm Optimization (He and Wang 2007)
CS	Cuckoo Search Algorithm (Gandomi et al. 2011)
DE	Differential Evolution (Mezura-Montes et al. 2006)
DEDS	Differential Evolution with Dynamic Stochastic Selection (Zhang et al. 2008)
GA	Genetic Algorithm (Dhiman and Kumar 2017)
GSA	Gravitational Search Algorithm (Dhiman and Kumar 2017)
GWO	Grey Wolf Optimizer (Dhiman and Kumar 2017)
HEAA	Hybrid Evolutionary Algorithm and Adaptive Constraint-Handling Technique (Wang et al. 2009)
HPSO	Hybrid Particle Swarm Optimization Algorithm (He and Wang 2007)
HS	Harmony Search (Dhiman and Kumar 2017)
I-GSO	Improved Group Search Optimizer (Shen et al. 2009)
LCA	League Championship Algorithm (Kashan 2011)
MAL-FA	Modified Augmented Lagrangian Firefly Algorithm (Balande and Shrimankar 2017)
MFO	Moth-Flame Optimization (Dhiman and Kumar 2017)
MVDE	Multi-View Differential Evolution (De Melo and Carosio 2013)
MVO	Multi-Verse Optimizer (Dhiman and Kumar 2017)
PSO	Particle Swarm Optimization (Dhiman and Kumar 2017)
QPSO	Gaussian Quantum-Behaved Particle Swarm Optimization (Coelho, L.d.S, 2010)
SC	Society and Civilization (Ray and Liew 2003)
SCA	Sine Cosine Algorithm (Dhiman and Kumar 2017)
SHO	Spotted Hyena Optimizer (Dhiman and Kumar 2017)
WCA	Water Cycle Algorithm (Eskandar et al. 2012)

Note that not all algorithms are compared on all problems

that DSO may find good quality solutions even with a small number of FES.

4.2 CEC 2017 benchmark

The detailed results of the DSO algorithm on the CEC 2017 benchmark are reported in Tables 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37. The tables are formatted as indicated in Wu et al. (2017), to facilitate a comparison with the other algorithms tested on the same benchmark. More specifically, for each problem (C1, C2, ..., C28) we collected the best feasible solution (i.e., the one with the minimum function value), or, in case of missing feasible solutions, the best infeasible solution (i.e., the one with the minimum mean constraint violation) found at the end of each of the 25 runs of DSO (i.e., within a budget of $20,000 \times D$ FES). Then, we sorted these 25 solutions by ranking the feasible solutions (sorted according to their function values) before the infeasible ones (sorted according to the mean value of their constraint violations). In the tables, we report for each problem the function value of the Best, Worst and Median solution according to this sorting. We also report the average function value (Mean) across the 25 solutions (please note that this value in general does not correspond to an actual solution), as well as the corresponding standard deviation (std). As for the other metrics reported in the tables, $\overline{v}$ indicates the mean constraint violation at the median solution, c indicates the number of violated constraints at the median solution (divided based on their values, i.e., respectively bigger than 1.0, in the range [0.01, 1.0], and in the range [0.0001, 0.01]), $\overline{vio}$ indicates the mean constraint violation of the 25 solutions, and finally SR is the feasibility (or "success") rate, i.e., the % of runs in which at least one feasible solution was found within the allotted budget.

For the sake of brevity, we do not report here the complete results obtained by the algorithms that participated in the CEC 2017 competition, which are available online.[6] Instead, we highlight the main strengths and weaknesses of the proposed algorithm in comparison with the algorithms from the competition,[7] as well as another recent algorithm named HECO-DE (Xu et al. 2020), which currently represents the state-of-the-art on the CEC 2017 benchmark.

[6] A summary of the results is available at: https://github.com/P-N-Suganthan/CEC2017.

[7] For further analysis, we make our raw data available at: https://bit.ly/2XBHqCm.

Table 11 Engineering design problems: comparison of results for Problem 1—Three-bar truss (VTR: 263.895843376468 (Liu et al. 2010))

Algorithm	Best	Mean	Worst	std	FES
DSO	263.89584338	263.89705299	263.93992706	6.30e-03	7,000
MVDE (De Melo and Carosio 2013)	263.89584337	263.89584338	263.89585480	2.58e-07	7,000
MAL-FA (Balande and Shrimankar 2017)	263.89584300	263.89610100	263.89584700	9.70e-07	4,000
WCA (Eskandar et al. 2012)	263.89584300	263.89590300	263.89620100	8.71e-05	5,250
LCA (Kashan 2011)	263.89584340	263.89584340	263.89584340	5.68e-14	10,000
DEDS (Zhang et al. 2008)	263.89584300	263.89584900	263.89584300	9.70e-07	15,000
HEAA (Wang et al. 2009)	263.89584300	263.89609900	263.89586500	4.90e-05	15,000
AATM (Wang et al. 2009)	263.89584350	263.89660000	263.90041000	1.10e-03	17,000
SC (Ray and Liew 2003)	263.89584660	263.90330000	263.96975000	1.26e-02	17,610
PSO-DE (Liu et al. 2010)	263.89584338	263.89584338	263.89584338	4.50e-10	17,600

Table 12 Engineering design problems: comparison of results for Problem 2—Speed reducer (VTR: 2994.471066 (Zhang et al. 2008))

Algorithm	Best	Mean	Worst	std	FES
DSO	2994.47106770	2994.69165169	3004.42909435	1.406	24,000
MVDE (De Melo and Carosio 2013)	2994.47106600	2994.47106600	2994.47106900	2.819316e-7	24,000
WCA (Eskandar et al. 2012)	2994.47106600	2994.47439200	2994.50557800	7.4e-03	15,150
DE (Mezura-Montes et al. 2006)	2996.35668900	2,996.36722000	2,996.39013700	8.2e-03	24,000
SHO (Dhiman and Kumar 2017)	2998.55070000	2999.64000000	3003.88900000	1.93193	30,000
GWO (Dhiman and Kumar 2017)	3001.28800000	3005.84500000	3008.75200000	5.83794	30,000
GSA (Dhiman and Kumar 2017)	3051.12000000	3170.33400000	3363.87300000	92.5726	30,000
HS (Dhiman and Kumar 2017)	3029.00200000	3295.32900000	3619.46500000	57.0235	30,000
MFO (Dhiman and Kumar 2017)	3009.57100000	3021.25600000	3054.52400000	11.0235	30,000
PSO (Dhiman and Kumar 2017)	3067.56100000	3186.52300000	3313.19900000	17.1186	30,000
AATM (Wang et al. 2009)	2994.51677800	2994.58541700	2994.65979700	3.3e-02	40,000
PSO-DE (Liu et al. 2010)	2996.34816700	2996.34817400	2996.34820400	6.4e-06	54,350
SC (Ray and Liew 2003)	2994.74424100	3001.75826400	3009.96473600	4.0	54,456

First of all, it can be seen that DSO performs very well in terms of SR at 10D, but its performance degrades when the dimensionality increases. This seems to indicate that as the dimensionality grows more complex perturbations are needed, thus requiring more iterations for the GP to evolve them. Using a "avg/oneFeas/allFeas" notation, where "avg" indicates the average SR across all problems, "oneFeas" indicates the no. of problems in which the algorithm found a feasible solution in *at least one run*, and "allFeas" the no. of problems in which the algorithm found a feasible solution in *all* runs, we observe the following results in comparison with the winner of the CEC 2017 competition, L-SHADE44 (Polakova 2017):

10D: DSO 83.71%/24/20, L-SHADE44 72.43%/21/18;
30D: DSO 73.14%/22/19, L-SHADE44 74.28%/21/19;
50D: DSO 67.43%/21/17, L-SHADE44 74.86%/21/20;
100D: DSO 62.43%/19/16, L-SHADE44 71.14%/21/19.

Of note, there are two cases in which, compared to the algorithms from the competition (L-SHADE44 (Polakova 2017), UDE (Trivedi et al. 2017), L-SHADE-44-IDE (Tvrdík and Poláková 2017), CAL-SHADE (Zamuda 2017)), DSO is the only algorithm capable of finding feasible solutions: C18 and C27. These two functions have both one equality constraint and two inequality constraints (all non-separable or rotated). In 10D, DSO is able to solve (i.e., find at least one feasible solution) both functions with a high SR (88% and 68% respectively), whereas all the CEC 2017 competition algorithms fail. In 30D, DSO solves C18 and C27 with SR 56% and 16% (zero for the other competition algorithms). In 50D, DSO solves C18 with SR 4% (zero for the other algorithms) but not C27. In 100D, DSO solves C18 with SR 16% (zero for the other algorithms), but again not C27. This might indicate that the perturbation logics evolved in the GP part of DSO deal favorably with these two functions. It is quite likely that, including other perturbation elements to the ones listed in Table 6, also other benchmark functions —

Table 13 Engineering design problems: comparison of results for Problem 3—Pressure vessel (VTR: 6059.701660 (Mezura-Montes et al. 2006), analytical optimum: 6059.714335048436 (Yang et al. 2013))

Algorithm	Best	Mean	Worst	std	FES
DSO*	6059.71433565	6713.56036966	7544.49251821	477.7	30,000
MVDE (De Melo and Carosio 2013)*	6059.71438700	6059.99723600	6090.53352800	2.91028	15,000
QPSO (Coelho, L.d.S, 2010)*	6059.71100000	6464.81660000	7544.49250000	465.1386	8,000
DE (Mezura-Montes et al. 2006)*	6059.70166000	6059.70166000	6059.70166000	1.0e-12	24,000
I-GSO (Shen et al. 2009)*	6059.71400000	6238.80100000	6820.41000000	194.315	26,000
AATM (Wang et al. 2009)*	6059.72550000	6061.98780000	6090.80220000	4.7	30,000
CPSO (He and Wang 2007)*	6061.07770000	6147.13320000	6,363.80410000	86.4545	32,500
CS (Gandomi et al. 2011)*	6059.71400000	6447.73600000	6495.34700000	502.693	375,000
SHO (Dhiman and Kumar 2017)**	5885.57730000	5887.44410000	5892.32070000	2.893	30,000
GWO (Dhiman and Kumar 2017)**	5889.36890000	5891.52470000	5894.62380000	13.91	30,000
PSO (Dhiman and Kumar 2017)**	5891.38790000	6531.50320000	7394.58790000	534.119	30,000
MFO (Dhiman and Kumar 2017)**	6055.63780000	6360.68540000	7023.85210000	365.597	30,000
MVO (Dhiman and Kumar 2017)**	6011.51480000	6477.30500000	7250.91700000	327.007	30,000
SCA (Dhiman and Kumar 2017)**	6137.37240000	6326.76060000	6512.35410000	126.609	30,000
GSA (Dhiman and Kumar 2017)**	11,550.29760000	23,342.29090000	33,226.25260000	5790.625	30,000
GA (Dhiman and Kumar 2017)**	5890.32790000	6264.00530000	7005.75000000	496.128	30,000
HS (Dhiman and Kumar 2017)**	6550.02300000	6643.98700000	8005.43970000	657.523	30,000

References marked with "*" ("**") come from papers that treat both x_1 and x_2 as integer (continuous) variables

Table 14 Engineering design problems: comparison of results for Problem 4—Tension/compression spring (VTR: 0.012665 (Coelho, L.d.S, 2010), analytical optimum: 0.0126652327883 (Celik and Kutucu 2018))

Algorithm	Best	Mean	Worst	std	FES
DSO	0.01266539	0.01288959	0.01407721	3.245e-04	10,000
MVDE (De Melo and Carosio 2013)	0.01266500	0.01266700	0.01271900	2.452e-06	10,000
QPSO (Coelho, L.d.S, 2010)	0.01266600	0.01299600	0.01586900	6.280e-04	2,000
WCA (Eskandar et al. 2012)	0.01266500	0.01274600	0.01295200	8.060e-05	11,750
DE (Mezura-Montes et al. 2006)	0.01266500	0.01266600	0.01267400	2.000e-06	24,000
AATM (Wang et al. 2009)	0.01266800	0.01270800	0.01286100	4.500e-05	25,000
SC (Ray and Liew 2003)	0.01266900	0.01292300	0.01671700	5.900e-04	25,167
GSO (Shen et al. 2009)	0.01266500	0.01270800	0.01299400	5.100e-05	26,000
SHO (Dhiman and Kumar 2017)	0.01267400	0.01268400	0.01271500	2.700e-05	30,000
GWO (Dhiman and Kumar 2017)	0.01267800	0.01269700	0.01272100	4.100e-05	30,000
PSO (Dhiman and Kumar 2017)	0.01319300	0.01481700	0.01786300	2.272e-03	30,000
MFO (Dhiman and Kumar 2017)	0.01275400	0.01402400	0.01723700	1.390e-03	30,000
MVO (Dhiman and Kumar 2017)	0.01281700	0.01446400	0.01784000	1.622e-03	30,000
SCA (Dhiman and Kumar 2017)	0.01271000	0.01284000	0.01299800	7.800e-05	30,000
GSA (Dhiman and Kumar 2017)	0.01287400	0.01343900	0.01421200	2.870e-04	30,000
GA (Dhiman and Kumar 2017)	0.01303600	0.01403600	0.01625100	2.073e-03	30,000
HS (Dhiman and Kumar 2017)	0.01277600	0.01307000	0.01521400	3.750e-04	30,000
CPSO (He and Wang 2007)	0.01267500	0.01273000	0.01292400	5.198e-05	32,800
HPSO (He and Wang 2007)	0.01266500	0.01270700	0.01271910	1.582e-05	81,000

Table 15 Engineering design problems: comparison of results for Problem 5—Welded beam (VTR: 1.724852 (Mezura-Montes et al. 2006; He and Wang 2007; Coello Coello and Becerra 2004))

Algorithm	Best	Mean	Worst	std	FES
DSO	1.72485332	2.00059370	3.18904036	3.401e-01	15,000
MVDE (De Melo and Carosio 2013)	1.72485200	1.72486200	1.72492100	7.883e-06	15,000
GA (Coello Coello and Mezura-Montes 2002)	1.72822600	1.79265400	1.99340800	7.471e-02	24,000
DE (Mezura-Montes et al. 2006)	1.72485200	1.72485300	1.72485400	1.000e-15	24,000
SHO (Dhiman and Kumar 2017)	1.72566100	1.72582800	1.72606400	2.870e-04	30,000
GWO (Dhiman and Kumar 2017)	1.72699500	1.72712800	1.72756400	1.157e-03	30,000
PSO (Dhiman and Kumar 2017)	1.82039500	2.23031000	3.04823100	3.245e-01	30,000
MFO (Dhiman and Kumar 2017)	1.73254100	1.77523100	1.80236400	1.240e-02	30,000
MVO (Dhiman and Kumar 2017)	1.72547200	1.72968000	1.74165100	4.866e-03	30,000
SCA (Dhiman and Kumar 2017)	1.75917300	1.81765700	1.87340800	2.754e-02	30,000
GSA (Dhiman and Kumar 2017)	2.17285800	2.54423900	3.00365700	2.559e-01	30,000
GA (Dhiman and Kumar 2017)	1.87397100	2.11924000	2.32012500	3.482e-02	30,000
HS (Dhiman and Kumar 2017)	1.83625000	1.36352700	2.03524700	1.395e-01	30,000
CPSO (He and Wang 2007)	1.72802400	1.74883100	1.78214300	1.292e-02	>30,000
WCA (Eskandar et al. 2012)	1.72485600	1.72642700	1.74469700	4.290e-03	46,450
HPSO (He and Wang 2007)	1.72485200	1.74904000	1.81429500	4.004e-02	81,000

Table 16 Engineering design problems: comparison of results for Problem 6—rolling element bearing (VTR: 85, 538.48 (Eskandar et al. 2012))

Algorithm	Best	Mean	Worst	std	FES
DSO**	85,539.07823091	83,173.11537690	74,345.08450340	3.154e+03	10,000
DSO*	85,533.18278573	81,645.37176370	39,532.06145658	7.913E+03	10,000
WCA (Eskandar et al. 2012)**	85,538.48000000	83,847.16000000	83,942.71000000	4.883e+02	3,950
MAL-FA (Balande and Shrimankar 2017)	81,701.00000000	81,476.57000000	81,346.26500000	7.100e-01	4,000
SHO (Dhiman and Kumar 2017)**	84,807.11000000	84,791.61000000	84,517.92000000	1.867e+02	30,000
GWO (Dhiman and Kumar 2017)**	81,691.20000000	50,435.02000000	32,761.55000000	1.372e+02	30,000
PSO (Dhiman and Kumar 2017)**	84,002.52000000	82,357.49000000	83,979.25000000	1.396e+04	30,000
MFO (Dhiman and Kumar 2017)**	84,491.27000000	84,353.69000000	84,100.83000000	1.402e+03	30,000
MVO (Dhiman and Kumar 2017)**	83,431.12000000	81,005.23000000	77,992.48000000	3.924e+02	30,000
SCA (Dhiman and Kumar 2017)**	82,276.94000000	78,002.11000000	71,043.11000000	1.711e+03	30,000
GSA (Dhiman and Kumar 2017)**	82,773.98000000	81,198.75000000	80,687.24000000	3.120e+03	30,000
GA (Dhiman and Kumar 2017)**	81,569.53000000	80,398.00000000	79,412.78000000	1.679e+03	30,000
HS (Dhiman and Kumar 2017)**	81,569.52700000	80,397.99800000	79,412.77900000	1.757e+03	30,000

References marked with "*" ("**") come from papers that treat x_3 as integer (continuous) variable (not specified for MAL-FA). This is a maximization problem

Table 17 Engineering design problems: comparison of results for Problem 7—Multiple disk clutch brake (VTR: 0.313656 (Balande and Shrimankar 2017))

Algorithm	Best	Mean	Worst	std	FES
DSO	0.31365661	0.32881603	0.38815006	1.450e-02	500
MAL-FA (Balande and Shrimankar 2017)	0.31365300	0.34365600	0.32862800	1.400e-02	400
WCA (Eskandar et al. 2012)	0.31365600	0.31365600	0.31365600	1.690e-16	500

Table 18 Function values at FES = 2×10^5 for 10D Problems C01–C06

FES		C01	C02	C03	C04	C05	C06
2×10^5	Best	8.5003e-09	3.8635e-09	6014.9972	13.5728	0.00086714	133.2047
	Median	6.0009e-08	3.8772e-08	102386.8624	15.9192	0.2293	1024.2589
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	6.7529e-08	4.6765e-08	163371.5295	15.484	0.83822	1022.4469
	Worst	2.5314e-07	1.5482e-07	556604.1303	16.9142	3.8604	1463.3025
	std	5.0583e-08	3.5216e-08	146200.0583	1.2828	1.171	403.7313
	SR	100%	100%	100%	100%	100%	92%
	$\overline{vio}$	0	0	0	0	0	0.0060728

Table 19 Function values at FES = 2×10^5 for 10D Problems C07–C12

FES		C07	C08	C09	C10	C11	C12
2×10^5	Best	− 161.6822	− 0.0013465	− 0.0049745	− 0.0005071	− 0.1688	3.9879
	Median	− 69.4884	− 0.0012262	0.96806	− 0.00048191	− 0.16848	3.988
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	− 46.7	0.67998	1.2764	− 0.00044859	− 0.1618	7.8585
	Worst	105.5481	17.0261	8.9903	− 0.00011869	− 0.091634	22.9351
	std	74.2285	3.4054	1.8491	8.4269e-05	0.02113	7.2881
	SR	100%	96%	100%	100%	100%	100%
	$\overline{vio}$	0	5.9944	0	0	0	0

Table 20 Function values at FES = 2×10^5 for 10D Problems C13–C18

FES		C13	C14	C15	C16	C17	C18
2×10^5	Best	0.00013357	2.5246	11.7809	51.8361	1.3086	29.25
	Median	0.36329	3.4162	18.0641	69.1149	1.008	945.75
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	5.5	0
	Mean	3.71	3.3558	17.3101	68.8008	1.0501	522.4887
	Worst	76.0457	3.9163	27.4889	81.6813	1.1789	148.2915
	std	15.1055	0.37239	4.6491	7.1553	0.21836	435.246
	SR	100%	100%	100%	100%	0%	88%
	$\overline{vio}$	0	0	0	0	5.3	6.153

where instead DSO shows a slightly worse performance—might be solved more efficiently.

Another interesting achievement of DSO —which is also linked to its efficiency on C18 and C27— is that it obtains the overall lowest mean $\overline{vio}$ (averaged across all problems at all dimensionalities), up to four orders of magnitude smaller that the other competition algorithms. A similar difference in performance is observed on the mean value of $\overline{v}$, again averaged across all problems and dimensionalities. On the other hand, DSO appears less efficient at refining the optimization of the function value: in fact, the average Mean function value (averaged across all problems and dimensionalities) obtained by DSO is up four orders of magnitude higher than that the lowest Mean function value obtained by the other competition algorithms. Similar considerations apply also to the mean Median function value averaged across all problems and dimensionalities. This might indicate that, in the tested settings, DSO is configured to be too explorative and less exploitative, but this is inherently caused by the concurrent GP evolution.

As for the comparison vs HECO-DE (for which we considered the original results reported in Xu et al. (2020)), consid-

Table 21 Function values at FES = 2×10^5 for 10D Problems C19–C24

FES		C19	C20	C21	C22	C23	C24
2×10^5	Best	0.00054195	0.23906	3.9879	1.4689	3.1009	8.6393
	Median	0.0010589	0.57814	4.099	65.2193	3.7242	14.9225
	c	1, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	6633.5922	0	0	0	0	0
	Mean	0.0011544	0.56777	12.1423	11632.9114	3.6759	16.0535
	Worst	0.0017538	0.94109	23.8621	151466.3095	4.0567	21.2057
	std	0.00030301	0.16487	9.0357	38839.4005	0.29251	3.3787
	SR	0%	100%	100%	100%	100%	100%
	$\overline{vio}$	6633.5922	0	0	0	0	0

Table 22 Function values at FES = 2×10^5 for 10D Problems C25–C28

FES		C25	C26	C27	C28
2×10^5	Best	43.9822	1.3724	36.5994	22.3268
	Median	75.3981	1.004	37.7648	53.6628
	c	0, 0, 0	1, 0, 0	0, 0, 0	1, 0, 0
	$\overline{v}$	0	5.5	0	6658.5922
	Mean	72.1937	1.0297	1189.4464	38.0683
	Worst	89.5353	1.004	14049.5047	60.0794
	std	10.022	0.074811	2902.8158	12.3019
	SR	100%	0%	68%	0%
	$\overline{vio}$	0	5.46	725.6411	6659.118

Table 23 Function values at FES = 6×10^5 for 30D Problems C01–C06

FES		C01	C02	C03	C04	C05	C06
6×10^5	Best	8.453e-06	1.9174e-06	171102.9095	13.5728	0.0025315	2810.7647
	Median	6.1005e-05	6.6596e-05	781454.008	17.9096	4.2816	4807.3263
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	0.025879	0.012179	1171659.9264	24.3729	9.6818	4804.2279
	Worst	0.48651	0.29902	6962134.1742	71.6372	69.2083	6698.6136
	std	0.097796	0.05976	1355950.6868	15.2702	14.9798	1120.1365
	SR	100%	100%	100%	100%	100%	100%
	$\overline{vio}$	0	0	0	0	0	0

ering the above mentioned "avg/oneFeas/allFeas" notation, the results can be summarized as follows:

10D: DSO 83.71%/24/20, HECO-DE 84.14%/24/23;
30D: DSO 73.14%/22/19, HECO-DE 82.14%/23/23;
50D: DSO 67.43%/21/17, HECO-DE 81.14%/23/22;
100D DSO 62.43%/19/16, HECO-DE 77.71%/23/19.

From these values, apart from noting that HECO-DE performs and scales better than the L-SHADE44, it can be appreciated that DSO performs very similarly to HECO-DE, at least in 10D. As said earlier though, the proposed method shows a performance decrease when the problem dimensionality increases. Looking at the specific benchmark functions, we note that in 10D HECO-DE cannot find any feasible solution on C17, C19, C26 and C28 (the same as DSO), while it has a 56% SR on C11 (DSO: 100%). On the other hand, DSO shows a lower SR on C06, C08 and C18. In 30D, both HECO-DE and DSO cannot find any feasible solution on C11, C17, C19, C26 and C28, but DSO is not able to solve C22 either; other than that, DSO shows a lower SR on C08, C18 and C27. In 50D, both HECO-DE and DSO are again not

Table 24 Function values at FES = 6×10^5 for 30D Problems C07–C12

FES		C07	C08	C09	C10	C11	C12
6×10^5	Best	− 270.2746	0.00097483	− 0.0025974	0.0002627	9.631	3.9825
	Median	− 75.5792	0.0050751	0.99726	0.00054468	4.3195	3.9826
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 1, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0.027341	0
	Mean	− 72.7935	0.0090872	1.8104	0.00066987	19.2712	6.4639
	Worst	163.8242	0.050826	4.7199	0.0023757	95.2633	14.6142
	std	93.7991	0.011202	1.5952	0.00042909	32.7531	3.5911
	SR	100%	76%	100%	100%	0%	100%
	$\overline{vio}$	0	0.00013405	0	0	0.094463	0

Table 25 Function values at FES = 6×10^5 for 30D Problems C13–C18

FES		C13	C14	C15	C16	C17	C18
6×10^5	Best	0.00024162	1.6983	18.0641	182.2122	1.0317	36.5348
	Median	81.9116	2.0504	24.3473	207.345	1.0298	2327
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	15.5	0
	Mean	2586.4445	2.0566	25.6039	210.9264	1.025	1094.0255
	Worst	40291.4088	2.3174	33.7721	252.8981	1.0317	192.9101
	std	8977.7742	0.13712	4.4429	16.7931	0.045713	1372.9936
	SR	100%	100%	100%	100%	0%	56%
	$\overline{vio}$	0	0	0	0	15.5	10.4419

Table 26 Function values at FES = 6×10^5 for 30D Problems C19–C24

FES		C19	C20	C21	C22	C23	C24
6×10^5	Best	0.0057808	1.3897	3.9826	472366.7077	1.8988	18.0641
	Median	0.0090854	2.0793	14.8219	4495136.9067	2.1222	21.2057
	c	1, 0, 0	0, 0, 0	0, 0, 0	2, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	21374.9081	0	0	199.1447	0	0
	Mean	0.010911	2.0461	17.117	5521644.1785	2.1277	22.0853
	Worst	0.019743	2.634	42.272	20520810.3312	2.2926	30.6305
	std	0.0041677	0.32058	11.6841	5302415.5362	0.11789	2.6472
	SR	0%	100%	100%	0%	100%	100%
	$\overline{vio}$	21374.9081	0	0	214.8362	0	0

capable to solve C11, C17, C19, C26 and C28, but DSO cannot solve C22 and C27 either, and shows a lower SR on C08, C09, C10 and C18. In 100D both HECO-DE and DSO are again not capable to solve C11, C17, C19, C26 and C28, but DSO cannot solve C08, C10, C14, C15 and C16, and shows a lower SR on C09 and C13 yet a slightly higher (16% vs 12%) SR on C18. Finally, concerning the comparisons on the Mean/Median function values, DSO seems slightly less efficient than HECO-DE, confirming our previous observation on the lack of a powerful exploitative behavior.

For the sake of completeness, we report in the Appendix the trends of the minimum fitness error w.r.t. the known optimum at each step of the algorithm (note that the reported error values also include the adaptive penalty, i.e., at each step the minimum fitness error may correspond to an infeasible solution) for each of the 25 DSO runs on the 28 CEC 2017 benchmark problems in 10, 30, 50 and 100D. First of all, it should be noted that the non-monotonic trend observed on some problems depends on the fact that as soon as one feasible solution is found, the fitness values are penalized and this produces an increase in the current generation's fit-

Table 27 Function values at FES = 6×10^5 for 30D Problems C25–C28

FES		C25	C26	C27	C28
6×10^5	Best	196.3494	1.03	41.8851	140.4125
	Median	232.4777	1.0295	28.0164	161.4249
	c	0, 0, 0	1, 0, 0	0, 2, 0	1, 0, 0
	$\overline{v}$	0	15.5	0.07691	21499.3984
	Mean	230.8441	1.0296	3254.5829	172.7516
	Worst	257.6105	1.0286	72155.0331	205.8246
	std	12.8853	0.007088	14381.6831	20.9351
	SR	100%	0%	16%	0%
	$\overline{vio}$	0	15.5	1492.7456	21498.0893

Table 28 Function values at FES = 1×10^6 for 50D Problems C01–C06

FES		C01	C02	C03	C04	C05	C06
1×10^6	Best	0.00014351	0.0014707	463518.1607	13.5731	0.014921	5282.5561
	Median	0.070805	0.20531	4355251.7543	16.927	8.9586	8555.1688
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	0.58552	0.82372	6052482.0065	25.1435	26.6579	8391.8881
	Worst	4.5867	3.989	26106372.8152	58.7027	136.981	10446.3331
	std	1.1252	1.2521	5768824.335	14.2925	35.6857	1281.6453
	SR	100%	100%	100%	100%	100%	100%
	$\overline{vio}$	0	0	0	0	0	0

Table 29 Function values at FES = 1×10^6 for 50D Problems C07–C12

FES		C07	C08	C09	C10	C11	C12
1×10^6	Best	− 524.5674	0.0039624	0.56145	0.00064744	− 3.1424	3.9815
	Median	− 42.4644	0.18226	5.2945	0.00091045	155.6623	7.0732
	c	0, 0, 0	0, 2, 0	0, 0, 0	0, 0, 0	0, 1, 0	0, 0, 0
	$\overline{v}$	0	0.04045	0	0	0.31157	0
	Mean	− 88.6934	0.28231	6.4206	0.0015667	153.6773	7.4132
	Worst	270.9917	0.99163	18.0743	0.009474	1157.4845	18.0238
	std	193.9724	0.29316	4.0997	0.0019254	232.8098	4.342
	SR	100%	4%	88%	92%	0%	100%
	$\overline{vio}$	0	0.13185	0.038173	4.9524e-05	1.1556	0

ness values. Apart from this, we can observe that DSO is quite robust across the different runs on each problem, apart from a few selected cases (e.g. C06, C22 and C27 in 10D, C07 in 50D, and C28 in 50D and 100D). The effect of the restart mechanism is also quite evident in some cases, such as C09 in 30D, 50D and 100D, or C18 in 10D and 30D, where the trends show some negative "peaks" corresponding to the various restarts.

To conclude, the proposed DSO algorithm appears very efficient at finding the feasible solutions on a larger spectrum of problems, even more effectively than some of the best algorithms from the CEC 2017 competition, but less than a very modern algorithm such as HECO-DE. Interestingly, there is a group of problems (C11, C17, C19, C26 and C28) that especially in larger dimensions seem to be harder to solve, even for the state-of-the-art methods. Therefore, the poor performances on these benchmark functions cannot be considered an intrinsic weakness of DSO. As we said, what can be seen instead as a major weak point of the proposed algorithm, at least in the currently tested settings, is that it seems less good than the state-of-the-art methods at performing a "deeper"

Table 30 Function values at FES = 1×10^6 for 50D Problems C13–C18

FES		C13	C14	C15	C16	C17	C18
1×10^6	Best	94.8067	1.3197	21.2057	282.7432	1.0501	58.0271
	Median	772.183	1.5742	30.6305	376.991	1.0498	168.8352
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 0, 0	1, 0, 0
	$\overline{v}$	0	0	0	0	25.5	21.2554
	Mean	3024.7477	1.5641	29.8765	366.8122	1.0496	1676.156
	Worst	20380.0438	1.686	43.1968	408.4069	1.04	3495.342
	std	4729.1676	0.085476	6.1696	28.2092	0.0032775	1898.3861
	SR	100%	100%	100%	100%	0%	4%
	$\overline{vio}$	0	0	0	0	25.5	217.6237

Table 31 Function values at FES = 1×10^6 for 50D Problems C19–C24

FES		C19	C20	C21	C22	C23	C24
1×10^6	Best	0.013138	2.7883	3.9815	417569.6433	1.4136	21.2057
	Median	0.022806	3.6814	14.6177	15322311.9692	1.5869	24.3473
	c	1, 0, 0	0, 0, 0	0, 0, 0	2, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	36116.224	0	0	509.5447	0	0
	Mean	1.3726	3.7006	16.4359	16987279.2639	1.5731	24.4729
	Worst	33.3324	4.5507	80.2935	31057134.2101	1.6798	27.4889
	std	6.6585	0.46184	17.9704	12801199.3995	0.067978	2.3086
	SR	0%	100%	100%	0%	100%	100%
	$\overline{vio}$	36117.9157	0	0	494.909	0	0

Table 32 Function values at FES = 1×10^6 for 50D Problems C25–C28

FES		C25	C26	C27	C28
1×10^6	Best	370.7078	1.0503	34.9441	290.5316
	Median	395.8405	1.05	89.1198	328.6172
	c	0, 0, 0	1, 0, 0	1, 0, 0	1, 0, 0
	$\overline{v}$	0	25.5	20.9596	36335.845
	Mean	396.9715	1.05	1318.3487	306.2171
	Worst	427.2565	1.05	16234.3393	331.4477
	std	14.5706	0.00094931	3330.9514	23.8294
	SR	100%	0%	0%	0%
	$\overline{vio}$	0	25.5	391.9888	36334.7914

exploitation of the function value once the feasible region is found.

5 Conclusions

We introduced Dual Search Optimization (DSO), a kind of co-evolutionary algorithm with adaptive penalty function for solving constrained optimization problems. The most important aspect of the proposed algorithm is its capability to adapt its own perturbation logics to the problem. However, such adaptation takes time because these perturbations are evolved at runtime during the optimization process. In other words, DSO can discover efficient perturbations to solve the problem at hand, provided a sufficient computational budget.

Our experimental analysis investigated how DSO, which can be seen as a self-improving algorithm, compares to algorithms carefully designed by researchers. When compared to a large number of state-of-the-art metaheuristics on various engineering design problems, DSO achieved in general a comparable —and in some cases better— performance. Furthermore, DSO was in general able to achieve or approx-

Table 33 Function values at FES = 2×10^6 for 100D Problems C01–C06

FES		C01	C02	C03	C04	C05	C06
2×10^6	Best	0.013257	1.4065	3959978.9328	15.9192	16.2678	15223.8344
	Median	63.6956	61.5066	16960993.7838	15.9214	94.1919	18084.1767
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	0	0	0	0	0	0
	Mean	178.4512	327.9204	25169748.4402	33.29	112.6778	18099.5698
	Worst	1154.8345	5752.8437	80829455.9264	154.2943	232.6623	21107.8144
	std	255.6544	1141.4825	24516520.9291	35.2685	49.6186	1585.0829
	SR	100%	100%	100%	100%	100%	100%
	$\overline{vio}$	0	0	0	0	0	0

Table 34 Function values at FES = 2×10^6 for 100D Problems C07–C12

FES		C07	C08	C09	C10	C11	C12
2×10^6	Best	− 778.1561	0.31191	8.9883	0.0037086	89.0433	3.9807
	Median	− 55.7547	1.7718	19.1106	0.030235	408.2119	5.0433
	c	0, 0, 0	2, 0, 0	0, 1, 0	0, 2, 0	1, 0, 0	0, 0, 0
	$\overline{v}$	0	4.9403	0.31667	0.018535	0.57879	0
	Mean	− 99.8666	1.9368	14.6308	0.064294	351.8465	7.6933
	Worst	313.6538	4.1631	17.8999	0.40532	1283.1591	30.9209
	std	319.2343	1.1337	3.6931	0.083882	396.8124	6.1863
	SR	100%	0%	44%	0%	0%	100%
	$\overline{vio}$	0	7.3911	0.21897	0.11634	0.94302	0

Table 35 Function values at FES = 2×10^6 for 100D Problems C13–C18

FES		C13	C14	C15	C16	C17	C18
2×10^6	Best	92.7113	0.93216	21.2057	703.7166	1.0994	40.6712
	Median	1946.2338	1.0386	27.4889	735.1325	1.0999	278.6418
	c	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	1, 0, 0	1, 0, 0
	$\overline{v}$	0	0	0	0	50.5	42.2872
	Mean	601739.974	1.0336	31.2588	740.7874	1.0998	332.2223
	Worst	14232606.8413	1.1009	43.1968	791.6812	1.0997	579.9318
	std	2841012.372	0.044822	5.807	28.3796	0.0013193	486.1305
	SR	88%	100%	100%	100%	0%	16%
	$\overline{vio}$	9.6522	0	0	0	50.5	34.0337

imate the VTR of each problem in a smaller number of FES (compared to the other algorithms), despite the additional computational cost due to the self-adaptation. It is also noticeable that DSO found a new best solution for one of the tested engineering design problems, i.e., the design of a rolling element bearing (under the assumption of continuity of x_3, made to compare our results with the most recent literature).

Tested on the CEC 2017 benchmark for constrained optimization, DSO also showed very good results, especially on the lower dimensionality (10D), where it resulted on par with the state-of-the-art and even capable of solving —albeit not in all runs— two problems that could not be solved by the CEC 2017 competition algorithms. Overall, DSO seems to be very good at finding feasible solutions on most of the problems at different dimensionalities, while the additional overhead introduced by the search for optimal perturbations represents an obstacle to a better exploitation, especially at larger dimensionalities. As said, in fact, the adaptation of the perturbation logics can be time-consuming, and its benefits can be more evident only on some particularly hard-to-solve problems. In simpler optimization cases, using handcrafted perturbations may be more effective. In this study, we partially addressed

Table 36 Function values at FES = 2×10^6 for 100D Problems C19–C24

FES		C19	C20	C21	C22	C23	C24
2×10^6	Best	0.047496	6.498	3.9808	34637373.1793	0.96582	24.3473
	Median	0.14808	7.701	14.7001	238489225.9708	1.0529	27.4889
	c	1, 0, 0	0, 0, 0	0, 0, 0	1, 0, 0	0, 0, 0	0, 0, 0
	$\overline{v}$	72969.5147	0	0	2641.3478	0	0
	Mean	5.131	7.8226	31.527	189869263.6285	1.0475	29.2482
	Worst	112.9504	8.7549	216.0218	341046165.0936	1.1107	36.9136
	std	22.5901	0.57438	56.0827	100806341.4905	0.042057	3.152
	SR	0%	100%	100%	0%	100%	100%
	$\overline{vio}$	72976.0029	0	0	2587.5993	0	0

Table 37 Function values at FES = 2×10^6 for 100D Problems C25–C28

FES		C25	C26	C27	C28
2×10^6	Best	735.1325	1.1005	264.8755	691.1446
	Median	785.398	1.0998	1549.9332	603.2635
	c	0, 0, 0	1, 0, 0	1, 0, 0	1, 0, 0
	$\overline{v}$	0	50.5	1231.9415	73429.7383
	Mean	781.5653	1.1003	11163.3156	645.3978
	Worst	824.6679	1.1012	32259.4912	673.6193
	std	22.9836	0.00075745	9487.5884	32.1787
	SR	100%	0%	0%	0%
	$\overline{vio}$	0	50.5	2539.6674	73430.421

this issue by including a reference perturbation (namely, the "rand/1" mutation strategy from DE) so to avoid the GP part of the algorithm to start from scratch, hence requiring even more time to find effective perturbations. Another possibility would be to include a richer pool of reference perturbations, e.g. including other well-known mutations used in DE as well as other perturbation mechanisms used in other metaheuristics such as PSO, ES (including CMA-ES), etc. This pool would provide a rich material for further evolution through mutations and crossover in the GP part. Other alternative would be to design proxies for the evaluation of the perturbations, i.e. ways to immediately discard invalid perturbations without actually applying them to the current solutions. All these possibilities should be further investigated in future works.

Overall, our results prove that DSO is a viable alternative for solving engineering optimization problems, especially when the number of variables is moderate (i.e., less than 30). From a practitioner's point of view, the most attractive feature of the proposed algorithm is its self-improving capability, as the user is not required to perform any specific tuning.

Still, there are several directions that could be followed to improve the algorithm presented in this paper. For instance, here we employed an adaptive constraint handling technique, but other CHTs (or ensembles thereof) could be included in the algorithm. In principle, even the CHT itself could be evolved by means of GP, which might be an interesting research question. Another possible improvement would be to add one or more local search algorithms, or include additional perturbation elements in the GP part of the algorithm, to improve its exploitation capabilities or exploit algorithmic/domain knowledge and guide the GP part towards more effective perturbations.

A fitness error trends on CEC 2017 problems

Figures 3, 4, 5, 6 show the fitness error trends (taking into account the adaptive penalty) of each of the 25 DSO runs on the 28 CEC 2017 benchmark problems in 10, 30, 50 and 100D. The figures also report the mean and the median error trend for each problem, depicted respectively as thick red and blue lines.

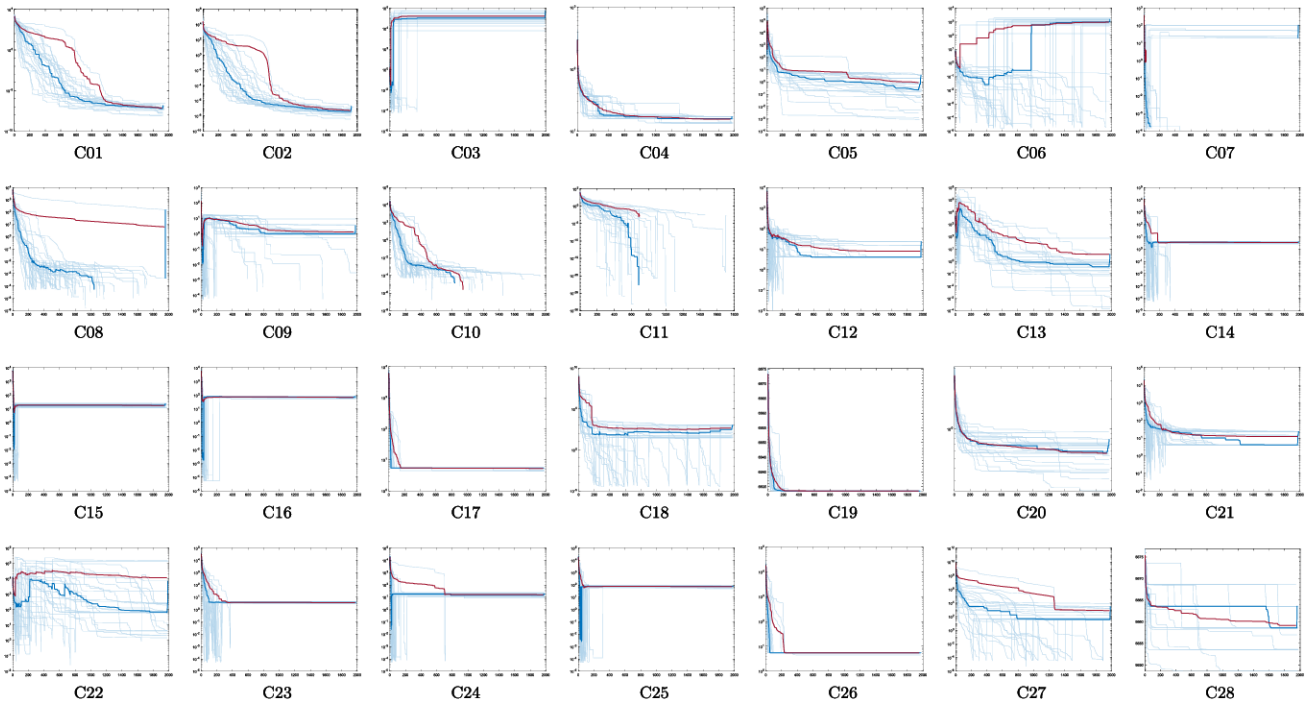

Fig. 3 Fitness trends of DSO on the CEC 2017 benchmark functions ($D = 10$). The x-axis and y-axis show, respectively, the number of function evaluations and the fitness error (including the adaptive penalty, log scale). Every plot shows the trends of 25 runs, their mean (thick red line) and median (thick blue line)

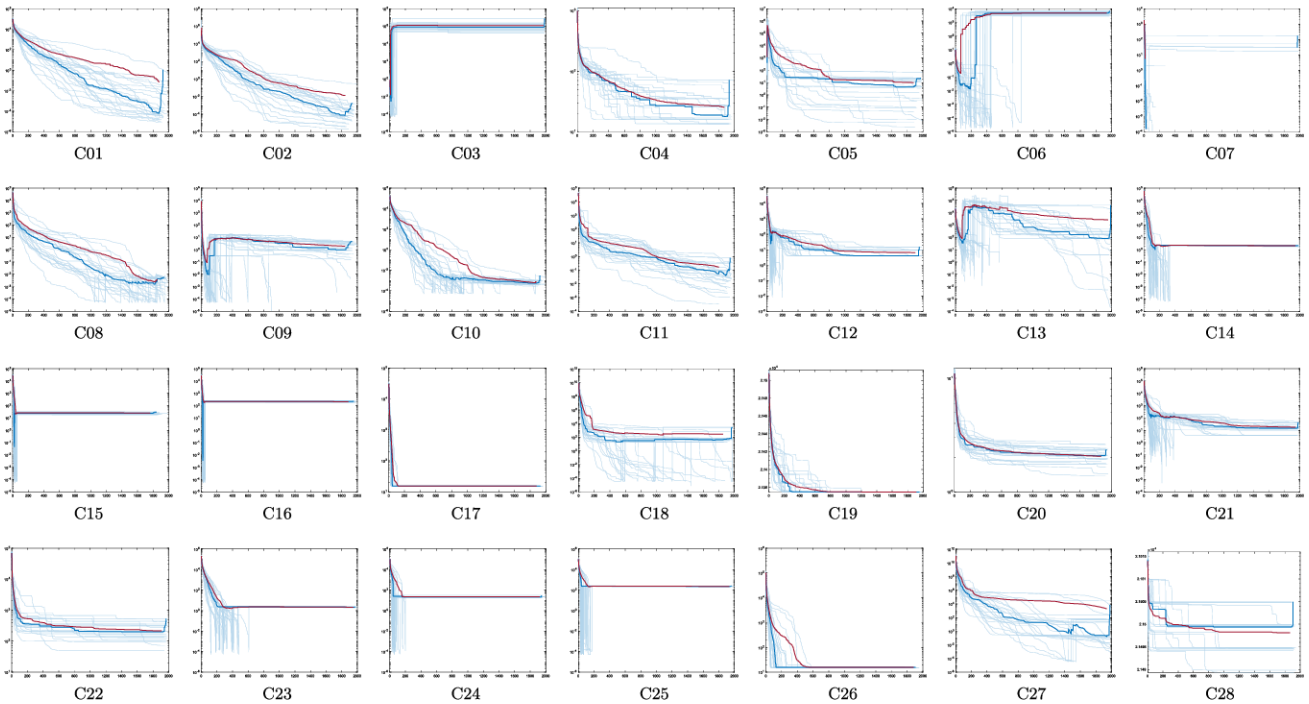

Fig. 4 Fitness trends of DSO on the CEC 2017 benchmark functions ($D = 30$). The x-axis and y-axis show, respectively, the number of function evaluations and the fitness error (including the adaptive penalty, in log scale). Every plot shows the trends of 25 runs, their mean (thick red line) and median (thick blue line)

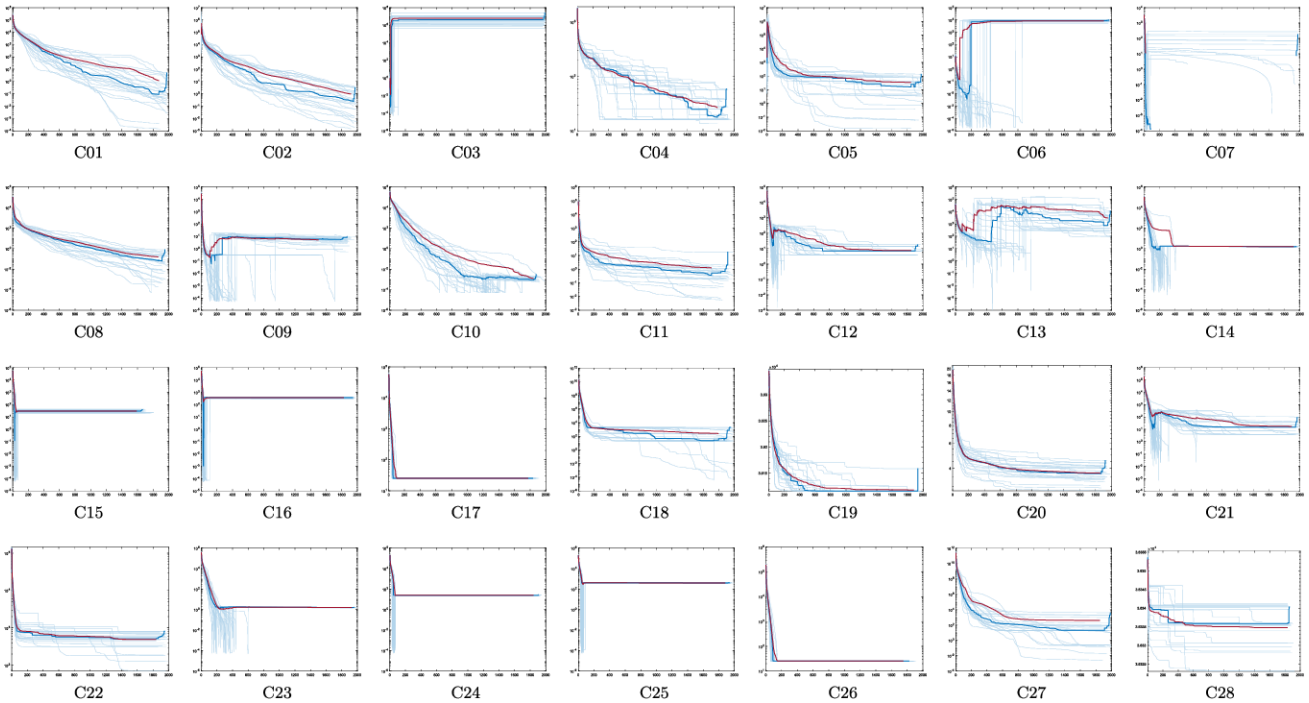

Fig. 5 Fitness trends for DSO on the CEC 2017 benchmark functions ($D = 50$). The x-axis and y-axis show, respectively, the number of function evaluations and the fitness error (including the adaptive penalty, in log scale). Every plot shows the trends of 25 runs, their mean (thick red line) and median (thick blue line)

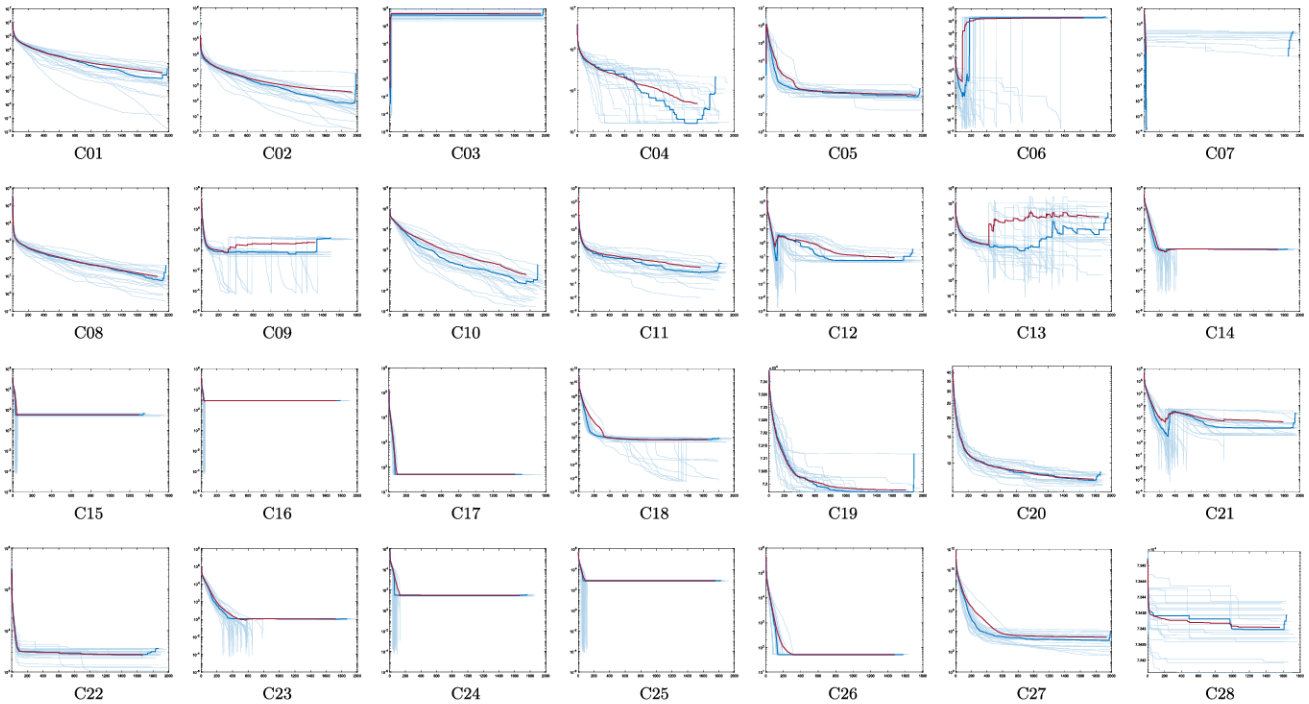

Fig. 6 Fitness trends for DSO on the CEC 2017 benchmark functions ($D = 100$). The x-axis and y-axis show, respectively, the number of function evaluations and the fitness error (including the adaptive penalty, in log scale). Every plot shows the trends of 25 runs, their mean (thick red line) and median (thick blue line)

Author Contributions All authors contributed to the design of this study. Experiments and data collection were performed by V. V. de Melo and A. M. Nascimento. The analysis was performed by V. V. de Melo and G. Iacca. The manuscript was written by all authors. All authors read and approved the final manuscript.

Funding Open access funding provided by Università degli Studi di Trento within the CRUI-CARE Agreement.

Data availability The raw data are available upon request.

Code availability The code is available upon request.

Declarations

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

References

Aguirre A, Muñoz Zavala A, Villa Diharce E, Botello Rionda S (2007) COPSO: Constrained optimization via PSO algorithm. Technical Report I-07-04, Center of Research in Mathematics (CIMAT)

Akay B, Karaboga D (2012) Artificial bee colony algorithm for large-scale problems and engineering design optimization. J Intel Manuf 23(4)

Ali M, Zhu W (2013) A penalty function-based differential evolution algorithm for constrained global optimization. Comput Optim Appl 54(3):707–739

Arnold DV, Hansen N (2012) A (1+1)-CMA-ES for constrained optimisation. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 297–304

Askarzadeh A (2016) A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. Comput Struct 169:1–12

Balande U, Shrimankar D (2017) An oracle penalty and modified augmented Lagrangian methods with firefly algorithm for constrained optimization problems. Oper Res 1–26

Bao Q, Wang M, Dai G, Chen X, Song Z, Li S (2023) A dual-population based bidirectional coevolution algorithm for constrained multi-objective optimization problems. Expert Syst Appl 215:119258

Barbosa HJ (1999) A coevolutionary genetic algorithm for constrained optimization. IEEE Congress Evol Comput (CEC) 3:1605–1611 (**IEEE**)

Barkat Ullah A, Sarker R, Lokan C (2011) Handling equality constraints with agent-based memetic algorithms. Memet Comput 3(1):51–72

Bernardino H, Barbosa H, Lemong A (2007) A hybrid genetic algorithm for constrained optimization problems in mechanical engineering. In: IEEE Congress on Evolutionary Computation (CEC), pp. 646–653

Bonyadi M, Li X, Michalewicz Z (2013) A hybrid particle swarm with velocity mutation for constraint optimization problems. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 1–8

Brajevic I, Tuba M, Subotic M (2011) Performance of the improved artificial bee colony algorithm on standard engineering constrained problems. Int J Math Comput Simul 5(1):135–143

Cagnina LC, Esquivel SC, Coello Coello CA (2008) Solving engineering optimization problems with the simple constrained particle swarm optimizer. Informatica 32:319–326

Cai X, Sun Q, Li Z, Xiao Y, Mei Y, Zhang Q, Li X (2022) Cooperative coevolution with knowledge-based dynamic variable decomposition for bilevel multiobjective optimization. IEEE Trans Evol Comput 26(6):1553–1565. https://doi.org/10.1109/TEVC.2022.3154057

Celik Y, Kutucu H (2018) Solving the tension/compression spring design problem by an improved firefly algorithm. IDDM 1(2255):1–7

Coelho LS (2010) Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems. Expert Syst Appl 37(2):1676–1683

Coello Coello CA (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Comput Ind 41:113–127

Coello Coello CA (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Method Appl Mech Eng 191(11–12):1245–1287

Coello Coello CA, Becerra RL (2004) Efficient evolutionary optimization through the use of a cultural algorithm. Eng Optim 36(2):219–236

Coello Coello CA, Mezura-Montes E (2002) Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Adv Eng Inform 16(3):193–203

Collange G, Delattre N, Hansen N, Quinquis I, Schoenauer M (2010) Multidisciplinary optimization in the design of future space launchers. In: Multidisciplinary Design Optimization in Computational Mechanics, pp. 459–468. John Wiley & Sons, Inc

Cuevas E, Cienfuegos M (2014) A new algorithm inspired in the behavior of the social-spider for constrained optimization. Expert Syst Appl 41(2):412–425

Das S, Mullick SS, Suganthan P (2016) Recent advances in differential evolution—an updated survey. Swarm Evol Comput 27:1–30

de Melo VV, Carosio GLC (2012) Evaluating differential evolution with penalty function to solve constrained engineering problems. Expert Syst Appl 39(9):7860–7863

De Melo VV, Carosio GL (2013) Investigating multi-view differential evolution for solving constrained engineering design problems. Expert Syst Appl 40(9):3370–3377

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

de Melo VV, Banzhaf W (2017) Drone squadron optimization: a novel self-adaptive algorithm for global numerical optimization. Neural Comput. Appl. pp. 1–28

de Melo VV, Iacca G (2014) A CMA-ES-based 2-stage memetic framework for solving constrained optimization problems. In: IEEE Symposium on Foundations of Computational Intelligence (FOCI), pp. 143–150. IEEE

Dhiman G, Kumar V (2017) Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. Adv Eng Softw 114:48–70

Eskandar H, Sadollah A, Bahreininejad A, Hamdi M (2012) Water cycle algorithm-a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput Struct 110:151–166

Fan Q, Yan X (2012) Differential evolution algorithm with co-evolution of control parameters and penalty factors for constrained optimization problems. Asia-Pac J Chem Eng 7(2):227–235

Fioravanzo S, Iacca G(2019) Evaluating map-elites on constrained optimization problems. In: Genetic and Evolutionary Computation Conference Companion, pp. 253–254. ACM

Fz H, Wang L, He Q (2007) An effective co-evolutionary differential evolution for constrained optimization. Appl Math Comput 186(1):340–356

Gandomi A, Yang XS, Alavi A (2011) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 1–19

Gao WF, Yen GG, Liu SY (2014) A dual-population differential evolution with coevolution for constrained optimization. IEEE Trans Cybern 45(5):1108–1121

Ghasemishabankareh B, Li X, Ozlen M (2016) Cooperative coevolutionary differential evolution with improved augmented lagrangian to solve constrained optimisation problems. Inf Sci 369:441–456

Gieseke F, Kramer O (2013) Towards non-linear constraint estimation for expensive optimization. In: Applications of Evolutionary Computation, pp. 459–468. Springer

Goh CK, Lim D, Ma L, Ong YS, Dutta PS (2011) A surrogate-assisted memetic co-evolutionary algorithm for expensive constrained optimization problems. In: IEEE Congress of Evolutionary Computation (CEC), pp. 744–749. IEEE

Guedria NB (2016) Improved accelerated PSO algorithm for mechanical engineering optimization problems. Appl Soft Comput 40:455–467

Guo Y, Cao X, Yin H, Tang Z (2007) Coevolutionary optimization algorithm with dynamic sub-population size. Int J Innov Comput Inform Control 3(2):435–448

Gutérrez-Antonio C, Briones-Ramírez A, Jiménez-Gutiérrez A (2011) Optimization of petlyuk sequences using a multi objective genetic algorithm with constraints. Comput Chem Eng 35(2):236–244

Hamza NM, Sarker RA, Essam DL, Deb K, Elsayed SM (2014) A constraint consensus memetic algorithm for solving constrained optimization problems. Eng Optim 46(11):1447–1464

Handa H, Katai O, Konishi T, Baba M (1999) Coevolutionary genetic algorithms for solving dynamic constraint satisfaction problems. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 252–257

Handoko S, Kwoh CK, Ong YS (2010) Feasibility structure modeling: an effective chaperone for constrained memetic algorithms. IEEE Trans Evol Comput 14(5):740–758

He Q, Wang L (2007) An effective co-evolutionary particle swarm optimization for constrained engineering design problems. Eng Appl Artif Intel 20(1):89–99

He Q, Wang L (2007) A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Appl Math Comput 186(2):1407–1422

He Q, Wang L, Huang Fz (2008) Nonlinear constrained optimization by enhanced co-evolutionary PSO. In: IEEE Congress on Evolutionary Computation (CEC), pp. 83–89. IEEE

Hu Z, Gong W, Pedrycz W, Li Y (2023) Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization. Swarm Evol Comput 83:101387

Huang V, Qin A, Suganthan P (2006) Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 17–24

Hu X, Eberhart RC, Shi Y (2003) Engineering optimization with particle swarm. In: IEEE Swarm Intelligence Symposium, pp. 53–57

Kashan AH (2011) An efficient algorithm for constrained global optimization and application to mechanical engineering design: league championship algorithm (LCA). Comput Aided Des 43(12):1769–1792

Kim J, Tahk MJ (2001) Co-evolutionary computation for constrained min-max problems and its applications for pursuit-evasion games. In: Congress on Evolutionary Computation (CEC), vol. 2, pp. 1205–1212. IEEE

Kononova AV, Caraffini F, Bäck T (2020) Differential evolution outside the box. arXiv preprint arXiv:2004.10489 (2020)

Kou X, Liu S, Zhang J, Zheng W (2009) Co-evolutionary particle swarm optimization to solve constrained optimization problems. Comput Math Appl 57(11–12):1776–1784

Koza JR (1993) Genetic programming—on the programming of computers by means of natural selection. MIT Press

Kramer O, Barthelmes A, Rudolph G (2009) Surrogate constraint functions for CMA evolution strategies. Adv Artif Intell 5803:169–176

Kramer O, Schlachter U, Spreckels V (2013) An adaptive penalty function with meta-modeling for constrained problems. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1350–1354

Krohling RA, dos Santos Coelho L (2006) Coevolutionary particle swarm optimization using gaussian distribution for solving constrained optimization problems. IEEE Trans Syst Man Cybern 36(6):1407–1416

Kusakci AO, Can M (2013) A novel evolution strategy for constrained optimization in engineering design. In: International Symposium on Information, Communication and Automation Technologies (ICAT), pp. 1–6

Kwan RS, Mistry P (2003) A co-evolutionary algorithm for train timetabling. IEEE Congress Evol Comput (CEC) 3:2142–2148 (**IEEE**)

Leguizamón G, Coello Coello CA (2009) Boundary search for constrained numerical optimization problems with an algorithm inspired on the ant colony metaphor. IEEE Trans Evol Comput 13(2):350–368

Liang JJ, Zhigang S, Zhihui L (2010) Coevolutionary comprehensive learning particle swarm optimizer. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE

Liu B, Ma H, Zhang X (2007) A co-evolutionary differential evolution algorithm for constrained optimization. IEEE Int Conf Nat Comput (ICNC) 4:51–57 (**IEEE**)

Liu H, Cai Z, Wang Y (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10:629–640

Liu ZZ, Wang BC, Tang K (2022) Handling constrained multiobjective optimization problems via bidirectional coevolution. IEEE Trans Cybern 52(10):10163–10176. https://doi.org/10.1109/TCYB.2021.3056176

Liu B, Ma H, Zhang X, Zhou Y (2007) A memetic co-evolutionary differential evolution algorithm for constrained optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 2996–3002. IEEE

Long Q (2014) A constraint handling technique for constrained multiobjective genetic algorithm. Swarm Evol Comput 15:66–79

Machado-Coelho T, Machado AMC, Jaulin L, Ekel P, Pedrycz W, Soares GL (2017) An interval space reducing method for constrained problems with particle swarm optimization. Appl Soft Comput 59:405–417

Maesani A, Iacca G, Floreano D (2016) Memetic viability evolution for constrained optimization. IEEE Trans Evol Comput 20(1):125–144

Mazhoud I, Hadj-Hamou K, Bigeon J, Joyeux P (2013) Particle swarm optimization for solving engineering problems: a new constraint-handling mechanism. Eng Appl Artif Itel 26(4):1263–1273

Mezura-Montes E, Coello Coello CA (2005) A simple multimembered evolution strategy to solve constrained optimization problems. IEEE Trans Evol Comput 9(1):1–17

Mezura-Montes E, Coello Coello CA (2011) Constraint-handling in nature-inspired numerical optimization: past, present and future. Swarm Evol Comput 1(4):173–194

Mezura-Montes E, Miranda-Varela ME, del Carmen Gomez-Ramon R (2010) Differential evolution in constrained numerical optimization: an empirical study. Inf Sci 180(22):4223–4262

Mezura-Montes E, Coello Coello CA, Tun-Morales EI (2004) Simple feasibility rules and differential evolution for constrained optimization. In: Advances in Artificial Intelligence, pp. 707–716. Springer

Mezura-Montes E, Coello Coello CA, Velázquez-Reyes J (2006) Increasing successful offspring and diversity in differential evolution for engineering design. In: International Conference on Adaptive Computing in Design and Manufacture, pp. 131–139. The Institute for People-centred Computation (IP-CC)

Mezura-Montes E, Palomeque-Ortiz A (2009) Self-adaptive and deterministic parameter control in differential evolution for constrained optimization. In: Constraint-Handling in Evolutionary Optimization, pp. 95–120. Springer

Mezura-Montes E, Palomeque-Ortiz AG (2009) Parameter control in differential evolution for constrained optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1375–1382

Michalewicz Z (1995) A survey of constraint handling techniques in evolutionary computation methods. In: Conference on Evolutionary Programming, pp. 135–155

Michalewicz Z, Nazhiyath G (1995) Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. IEEE Int Conf Evol Comput 2:647–651 (**IEEE**)

Michalewicz Z, Schoenauer M (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4:1–32

Michalewicz Z, Dasgupta D, Riche RL, Schoenauer M (1996) Evolutionary algorithms for constrained engineering problems. Comput Ind Eng 30(4):851–870

Ming F, Gong W, Wang L, Lu C (2022) A tri-population based co-evolutionary framework for constrained multi-objective optimization problems. Swarm Evol Comput 70:101055. https://doi.org/10.1016/j.swevo.2022.101055 (https://www.sciencedirect.com/science/article/pii/S221065022200027X)

Mohemmed AW, Sahoo NC, Geok TK (2010) Hybrid co-evolutionary particle swarm optimization and noising metaheuristics for the delay constrained least cost path problem. J Heuristics 16(4):593–616

Noman N, Iba H (2008) Accelerating differential evolution using an adaptive local search. IEEE Trans Evol Comput 12(1):107–125

Nordmoen J, Nygaard TF, Samuelsen E, Glette K (2020) On restricting real-valued genotypes in evolutionary algorithms. arXiv preprint arXiv:2005.09380

Pant M, Thangaraj R, Abraha A (2009) Low discrepancy initialized particle swarm optimization for solving constrained optimization problems. Fundam Inform 95:511–531

Paredis J (1994) Co-evolutionary constraint satisfaction. In: Parallel Problem Solving from Nature (PPSN), pp. 46–55. Springer

Pescador Rojas M, Coello Coello CA (2012) A memetic algorithm with simplex crossover for solving constrained optimization problems. In: World Automation Congress (WAC), pp. 1–6

Polakova R (2017) L-SHADE with competing strategies applied to constrained optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1683–1689. IEEE

Purcaru C, Precup RE, Iercan D, Fedorovici LO, David RC, Dragan F (2013) Optimal robot path planning using gravitational search algorithm. Int J Artif Intell Trans 10(S13):1–20

Rasheed K (1998) An adaptive penalty approach for constrained genetic-algorithm optimization. In: Genetic Programming Conference, pp. 584–590

Ray T, Liew KM (2003) Society and civilization: an optimization algorithm based on the simulation of social behavior. IEEE Trans Evol Comput 7(4):386–396

Salcedo-Sanz S (2009) A survey of repair methods used as constraint handling techniques in evolutionary algorithms. Comput Sci Rev 3(3):175–192

Schmidt MD, Lipson H (2008) Coevolution of fitness predictors. IEEE Trans Evol Comput 12(6):736–749

Sergienko RB, Semenkin ES (2010) Competitive cooperation for strategy adaptation in coevolutionary genetic algorithm for constrained optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–6. IEEE

Shen H, Zhu Y, Niu B, Wu Q (2009) An improved group search optimizer for mechanical design optimization problems. Prog Nat Sci 19(1):91–97

Sipper M, Moore JH (2019) OMNIREP: originating meaning by coevolving encodings and representations. Memet Comput 11(3):251–261

Sipper M, Moore JH, Urbanowicz RJ (2019) Solution and fitness evolution (SAFE): Coevolving solutions and their objective functions. In: European Conference on Genetic Programming, pp. 146–161

Sun J, Garibaldi JM (2010) A novel memetic algorithm for constrained optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 549–556

Tahk MJ, Sun BC (2000) Coevolutionary augmented Lagrangian methods for constrained optimization. IEEE Trans Evol Comput 4(2):114–124

Takahama T, Sakai S (2006) Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8

Takahama T, Sakai S (2010) Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–9

Takahama T, Sakai S (2012) Efficient constrained optimization by the ε constrained rank-based differential evolution. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–8

Takahama T, Sakai S (2013) Efficient constrained optimization by the ε constrained differential evolution with rough approximation using kernel regression. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1334–1341

Takahama T, Sakai S, Iwane N (2005) Constrained optimization by the ε constrained hybrid algorithm of particle swarm optimization and genetic algorithm. In: Advances in Artificial Intelligence, vol. 3809, pp. 389–400. Springer

Trivedi A, Sanyal K, Verma P, Srinivasan D (2017) A unified differential evolution algorithm for constrained optimization problems. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1231–1238. IEEE

Tvrdík J, Poláková R (2017) A simple framework for constrained problems with application of L-SHADE44 and IDE. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1436–1443. IEEE

Wang Y, Cai Z, Zhou Y (2009) Accelerating adaptive trade-off model using shrinking space technique for constrained evolutionary optimization. Int J Numer Meth Eng 77(11):1501–1534

Wang Y, Cai Z, Zhou Y, Fan Z (2009) Constrained optimization based on hybrid evolutionary algorithm and adaptive constraint-handling technique. Struct Multidiscip Optim 37(4):395–413

Wang R, Lehman J, Clune J, Stanley KO (2019) POET: open-ended coevolution of environments and their optimized solutions. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 142–151

Wu G, Mallipeddi R, Suganthan P (2017) Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization. Tech. rep., National University of Defense Technology, Kyungpook National University and Nanyang Technological University, Singapore, Technical Report

Xu P, Luo W, Lin X, Zhang J, Qiao Y, Wang X (2021) Constraint-objective cooperative coevolution for large-scale constrained optimization. ACM Trans Evol Learn Optim. https://doi.org/10.1145/3469036

Xu T, He J, Shang C (2020) Helper and equivalent objectives: efficient approach for constrained optimization. IEEE Trans Cybern

Yang XS, Deb S (2010) Engineering optimisation by cuckoo search. Int J Math Mod Num Optim 1(4):330–343

Yang XS, Huyck C, Karamanoglu M, Khan N (2013) True global optimality of the pressure vessel design problem: a benchmark for bio-inspired optimisation algorithms. Int J Bio-Inspir Comput 5(6):329–335

Zamuda A (2017) Adaptive constraint handling and success history differential evolution for CEC 2017 constrained real-parameter optimization. In: IEEE Congress on Evolutionary Computation (CEC), pp. 2443–2450. IEEE

Zhang M, Luo W, Wang X (2008) Differential evolution with dynamic stochastic selection for constrained optimization. Inf Sci 178(15):3043–3074

Zhang W, Yen GG, He Z (2014) Constrained optimization via artificial immune system. IEEE Trans Cybern 44(2):185–198

Zou D, Liu H, Gao L, Li S (2011) A novel modified differential evolution algorithm for constrained optimization problems. Comput Math Appl 61(6):1608–1623