Universiteit
Leiden
The Netherlands

Differential evolution outside the box

Kononova, A.V.; Caraffini, F.; Bäck, T.H.W.

Citation

Kononova, A. V., Caraffini, F., & Bäck, T. H. W. (2021). Differential evolution outside the box. *Information Sciences*, *581*, 587-604. doi:10.1016/j.ins.2021.09.058

Version:	Publisher's Version
License:	Licensed under Article 25fa Copyright Act/Law (Amendment Taverne)
Downloaded from:	https://hdl.handle.net/1887/3276951

Note: To cite this publication please use the final published version (if applicable).

Information Sciences 581 (2021) 587–604

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Differential evolution outside the box

Anna V. Kononova [a], Fabio Caraffini [b,*], Thomas Bäck [a]

[a] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
[b] Institute of Artificial Intelligence, De Montfort University, United Kingdom

ARTICLE INFO

Article history:
Received 28 November 2020
Received in revised form 12 September 2021
Accepted 28 September 2021
Available online 30 September 2021

Keywords:
Differential evolution
Infeasibility
Constraints handling
Evolutionary computing
Box-constrained problem
Metaheuristic

ABSTRACT

This paper investigates how often the popular configurations of Differential Evolution generate solutions outside the feasible domain. Following previous publications in the field, we argue that what the algorithm does with such solutions and how often this has to happen is important for the overall performance of the algorithm and interpretation of results. Based on observations therein, we conclude that significantly more solutions than what is usually assumed by practitioners need to undergo some sort of 'correction' to conform with the definition of the problem's search domain. A wide range of popular Differential Evolution configurations is considered in this study. Conclusions are made regarding the effect the Differential Evolution components and parameter settings have on the distribution of proportions of infeasible solutions generated in a series of independent runs. Results shown in this study suggest strong dependencies between proportions of generated infeasible solutions and every aspect mentioned above. Further investigation of the distribution of proportions of generated infeasible solutions is required.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Typical optimisation problems used for comparing and benchmarking nonlinear optimisation heuristics are *hypercube-constrained* (also known as problems with box constraints), i.e., they are of the form

$$f : \mathbf{D} = \times_{i=1}^{n} [a_i, b_i] \rightarrow \mathbb{R}$$

where $-\infty < a_i < b_i < \infty$ and $\mathbf{D} \subset \mathbb{R}^n$ is called the *feasible region* in the following. Such *box constraints* represent the lowest complexity of arbitrary inequality constraints $g_j(\mathbf{x}) \leqslant 0$, while they are also omnipresent in most real-world applications where feasible ranges of variables are typically known or can well be estimated. Consequently, any optimisation algorithm, including nonlinear optimisation heuristics, should be able to deal with such constraints by means of a *constraint handling method*. Such a method deals with *infeasible solution* (IS) candidates $\mathbf{x} \notin \mathbf{D}$ by means of a suitable approach, involving concepts such as, e.g., ignoring or repairing them.

In nonlinear optimisation heuristics inspired by nature, the infeasible components of a solution are generated by the mutation operator, which is expected to help explore regions of the search space outside the scope of the crossover operator and then converge towards solution candidates for which f is minimised or maximised. Intuitively, this search process is disrupted and thus lacks the ability to adapt itself to the properties of the objective function f when it generates many infeasible solutions during the course of the search.

* Corresponding author.
E-mail address: fabio.caraffini@dmu.ac.uk (F. Caraffini).

https://doi.org/10.1016/j.ins.2021.09.058
0020-0255/© 2021 Elsevier Inc. All rights reserved.

In this paper, we present an empirical investigation of the proportion of infeasible solutions generated for various variants and parameter settings of Differential Evolution. The algorithm variants under consideration are introduced in Section 2 while the adopted methods of dealing with generated infeasible solutions, as well as the experimental setup, are introduced in Section 3. The results are discussed in Section 4 and conclusions are drawn in Section 5.

2. Differential evolution

Originally intended for a simple fitting problem [36,31], Differential Evolution (DE) has soon become an established meta-heuristic method for general-purpose real-valued optimisation, finding its place among other optimisation methods for real-world applications in engineering, robotics and other fields [35,30,41]. Besides the effectiveness of the DE optimisation framework, its success is attributed to the simplicity of its algorithmic structure. As can be seen from the pseudocode in Algorithm 1, it requires tuning only three parameters: the population size N (i.e., number of candidate solutions), the scaling factor F (i.e., a prefixed scalar multiplier in the range $(0, 2]$ involved in the mutation process) and the crossover rate C_r (i.e., a fixed probability value in [0,1] used to control the number of exchanged design variables between two candidate solutions).

Algorithm 1: Differential Evolution

Initialise $N \in \mathbb{N}^+, F \in (0, 2], C_r \in [0, 1]$ ▷ User defined
$g \leftarrow 1$ ▷ First generation
$\mathbf{P}_g \leftarrow$ uniformly draw N individuals in $\mathbf{D} \subset \mathbb{R}^n$ ▷ Initialise population $\mathbf{P}_g$
Compute $f(\mathbf{P}_g[i])\ \forall i = 1, \ldots, N$.
$\mathbf{x}_{\text{best}} \leftarrow$ best individual in $\mathbf{P}_g$
while condition on budget not met **do**
  for $i \leftarrow 1, 2, \ldots, N$ **do**
    $\mathbf{x}_m \leftarrow \text{Mutation}\,(\mathbf{P}_g, F)$ ▷ e.g. equations 1 to 4
    $\mathbf{x}_o \leftarrow \text{Crossover}\,(\mathbf{P}_g[i], \mathbf{x}_m, C_r)$ ▷ Algorithm 2 or 3
    $\mathbf{x}_o \leftarrow$Feasibility_Check$(\mathbf{x}_o, \mathbf{P}[i], \mathbf{D})$ ▷ Apply a strategy (equations 5 to 11)
    Compute $f(\mathbf{x}_o)$
    if $f\,(\mathbf{x}_o) \leqslant f\,(\mathbf{P}_g[i])$ **then** ▷ Fill the next population
      $\mathbf{P_{g+1}}[i] \leftarrow \mathbf{x}_o$
    else
      $\mathbf{P_{g+1}}[i] \leftarrow \mathbf{P}_g[i]$
    end if
  end for
  $g \leftarrow g + 1$ ▷ Replace the old with the new population
  $\mathbf{x}_{\text{best}} \leftarrow$ best individual in $\mathbf{P}_g$ ▷ Update best
end while
Output $\mathbf{x}_{\text{best}}$

In contrast to Evolutionary Algorithms (EA), DE is characterised by the following two significant differences:

- The parent and survivor selection mechanisms are simplified – both are replaced by the so-called '1-to-1 spawning' logic, typical of swarm intelligence algorithms, where one solution leads to the production of one other which might replace it.
- The mutation operator precedes the application of the crossover operator – hence, the latter might select some previously generated infeasible components and transfer them to the newly generated candidate solution, thus playing a role in its infeasibility.

Hence, during the g^{th} generation of DE, each individual $\mathbf{x}_t$ - referred to as 'target vector' in this context [31] - is selected one at a time from the population $\mathbf{P}_g$ (i.e. $\mathbf{x}_t = \mathbf{P}_g[i]$ with $i = 1, 2, 3, \ldots, N$) to be mated with a previously prepared 'mutant vector' $\mathbf{x_m}$ and produce a new 'offspring' solution $\mathbf{x_o}$. The latter, gets the place of its parent $\mathbf{x}_t$ in the next population only if it displays a better objective function value, thus following a greedy logic. This is not common in EAs, where more than one individual (usually two, but multi-parent crossovers also exist [28]) must be selected to generate at least one offspring vector.

To fully describe a DE algorithm, the notation DE/x/y/z is typically used, where z indicates the abbreviation of the crossover operator name, while x/y defines, respectively, how a candidate solution is selected for mutation and on how many differences between individuals randomly taken from the population (referred to as 'difference vectors') it is based. Different mutation operator require a different number of individuals randomly picked from the population to function. In established mutations [18], such number goes from a minimum of 2 individuals, when only 1 difference vector is employed (i.e. $y = 1$), to a maximum of 5, when one individual plus two difference vectors are employed (i.e. $y = 2$). We indicate these distinct

individuals, i.e. the same solution cannot be selected twice to avoid a having a null difference vector, with the notation $\mathbf{x}_{r_j}$, where the index r_j refer to the j^{th} selected solution with $j \in \{1,2,3,4,5\}$. Amongst the most popular x/y combinations the following are worth mentioning:

- rand/1:

$$\mathbf{x}_m = \mathbf{x}_{r_1} + F(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \tag{1}$$

- rand/2:

$$\mathbf{x}_m = \mathbf{x}_{r_1} + F(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) + F(\mathbf{x}_{r_4} - \mathbf{x}_{r_5}) \tag{2}$$

- best/1:

$$\mathbf{x}_m = \mathbf{x}_{\text{best}} + F(\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \tag{3}$$

- current-to-best/1:

$$\mathbf{x}_m = \mathbf{x}_t + F(\mathbf{x}_{\text{best}} - \mathbf{x}) + F(\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \tag{4}$$

However, many more variants exist in the literature [33,18,16,32,37]. These include also more advanced self-adaptive DE algorithms, where mechanisms for adjusting the control parameters [3,34] and the population size [4,38] on-the-fly have been embedded within the classic DE framework. However, the latter are not in the focus of this study, as we are interested in discovering relations between parameters setting and corresponding algorithmic behaviour. Moreover, given that more complex DE algorithms make use of multiple classic variation operators (or their minor modifications) and complex mechanisms for their coordination, we believe that it is beneficial to use 'bottom up' analysis and first put a stronger emphases on established mutation and crossover operators.

It must be mentioned that what is referred to as 'mutation' in the DE framework, i.e., a linear combination of individuals, is called 'arithmetic crossover' in the Genetic Algorithm (GA) [19]. Therefore, the whole variety of crossover methods for real-valued GAs has not 'migrated' to the DE world. Meanwhile, in DE, crossover is only meant for exchanging design variable between solutions and only two consolidated strategies, namely the binomial crossover (indicated with `bin`) and the exponential crossover (indicated with `exp`), are commonly used. Their descriptions are given in Algorithms 1 and 2, respectively.

Algorithm 2: Binomial crossover

Input two parents $\mathbf{x_1}$ and $\mathbf{x_2}$ ▷ $\mathbf{x_1}, \mathbf{x_2} \in \mathbf{D} \subset \mathbb{R}^n$
A random index $\mathcal{I}$ is uniformly drawn in $[1, n] \subset \mathbb{N}$
for $i \leftarrow 1, 2, 3, \ldots, n$ **do**
  A random value $\mathcal{U}$ is uniformly drawn in $[0, 1] \subset \mathbb{R}$
  if $\mathcal{U} \leqslant C_r$ or $i = \mathcal{I}$ **then** ▷ $C_r \in [0, 1]$ is user defined
    $\mathbf{x_1}[i] \leftarrow \mathbf{x_2}[i]$ ▷ Exchange the i^{th} component
  end if
end for
Output $\mathbf{x_1}$

Algorithm 3: Exponential crossover

Input two parents $\mathbf{x_1}$ and $\mathbf{x_2}$ ▷ $\mathbf{x_1}, \mathbf{x_2} \in \mathbf{D} \subset \mathbb{R}^n$
A random index $\mathcal{I}$ is uniformly drawn in $[1, n] \subset \mathbb{N}$
$i \leftarrow \mathcal{I}$
do
  $\mathbf{x_1}[i] \leftarrow \mathbf{x_2}[i]$ ▷ exchange the i^{th} component
  $i \leftarrow (i \mod n) + 1$
  A random value $\mathcal{U}$ is uniformly drawn in $[0, 1] \subset \mathbb{R}$
while $\mathcal{U} \leqslant$ Cr and $i \neq \mathcal{I}$ ▷ $C_r \in [0, 1]$ is user defined
Output $\mathbf{x_1}$

2.1. DE parameters

The behaviour of all DE configurations is known to depend strongly on its control parameters N, F and C_r [44,27,45,12]. However, their individual contribution is clear as summarised in Section 2.1.1. Experimental setup of this paper is presented in Section 2.1.2.

2.1.1. Meaning of parameters

Control parameters of DE have a clear role if considered individually:

- [-] The N parameter defines the number of individuals in the population (i.e., population size). Intuitively, a large population size means a high diversity and therefore a better exploration of the search space. This is partially confirmed in terms of population diversity [45] and convergence rate [40], but does not necessarily result in a better performance. Indeed, in large-scale domains, 'micro'-populations of about 5 individuals have proven to be effective and not to suffer from premature convergence [14]. Furthermore, in [4] a DE variant reducing its population size during the optimisation process is proposed to reduce stagnation. It must be highlighted that too high values of N are also to be avoided as they are impractical for the large-scale problems and deleterious for the convergence process, at the expense of an adequate exploitation phase to refine promising solutions locally.
- [-] The F parameter is introduced to control the length of the *difference vectors* in the mutation process, which is responsible for moving/perturbing a candidate solution. Despite its original conception as a scaling factor, nowadays it commonly assumes values smaller than 1, thus shrinking the exploitative step of the mutation operator. Ideally, this is supposed to shrink the exploratory radius to generate feasible solutions: higher values of F lead to a longer step taken within the domain. However, we argue this a simplistic vision of the role played by F as the perturbation of a candidate solution also depends on the magnitude of the difference vector, since it a dynamic quantity which is expected to become very small as the algorithm converges [40].
- [-] The C_r parameter controls the number of design variables inherited from the mutant by fixing a probability to be used as a threshold (i.e., $C_r \in [0, 1]$).

However, there are studies pointing out a correlation amongst parameters [43,45] which suggest that when practitioners tune them, with the aim of improving performance on a specific real-world scenario [39], they should consider the effect of their mutual interaction rather than thinking of N, F and C_r as three independent factors. This is particularly true for the control parameters F and C_r [27], but also methods for adjusting N can make the difference [26,6,4,5].

2.1.2. Parameter values used in this study

All DE configurations mentioned in the previous sections are considered in this study with the following values of parameters:

- [-] population size $N \in \{5,20,100\}$;
- [-] scaling factor $F \in \{0.05, 0.266, 0.483, 0.7, 0.916, 1.133, 1.350, 1.566, 1.783, 2.0\}$, i.e. uniformly spaced in $(0, 2]$. Thus, *originally* suggested range [24] of values of F is tabulated here.
- [-] crossover rate $C_r \in \{0.05, 0.285, 0.52, 0.755, 0.99\}$, i.e., uniformly spaced in $[0, 1]$.

3. Outside the box

In this section, we introduce the considered approaches for dealing with infeasible solutions produced during the run (Section 3.1), discuss the proposed approach and its relevance (Sections 3.2 and 3.3), our approach towards measurement and visualisation of the effect (Section 3.4), and the experimental setup (Section 3.5).

3.1. How to deal with infeasible solutions

It is virtually inevitable that throughout a run of a heuristic optimisation algorithms, some generated solutions are infeasible. Thus, *a well-designed optimisation algorithm* should specify what should be done with such solutions. The literature reports on numerous strategies of how to deal with infeasible solutions for metaheuristic optimisation [15,2]. For DE, the most used strategies are based on average, re-initialisation or re-sampling methods [45], penalty functions [27,12] and others as e.g. saturation and toroidal transformations [12]. It is worth to point out that, as a consequence of the 1-to-1 spawning replacement logic, *penalty* strategies lead to always discard the penalised individual in favour of keeping the target vector [12]. In this light, penalty-based strategies were not included in our experimental setup; instead we consider only a '*dismiss*' strategy which produces identical results. The latter is the only strategy employed in this study that replaces all components of an infeasible solution, i.e. both infeasible and feasible ones, whereas with the remaining strategies under consideration *only infeasible dimensions* get modified as feasible ones are neutral elements of such operators.

More formally, assuming $\mathbf{x}[i]$ is the component of the newly generated solution in the i^{th} dimension, a_i and b_i are the corresponding lower and upper boundaries of the domain in the i^{th} dimension and $\mathbf{x}_f[i]$ is the new feasible value of the component, the employed strategies of dealing with infeasible solutions are formally described in the following sections. All of them implement the 'Feasibility_Check' method of Algorithm 1.

3.1.1. Saturation strategy

This strategy places those design variables exceeding their corresponding boundaries to the closest amongst their upper and the lower bound [12,10]. Mathematically, for each $\mathbf{x}[i]$ $(i = 1, 2, 3, \ldots, n)$, this operator applies the mapping $s : \mathbb{R} \to [a, b]$ defined as

$$s(\mathbf{x}[i]) = \begin{cases} a_i & \text{if } \mathbf{x}[i] < a_i \\ b_i & \text{if } \mathbf{x}[i] > b_i \\ \mathbf{x}[i] & \text{otherwise} \end{cases} \tag{5}$$

so returning feasible solutions by simply looping $\mathbf{x}_f[i] = s(\mathbf{x}[i])$ across each component. Fig. 1(a) graphically depicts this linear transformation. Variants of algorithms with this strategy are marked as 'sat' in Figs. 5–8.

3.1.2. Toroidal strategy

This strategy consists in reflecting only those values of coordinates that are outside the domain off the opposite domain boundary inwards – as if the boundaries are connected and the domain forms a ring [12,10]. In a Cartesian system, this transformation is depicted with the graph in Fig. 1(b). Operationally, this strategy implements the assignment $\mathbf{x}_f[i] = t(\mathbf{x}[i])$ $(\forall i = 1, 2, 3, \ldots, N)$ where the analytical expression of the function $t : \mathbb{R} \to [a, b]$ is expressed as follows

$$t(\mathbf{x}[i]) = \begin{cases} a_i + \left(1 - \left|\frac{\mathbf{x}[i]-a_i}{b_i-a_i} - \left\lfloor \frac{\mathbf{x}[i]-a_i}{b_i-a_i} \right\rfloor\right|\right) \cdot (b_i - a_i) & \text{if } \mathbf{x}[i] < a_i \\ a_i + \left(\frac{\mathbf{x}[i]-a_i}{b_i-a_i} - \left\lfloor \frac{\mathbf{x}[i]-a_i}{b_i-a_i} \right\rfloor\right) \cdot (b_i - a_i) & \text{if } \mathbf{x}[i] > b_i \\ \mathbf{x}[i] & \text{otherwise} \end{cases} \tag{6}$$

in which $\lfloor \ldots \rfloor$ represents the *floor* operator. Variants of algorithms with this strategy are marked as 'tor' in Figs. 5–8.

3.1.3. Mirror correction strategy

This strategy moves only those values of coordinates that are outside the domain by reflecting the infeasible value off the closest boundary inwards the domain [12] - as shown in Fig. 1(c). Let $r : \mathbb{R} \to \mathbb{R}$ be the 'reflection' defined as

$$r(\mathbf{x}[i]) = \begin{cases} a_i + (a_i - \mathbf{x}[i]) & \text{if } \mathbf{x}[i] < a_i \\ b_i - (\mathbf{x}[i] - b_i) & \text{if } \mathbf{x}[i] > b_i \\ \mathbf{x}[i] & \text{otherwise} \end{cases}, \tag{7}$$

then the mirroring operator can be written in the form of a recursive function $m : \mathbb{R} \to [a_i, b_i]$ defined as

$$m(\mathbf{x}[i]) = \begin{cases} \mathbf{x}[i] & \text{if } a_i \leqslant \mathbf{x}[i] \leqslant b_i \\ m(r(\mathbf{x}[i])) & \text{otherwise} \end{cases}. \tag{8}$$

If this strategy is employed, the 'Feasibility_Check' method of Algorithm 1 is simply implemented with a loop performing the assignment $\mathbf{x}_f[i] = m(\mathbf{x}[i]), \forall i \in \{1, 2, 3, \ldots, n\}$. Variants of algorithms with this strategy are marked as 'mirr' in Figs. 5–8.

3.1.4. Complete One-tailed normal correction strategy (COTN)

This is a probabilistic strategy that iteratively re-samples (until the point gets inside the domain) infeasible dimensions 'nearby' the violated bound by means of a one-tailed normal distribution with purposely small standard deviation, i.e. $\left|\mathcal{N}(0, \frac{b_i-a_i}{3})\right|$ [12], as formally indicated below

$$n(\mathbf{x}[i]) = \begin{cases} a_i + \left|\mathcal{N}(0, \frac{b_i-a_i}{3})\right| & \text{if } \mathbf{x}[i] < a_i \\ b_i - \left|\mathcal{N}(0, \frac{b_i-a_i}{3})\right| & \text{if } \mathbf{x}[i] > b_i \\ \mathbf{x}[i] & \text{otherwise} \end{cases}. \tag{9}$$

Normally distributed values are to be resampled independently for each $\mathbf{x}[i]$, their densities are shown in Fig. 1(d). Hence, COTN is mathematically formulated as a recursive function $c : \mathbb{R} \to [a_i, b_i]$ defined as

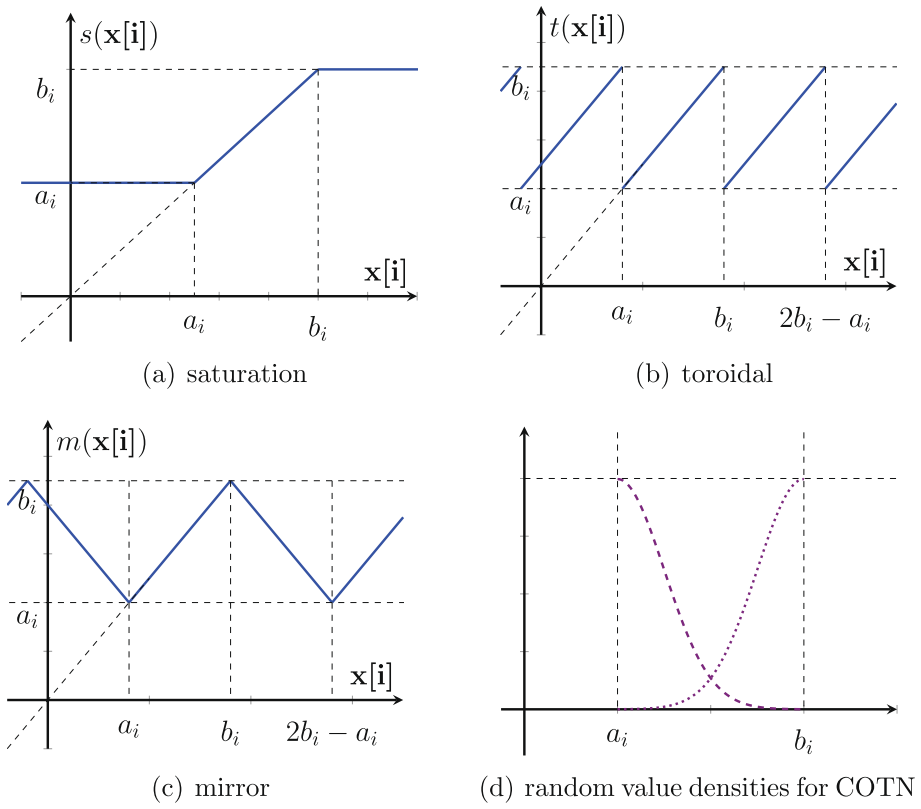


Fig. 1. A graphical representation of the strategies as functions of $\mathbf{x}[\mathbf{i}]$, assuming $0 < a_i < b_i$.

$$c(\mathbf{x}[i]) = \begin{cases} \mathbf{x}[i] & \text{if} a_i \leqslant \mathbf{x}[i] \leqslant b_i \\ c(n(\mathbf{x}[i])) & \text{otherwise} \end{cases}. \tag{10}$$

If this strategy is employed, the 'Feasibility_Check' method of Algorithm 1 is simply implemented with a loop performing the assignment $\mathbf{x}_f[i] = c(\mathbf{x}[i]), \forall i \in \{1, 2, 3, \ldots, n\}$. Variants of algorithms with this strategy are marked as 'COTN' in Figs. 5–8.

3.1.5. Dismiss strategy

As the name suggests, this is a 'greedy' strategy which scans through each i^{th} component of a solution $\mathbf{x}$ and, as soon as one infeasible $\mathbf{x}[i]$ is detected, discards $\mathbf{x}$ to the previous feasible position [12].

In DE, this means that an infeasible offspring $\mathbf{x}_o$ gets replaced by its target vector $\mathbf{x}_t$. This is fully described with $\mathbf{x}_f = d(\mathbf{x})$ where $d : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$d(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if } \mathbf{x} \in \mathbf{D} \\ \mathbf{x}_t & \text{if } \mathbf{x} \notin \mathbf{D} \end{cases}. \tag{11}$$

As mentioned above, this is the only strategy modifies all components of an infeasible solution and not just its infeasible components. Variants of algorithms with this strategy are marked as 'dism' in Figs. 5–8.

3.2. Keeping track of the number of infeasible solutions

As discussed in the previous section, ISs almost inevitably get generated during the runs of heuristic optimisation methods. Generating *excessive* number of such solutions is clearly undesirable as it wastes computational budget. Moreover, generating IS is an indication that algorithm's operators do not fully adapt to the optimisation problem. Furthermore, a high number of ISs potentially disrupts the search: depending on how such solutions are treated, the search might be e.g. misled by artificial function values in the case of penalty-based strategies or by changing the direction of search, for mirror and COTN strategies, or slowed down, as in the case of saturation strategy. Heuristically, toroidal strategy appears to be the most advantageous as it keeps the direction of search and does not slow down the search. However, clearly, it is not suitable for all types of problems.

At the same time, not generating *any* infeasible solutions is 'suspicious': is the algorithm exploring the areas close to the boundaries to a *sufficient degree*? Generating a relatively *low* number of such solutions potentially allows the algorithm to 'learn' the boundaries. Preferably, this should happen without over-exploring this area.

Another aspect that 'comes into play' here is problem's dimensionality. Trivially, a solution is infeasible if it has a coordinate in *at least* one dimension that is outside of its respective boundaries. Thus, the number of infeasible solutions is expected to grow exponentially with the dimensionality of the problem.

Simplifying the real situation, let n be the problem's dimensionality and $p \in [0, 1]$ – the probability with which solutions become infeasible *in exactly one* dimension (constant in time and across the domain). Then, ignoring dependencies between the dimensions, $f(p, n) = 1 - (1 - p)^n$ is the probability that a solution is infeasible in at least one dimension. To help visualise this expression Fig. 2 shows intervals (the vertical axis) for which $f(p, n)$ stays below the values shown in the horizontal axis. Clearly, only extremely low values of p lead to relatively low values of $f(p, n)$ – in dimensionality 500, to have the probability of generating an IS below 0.01, the probability of becoming infeasible in one dimension has to be below 0.0000201.

This figure serves as a justification for the following: the *choice* of algorithm's strategy of dealing with generated ISs is *of paramount importance for highly multidimensional problems* as there it is *extremely easy* to generate an IS. The latter observation also suggests that poor scalability of some general-purpose algorithms might be related to excessive production of ISs which, if not dealt with in an informed way, may mislead the search in such vast search spaces [13].

This, however, does not mean that in lower-dimensional cases ISs can be neglected without expecting deleterious consequences on the behaviour of the algorithm. To formally quantify their occurrence and the corresponding effect, let us assume any algorithm used in this study is a black box. The only observable information is the number of solutions the algorithm has generated outside the domain throughout a run with fixed budget – in this paper, such budget is expressed by the number of fitness evaluations, $10^4 n$ where $n = 30$ is the dimensionality of the problem. A fundamental *research question* then is, whether such limited information is sufficient to make any meaningful conclusions regarding the overall behaviour of the algorithm.

To answer this question, consider results on some objective function for two different algorithms: one always generating solutions outside the domain and one always generating an extremely low number of infeasible solutions. Two explanations for this are possible:

[-] This happens *due to the landscape*: due to particular 'features' of the landscape of the objective function, e.g., presence of good solutions close to the boundaries of domain, solutions tend to be generated outside the domain, i.e., to be infeasible;

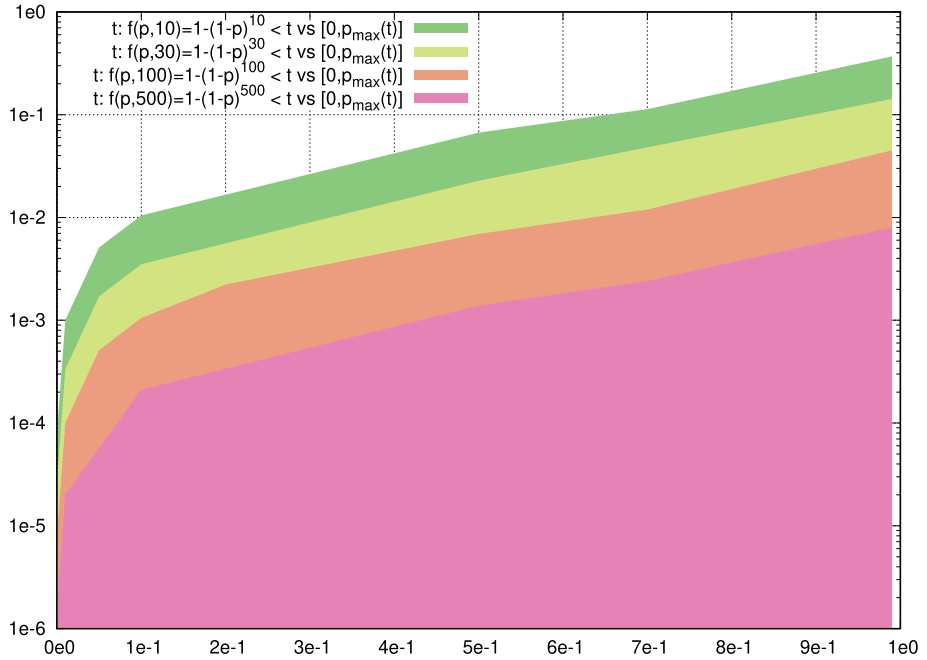


Fig. 2. If $p \in [0, 1]$ is the probability for a solution to become infeasible in exactly one dimension and n is the problem dimensionality, then the probability that solution is infeasible in at least one dimension is, trivially, a function of p and n, $f(p, n) = 1 - (1 - p)^n$. Values shown in this figure come from solving for p the inequality $1 - (1 - p)^n \leqslant t$ for given n. Values of t are shown on the horizontal axis and values of p_{max} obtained from solving the inequality are shown on the vertical axis. Shaded areas represent values of $p \in [0, p_{max}]$ for which the inequality holds, for four values of n. This figure is based on [12].

[-] This happens *due to the algorithm*: due to particular 'moves' prevalent in the algorithm, e.g., a too aggressive generating operator, newly generated solutions tend to be infeasible.

Clearly, in case of a general objective function, both aspects above are present. But what if one could use an objective function where *absence of correlations* between the 'geographical' location of the solution and the corresponding functional value *is guaranteed*? If results of the experiment above are replicated on such 'special' function, the only possible explanation of the phenomenon above is the *algorithm itself*.

3.3. State of the art in using f_0 for benchmarking

A fully stochastic objective function

$$f_0 : [0,1]^{30} \to [0,1], \forall x f_0(x) \sim \mathcal{U}(0,1) \tag{12}$$

has been recently investigated for a different purpose – searching for the so-called 'structural bias' of algorithms [23,10,11]. The randomness in assigning objective function values to the points allows decoupling interaction between the objective function and the algorithm.

The use of function f_0 in [12,22,21] as a test for identifying deficiencies in algorithms according to *alternative* performance measure highlights the assertion that *algorithmic design is in fact a multiobjective problem* – good algorithms are not only supposed to find good points but also find them *regardless* of their position in the domain. Clearly, different objectives in designing algorithms can be *conflicting* and it is important not to confuse them. Testing with f_0 is not intended as the *only* performance test. Such tests constitute only a partial, yet important characterisation of the algorithm.

The validity of the approach described above is further confirmed by results of this paper obtained on such experimental setup with f_0– outlined in Section 3.5. Here, we study proportions of infeasible solutions (POIS) generated in a series of independent budgeted runs, for a wide selection of DE configurations running on f_0. At the end of every run, the number of generated infeasible solutions is divided by the total fitness evaluation budget, thus giving a POIS. For each algorithm configuration, an empirical distribution of POIS (EDPOIS) is considered over a series of independent runs.

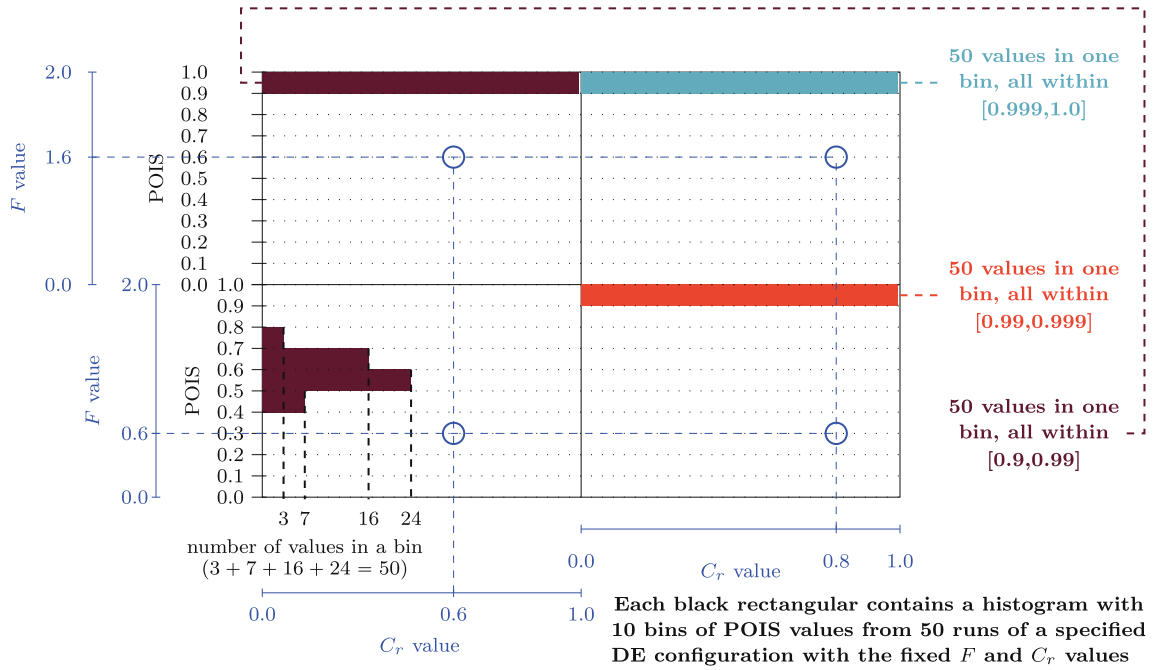


Fig. 3. Graphical explanation of bins, axes ranges and captions omitted for readability from Figs. 5–8. Details relevant for different layers of information discussed in Section 3.4.1 are shown here in their corresponding colours: *layer one* – lines and labels in black, *layer two* – lines and labels in blue, *layer three* – lines, labels and bins in teal, orange or violet. Axes on the left and below apply to all 4 small subfigures with histograms. Bottom left histogram shows a case where 50 POIS values fall in *different* bins - number of points in each bin is shown in black on the horizontal axis below. Other 3 histograms show cases where all 50 POIS values fall in one top $(0.9, 1.0]$ bin only. For such cases, the colour of the bins allows distinguishing cases where only extreme parts of this $(0.9, 1.0]$ bin are taken – see explanation in the figure on the right in teal, orange and violet. Histograms where all POIS values fall in the *bottom* $[0, 0.1]$ bin only are *not shown* here – explanation for them is identical to the explanation for to top left, top right and bottom right cases, where histograms are made up of one top bin only. See further explanation about layers one and two in Fig. 4.

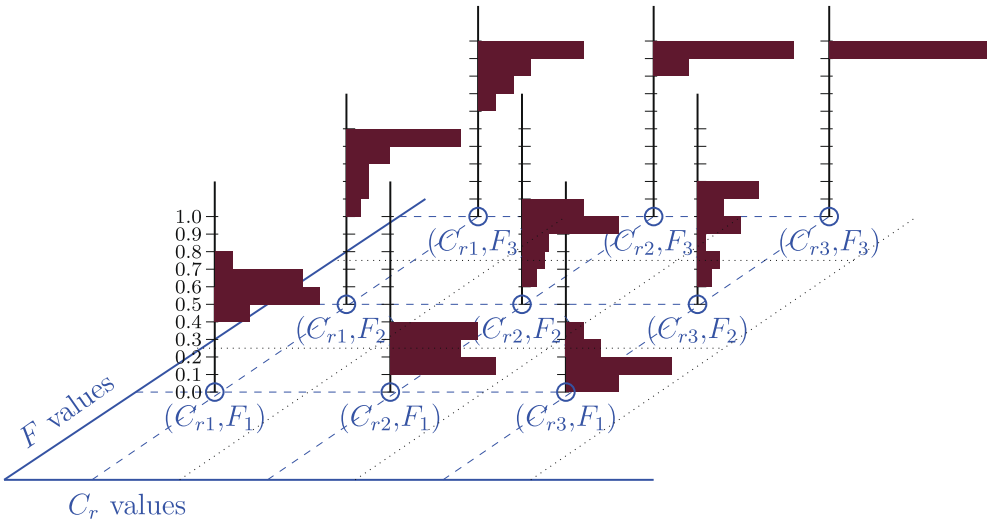


Fig. 4. Decoupling information layers one and two from Figs. 3,4,5,3,4,5,6,7,8. This *three-dimensional sketch* shown here in perspective illustrates the fact that Figs. 5–8 should be considered as an *attempt to draw a three-dimensional figure with projections in two dimensions*: individual histograms should be placed on the page in the points marked by the blue circles in a perpendicular fashion. In this sketch, (C_r, F) values and histogram bars do not represent real values and are not shown to scale.

3.4. Empirical distributions of POIS

To the best of our knowledge, empirical distributions of proportions of infeasible solutions have *never been studied in this field*. However, they provide a lot of information, as shown in the subsequent sections. Thus, here, we consider different combinations of DE control parameters and study the resulting EDPOISs.

3.4.1. How to read EDPOIS Figs. 5–8

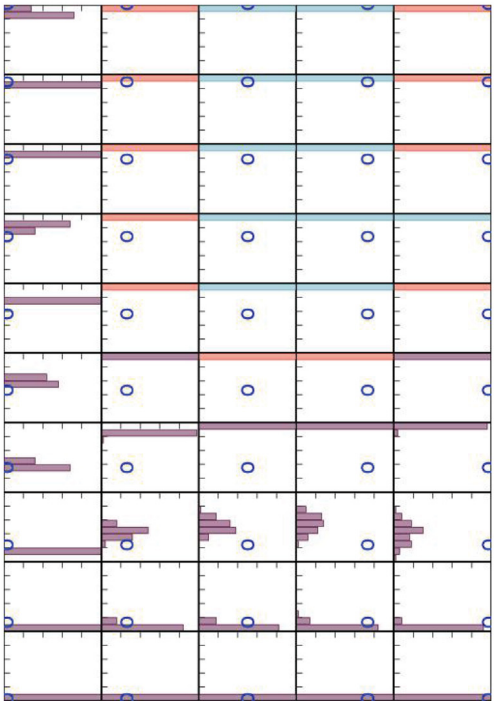
The setup outlined in the previous sections results in the need for rather complicated figures that require detailed explanation.

In Figs. 5–8, each DE configuration is shown in a separate subfigure, for each of the three values of population size considered – see configuration name and N in captions of subfigures. Furthermore, each configuration is considered for a selection of the parameters F and C_r – with values listed in Section 2.1.2. Thus, EDPOIS for each pair of (F, C_r)-values are shown in a small subfigure (small rectangles, each containing one histogram) within the larger subfigures per DE configurations. Each of these small subfigures is associated with the (F, C_r) pair – small subfigures are in fact placed 'on the grid' of (F, C_r) values and neighbouring small subfigures have close (F, C_r) values. Thus, empirical distributions of proportions of infeasible solutions shown in Figs. 5–8 *should be read as follows.* Each smaller subfigure carries three layers of information regarding *one* DE configuration considered with one pair of values of (F, C_r):

1. The *first layer* of information in small subfigures, made up of horizontal bars or bins (in other words, this layer is made up of a simple histogram with 10 bins), represents the empirical distribution of proportions of infeasible solutions in a series of runs of this configuration for a pair of (F, C_r)-values[1]. In these smaller subfigures, proportions of ISs accumulated by the end of each run in a series of runs are shown on the y-axis; this axis has a fixed range of $[0, 1]$ and points upwards. While the number of runs is reported on the x-axis; this axis points to the right and has a range of $[0, m]$, where m is the number of runs in a series, $m = 50$ here. Thus, the origin of the axes for this layer lies in the lower left corner. Additional explanation of this layer can be found in Fig. 4.
2. The *second layer* of information in small subfigures, shown in blue, indicates values of parameters F and C_r that have been used for this particular histogram. Also for this layer, the y-axis points upwards and the x-axis to the right, but they report values for F and C_r, respectively. Following Section 2.1.1, the ranges for F and C_r are fixed at $(0, 2]$ and $[0, 1]$, respectively. Thus, also for this layer, the origin lies in the lower left corner. Additional explanation of this layer can be found in Fig. 4.
3. The *third layer* of information in small subfigures is shown with the colour of histogram bins. Varying bins colour allows distinguishing distributions with *exclusively 'extreme' values* of POIS that cannot be otherwise distinguished from the case when *all* POIS values fall in one bottom or top bin only:
   [-] Bins marked in teal identify distributions where POIS values from *all* runs in a series fall exclusively within a range of $[0, 0.001]$ or $[0.999, 1.0]$, depending on whether only a bottom or top bin, respectively, is present.

[1] Meaning of colours of the bars in the histogram is explained in point 3 of this list.

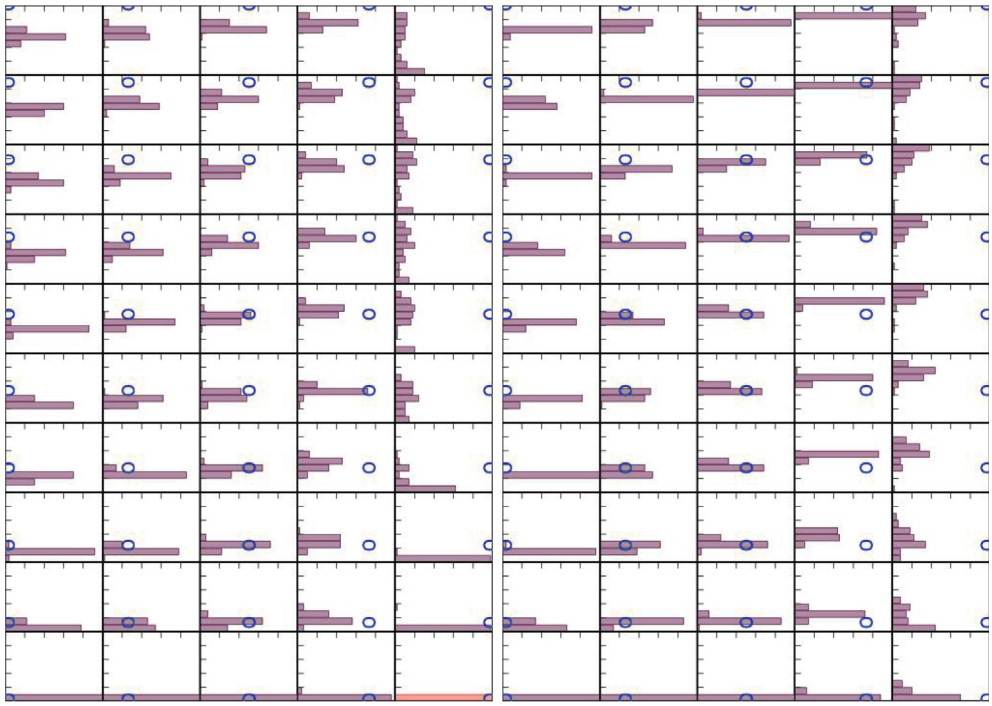
(a) DE/curr-to-best/1/bin sat, N=5 (b) DE/curr-to-best/1/bin sat, N=20



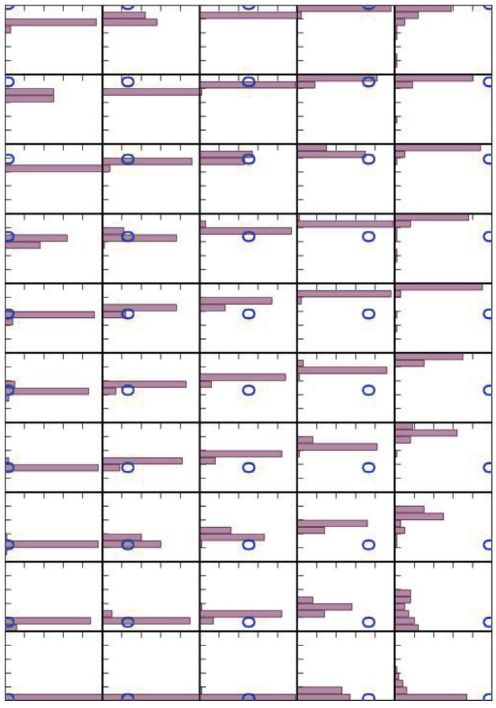
(c) DE/curr-to-best/1/bin sat, N=100

Fig. 5. EDPOIS generated in a series of runs. Figure is explained in Section 3.4.1.

[-] Bins marked in orange identify distributions where *all* runs in a series have resulted in POIS values only within a range of $[0.001, 0.01]$ or $[0.99, 0.999]$, depending on whether only a bottom or top bin, respectively, is present.



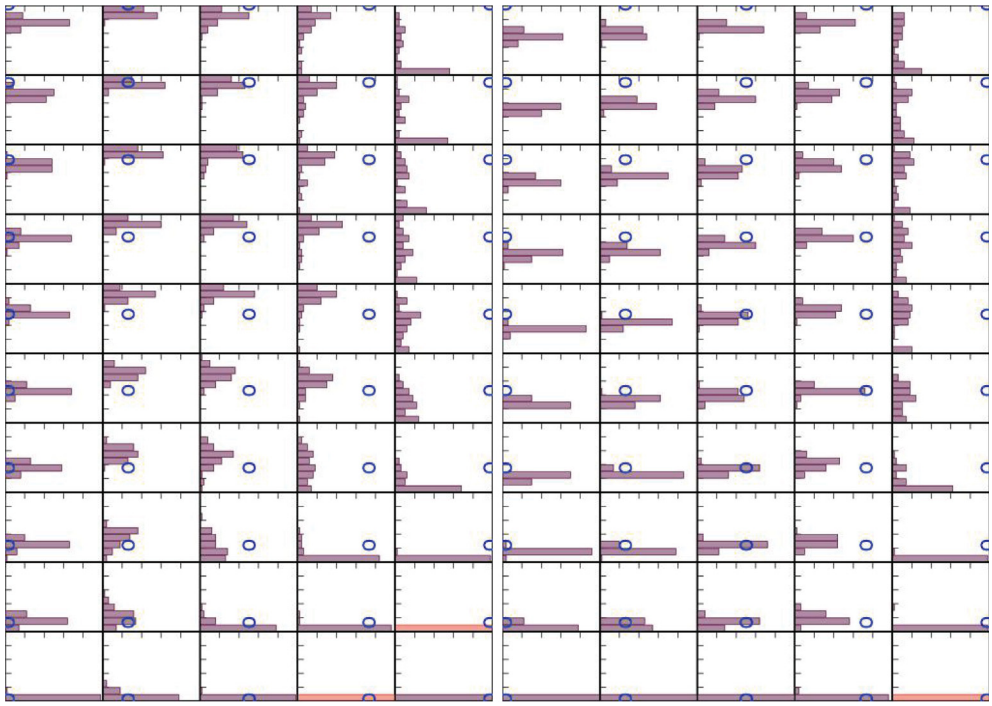
(a) DE/best/1/exp sat, N=5 (b) DE/rand/1/exp sat, N=5



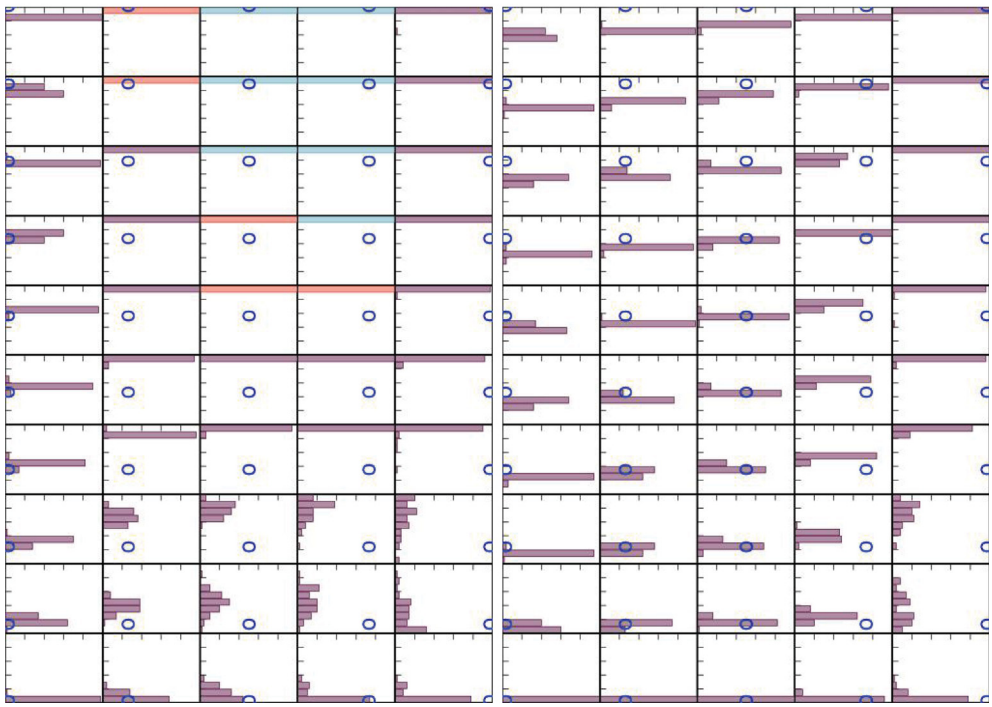
(c) DE/rand/2/exp sat, N=5

Fig. 6. EDPOIS generated in a series of runs. Figure is explained in Section 3.4.1.

[-] Bins marked in violet indicate all remaining distributions, including normal cases when POIS values fall in one or more histogram bins and a case when all POIS values fall in one bin but have values exclusively within $[0.01, 0.1]$ or $[0.9, 0.99]$, depending on whether only a bottom or top bin, respectively, is present.

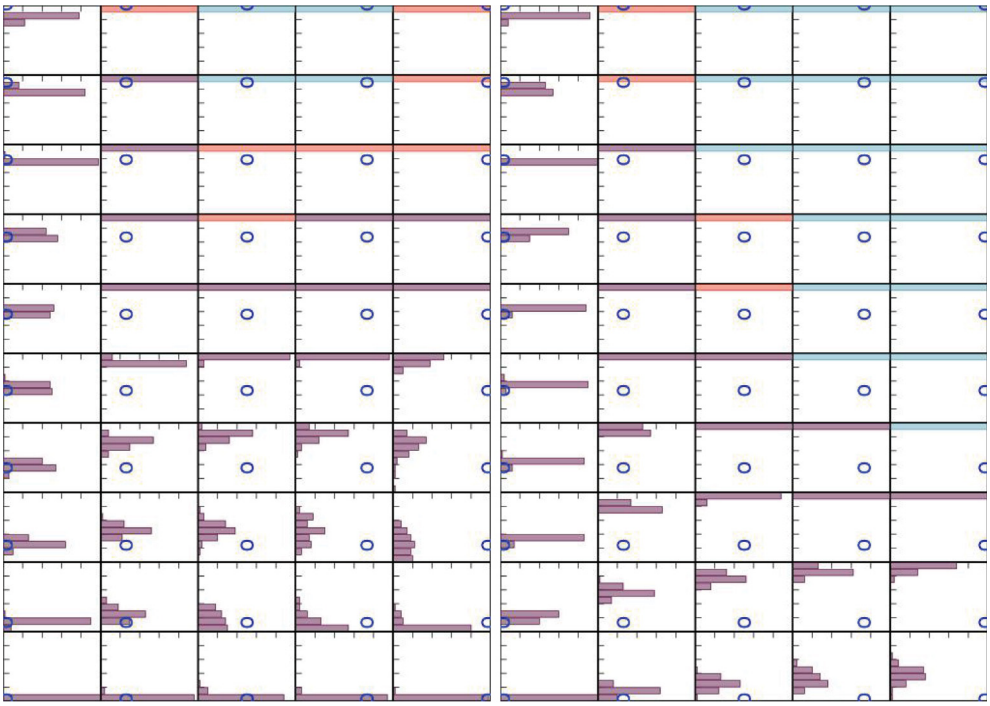


(a) DE/best/1/bin sat, N=5 (b) DE/best/1/exp sat, N=5

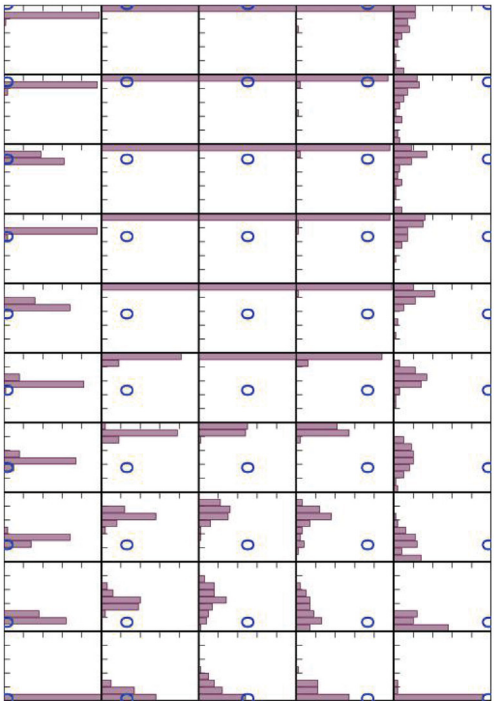


(c) DE/rand/1/bin tor, N=5 (d) DE/rand/1/exp tor, N=5

Fig. 7. EDPOIS generated in a series of runs. Figure is explained in Section 3.4.1.



(a) DE/rand/1/exp COTN, N=5 (b) DE/rand/1/exp mirr, N=5



(c) DE/rand/1/exp sat, N=5

Fig. 8. EDPOIS generated in a series of runs. Figure is explained in Section 3.4.1.

Axes ranges and captions are kept fixed and, thus, omitted from Figs. 5–8 for readability; all these ranges and captions, together with the summary of information per layer can be found in Fig. 3. To sum up, Figs. 5–8 should be considered as an *attempt to draw a three-dimensional figure with projections in two dimensions*: distributions shown in teal/orange/violet in each small subfigure should be placed on the page in the points marked by the blue circles in a perpendicular fashion, towards the reader – see Fig. 4.

3.5. Experimental setup

Results for this study have been produced by the SOS software platform [9] whose source code is freely available in [8] for reproducibility. Experimental setup based on the direct product of options for the following DE operators/parameters has been considered in this paper:

[-] Crossover operator: binomial, exponential.
[-] Mutation operator: DE/best/1, DE/current-to-best/1, DE/rand/1, DE/rand/2.
[-] Operator defining strategies of dealing with IS: COTN, dismiss, mirror, saturation, toroidal.
[-] Settings for parameters N, F and C_r as discussed in Section 2.1.2.

Each of the resulting 6000 configurations[2] has been run 50 times *minimising* the fully random objective function $f_0 : [0,1]^{30} \to [0,1]$ where $\forall x\, f_0(x) \sim \mathcal{U}(0,1)$. Each run has been allocated a budget of $3 \cdot 10^5$ fitness evaluations, following the general practice in the field of $10000 \times n$ where n is problem dimensionality [25,17].

4. Discussion of results

To keep the length of this manuscript reasonable, only few results are shown in this section in Figs. 5–8; results for all 120[3] considered DE configurations can be found in [11]. Comparisons can be made according to several aspects.

4.1. Comparison of EDPOIS

In this section, the results are discussed with respect to the observed impact of the single factors population size (Section 4.1.1), mutation and crossover variants (Sections 4.1.2 and 4.1.3), and strategy for handling infeasible solutions (Section 4.1.4).

4.1.1. Comparison across population sizes

DE configurations considered in this study clearly exhibit a different behaviour in terms of POIS depending on the population size. Shapes of EDPOIS from the smallest population size $N = 5$ considered are the most diverse across all configurations. When comparing $N = 5$ to $N = 20$, all DE configurations consistently show an increase in POIS. Population size of $N = 20$, which is typically considered small in DE [29,7], leads to the large portion of configurations producing 100% infeasible solutions in all runs in a series for only slightly more aggressive values of control parameters. EDPOIS for $N = 20$ results in smaller variance compared to $N = 5$. Moving from $N = 20$ to $N = 100$, even more configurations result in 100% infeasible solutions – barely any configurations have not generated POIS different from 0% or 100%. Fig. 5 shows typical EDPOIS when moving from $N = 5$ to $N = 20$ and $N = 100$ in the same DE configuration. It should be mentioned that there is a small number of configurations whose EDPOIS do not exhibit any noticeable change when comparing $N = 5$ to $N = 20$ and $N = 100$ (e.g., `DE/current-to-best/exp dismiss`).

Thus, the choice of population size does not appear to be the only factor in describing the variability among EDPOIS.

4.1.2. Comparison across mutation variants

Fig. 6 shows typical differences in EDPOIS for configurations identical in everything but their mutation operator. It is clear that a change in mutation operator leads to very *minor differences* in POIS when all other parameters are kept unchanged. More specifically, such change does not influence the general trend in POIS but rather changes the standard deviation of EDPOIS; much stronger differences are observed for the maximum value of C_r. Such behaviour is replicated across all configurations also when crossover is factored in.

Thus, the choice of mutation can be excluded from the factors describing the variability among EDPOIS.

4.1.3. Comparison across crossover variants

When comparing EDPOIS for groups of configurations identical in everything but their crossover operator, it becomes evident that these are two different groups of algorithms. Despite the fact that DE crossover operators are by design unable to produce unfeasible solutions from two feasible inputs, the cascade process resulting from the use of a mutation operator, which is responsible for generating ISs, followed by `bin` or `exp` crossover algorithms can lead to significantly different POIS.

[2] 2 crossovers × 4 mutations × 5 strategies × 5 C_r settings × 10 F settings × 3 N settings = 6000
[3] 2 crossovers × 4 mutations × 5 strategies × 3 N settings = 120

Generally, with the proposed experimental setup, exp crossover variant seems to result in smaller POIS. Fig. 7 shows two typical examples of differences in EDPOIS induced by choice of crossover operator.

Thus, the choice of crossover does not appear to be the only factor in describing the variability among EDPOIS. Further discussions regarding differences in crossover variants follow in Section 4.2.3.

4.1.4. Comparison across strategies

The general pattern appears to be the same for all of the strategies for handling infeasible solutions (COTN, dismiss, mirror, saturation, toroidal): for *exponential* crossover, range of generated POIS narrows down, meanwhile POIS values increase both with increasing values of F and C_r, while this trend depends mainly on F for *binomial crossover*.

In both cases, for population sizes 20 and 100, F values of 0.7 and higher rapidly shift the EDPOIS towards the $[0.9, 1.0]$-range, either independently on C_r for *binomial crossover* or as C_r approaches its maximum for *exponential crossover*. Such behaviour is observed independently of the feasibility handling strategy, but for COTN and saturation it is slightly less pronounced as F (and C_r) are increasing. For $N = 5$ and higher values of C_r, the saturation strategy generally shows a wider spread of the EDPOIS, especially when $C_r = 0.99$. The mirroring, toroidal and dismiss strategies generally show an indistinguishable behaviour in terms of EDPOIS. Overall, we can conclude that COTN, saturation and mirroring, toroidal, dismiss form *two different groups*, with the first one resulting in a slightly slower shift of the EDPOIS towards the $[0.9, 1.0]$-range as F and C_r increase. Fig. 8 shows configurations for $N = 5$, exponential crossover and COTN, mirroring, and saturation. For $N = 5$, the effects are mostly depending on F, less on C_r, except for saturation with $C_r = 0.99$. For larger population sizes, the distributions are quite narrow (see e.g., $N = 100$ in Fig. 5), and *the choice of the strategy for dealing with infeasible solutions is clearly not the dominating factor in describing the variability among EDPOIS.*

4.2. Additional observations

Additional observations are discussed in this section regarding the control parameter settings (Section 4.2.1) and specifically F (Section 4.2.2), the difference in DE algorithms caused by the crossover operators (Section 4.2.3), the maximum C_r value (Section 4.2.4), the interaction between parameters (Section 4.2.5) and the number of infeasible solutions are generated (Section 4.2.6).

4.2.1. Overall observations regarding parameter settings

For all DE configurations, POIS grows with the increase of control parameter values – both independently and simultaneously. Minimal values of control parameters induce very low POIS for all considered configurations. Increase in POIS with the increase in C_r is faster than with the increase in F. At the same time, increase in POIS for smaller population size is slower; meanwhile for many configurations with higher population size POIS values of either 0% or 100% prevail. This makes it problematic explaining how such populations manage to maintain higher diversity – a fact suggested by the theoretical analysis of DE [29]. Furthermore, the increase in POIS is not monotonous in some cases, see Section 4.2.4.

We conclude that connections between POIS and setting of DE control parameters is complex and requires further investigation.

4.2.2. Further discussion on the meaning of control parameter F

As mentioned in Section 2.1.1, parameter F has been originally thought to be within $(0, 2]$. However, in practice only values within $(0, 1]$ have been used widely in practice by the community. Results presented in this paper *confirm the validity of such empirical modification* since for majority of configurations high settings for F easily lead to 100% infeasible solutions generated during the run. This has potential to unnecessarily and prematurely decrease population diversity and leads to worsening of the algorithm's performance.

Thus, setting $F \in (1, 2]$ is indeed rarely justified unless it is used in conjunction with other components inducing lower POIS such as very small population size, exp *mutation or, sometimes, the maximum value of C_r.* All of those, however, also come at a price for the overall performance of the algorithm.

4.2.3. Exp and bin lead to two very different DE algorithms

Further theoretical inspection of control parameter C_r shows that its meaning depends on the specific crossover logic used. Algorithm 2 is based on a binomial distribution according to which each design variable has probability of being exchanged exactly equal to C_r. This means that the expected value of exchanged variables during the crossover process is C_r. Conversely, if Algorithm 3 is employed, a sequence of $m < n$ consecutive design variables has a probability to get exchanged which decreases exponentially (i.e. $(C_r)^m$ according to a geometric distribution). Hence, in this case, the crossover rate does not reflect the expected number of swaps. To tune C_r to reflect a required number of exchanges one can use the method in [20].

Therefore, results presented in this paper about exp *resulting in smaller POIS are perfectly in line with the discussion above.*

4.2.4. 'Collapse' of some EDPOIS for the maximum value of C_r

An interesting phenomenon is observed for a number of configurations (e.g., Fig. 8c) where POIS slowly increases with the increase of C_r value, for constant value of F, at times even 'saturating' in 100%, only to significantly drop down unexpectedly

later. Such 'collapse' happens only for the maximum C_r but consistently for different population sizes, mutation variants and strategies. As explained in Section 4.2.3, C_r controls the proportion of values inherited from the mutant vector. Thus, a naive explanation of the 'collapse' phenomena can be that mutant vectors are less prone to be infeasible than vectors after the crossover.

This observation clearly requires further investigation.

4.2.5. Nontrivial factor analysis

No single factor can be identified for describing the variability among all EDPOIS considered. Higher order factor analysis is required as considered configurations exhibit significant complex dependencies.

4.2.6. Too many infeasible solutions

In the setup described in this paper, significantly more ISs get generated throughout optimisation runs *than what might be expected*. For our experiments here, we have selected standard DE configurations and followed widely accepted recommendations by the DE community for setting control parameters [42]. And yet still, for this relatively low dimensional problem, *nearly every DE configuration ends up having 100% points generated as infeasible for every single run in a of series of 50 runs*. The same observation holds true even if we exclude the runs with $F \in [1, 2]$ which, over the years since DE has been proposed, has been excluded by collective intelligence of the community.

Objective function f_0 considered in this paper is indeed extremely difficult to be 'optimised' and this might be the reason for such a high number of ISs. In our opinion, POIS discussed here should serve as *upper bounds* for POIS generated for regular objective functions that are *necessarily* smoother than f_0.

Another reason for high POIS in results in this paper is the *dimensionality*. Researchers in computational intelligence and related fields routinely claim that, by modern standards, a 30-dimensional problem is in fact low dimensional [29]. However, mathematically speaking, it is not as it does present symptoms of the well-known 'curse of dimensionality' [1].

5. Conclusions and future work

This piece of research sheds light on the effect of different operators and parameters on the tendency of DE to generate infeasible solutions during optimisation runs rather than being exclusively contained within the domain boundaries. By doing this, we draw attention to an often overlooked aspect of DE's algorithmic behaviour. Indeed, our experimental methodology puts in evidence that many more solutions than what is generally expected are generated outside the 'box' within a single optimisation run. Studying the distribution of the proportions of infeasible solutions has led us to confirm some 'rule of thumb' for DE and its parameter settings, but also to observe several non-trivial facts about DE - as reported below:

- [-] Judging by our careful inspection of literature, researchers and practitioners are *rarely aware* of the high number of the ISs that get generated in various widely used DE configurations.
- [-] A large number of ISs potentially disrupts any search. However, having a relatively low number of such solutions allows the algorithm to 'learn' the boundaries, preferably without over-exploring this area. Absence of ISs might signify the under-exploration of some parts of the domain.
- [-] It is extremely easy to generate an IS in a highly multidimensional problem. Thus, choice of the strategy of dealing with ISs *should not be neglected* when designing algorithms for such problems. Practitioners should expect that in relatively highly dimensional problems, there is a fair chance that the vast majority of solutions evolved by an algorithm will be generated outside of the domain and therefore require to be somehow 'brought back' into the feasibility region, i.e. the domain. The proportion of such points can easily reach 100%.
- [-] The choice of strategy of dealing with generated infeasible solutions is among parameters that *control the proportion* of infeasible solutions generated during the DE run. Thus, it *should not be omitted* during the design of a particular DE used [12,22]. Other factors controlling POIS, to a varying degree, are the three DE parameters.
- [-] Results on POIS for f_0 presented in this paper should be considered as the upper bounds on POIS for general objective functions.
- [-] A carefully selected and tuned strategy of dealing with infeasible solutions can potentially warrant the algorithm's performance. Where possible, practitioners *should routinely include tracking* the (final) POIS within their implementations.
- [-] Researchers should *better acknowledge* the long established and confirmed differences in `DE/*/bin` and `DE/*/exp` configurations. No overreaching conclusions should be made about *DE in general* – it is highly doubtful that such conclusions would universally hold for both types of configurations.
- [-] Results of this paper suggest that setting of $F \in (1, 2]$ is indeed rarely justified which follows the empirical 'wisdom' of the DE community.

A number of aspects discussed in this paper requires further study. Among others, the connection between POIS and settings of the DE control parameters requires careful evaluation. Further work will also include higher order factor analysis of aspects influencing POIS distributions and devising ways to reliably interpolate EDPOIS for (F, C_r)-values not included in the original tabulation. Finally, we will continue searching for better, more compact and interpretable versions of EDPOIS visualisations.

CRediT authorship contribution statement

Anna V. Kononova: Conceptualization, Methodology, Validation, Visualization, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Fabio Caraffini:** Methodology, Validation, Software, Data curation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Thomas Bäck:** Formal analysis, Investigation, Writing - original draft, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R.E. Bellman, Dynamic programming, Princeton University Press, NJ, 1957.
[2] R. Biedrzycki, Handling bound constraints in cma-es: An experimental study, Swarm and Evolutionary Computation 52 (2020) 100627.
[3] J. Brest, B. Bošković, S. Greiner, V. Žumer, M.S. Maučec, Performance comparison of self-adaptive and adaptive differential evolution algorithms, Soft Computing 11 (7) (2007) 617–629.
[4] J. Brest, M.S. Maučec, Population size reduction for the differential evolution algorithm, Applied Intelligence 29 (3) (2008) 228–247.
[5] J. Brest, M.S. Maučec, Self-adaptive differential evolution algorithm using population size reduction and three strategies, Soft Computing 15 (11) (2011) 2157–2174.
[6] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, V. Zumer, High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, Hong Kong, China, June 2008, pp. 2032–2039.
[7] A. Caponio, A.V. Kononova, F. Neri, Differential evolution with scale factor local search for large scale problems, in: Y. Tenne, C.-K. Goh (Eds.), Computational Intelligence in Expensive Optimization Problems, Studies in Evolutionary Learning and Optimization, Vol. 2, Springer, Berlin Heidelberg, 2010, pp. 297–323.
[8] F. Caraffini, The Stochastic Optimisation Software (SOS) platform, 2019, doi:10.5281/zenodo.4678306.
[9] F. Caraffini, G. Iacca, The SOS platform: Designing, tuning and statistically benchmarking optimisation algorithms, Mathematics 8 (5) (May 2020) 785.
[10] Caraffini, F., Kononova, A.V., September 2018. Structural bias in differential evolution: a preliminary study. In: LeGO 2018–14th International Workshop on Global Optimization, Leiden, The Netherlands. Vol. 2070. AIP, Leiden, The Netherlands, p. 020005.
[11] F. Caraffini, A.V., Kononova, Differential evolution outside the box - extended results. www.doi.org/10.17632/cjjw6hpv9b.1, Mendeley Data, v1, 2020.
[12] F. Caraffini, A.V. Kononova, D.W. Corne, Infeasibility and structural bias in differential evolution, Information Sciences 496 (2019) 161–179.
[13] Caraffini, F., Neri, F., Iacca, G., April 19-21 2017. Large scale problems in practice: The effect of dimensionality on the interaction among variables. In: Squillero, G., Sim, K. (Eds.), Applications of Evolutionary Computation. EvoApplications 2017. Lecture Notes in Computer Science, vol 10199. Springer International Publishing, Cham, pp. 636–652.
[14] F. Caraffini, F. Neri, I. Poikolainen, Micro-differential evolution with extra moves along the axes, in: IEEE Symposium Series on Computational Intelligence, Symposium on Differential Evolution, IEEE, Singapore, April 2013, pp. 46–53.
[15] C.A.C. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Computer Methods in Applied Mechanics and Engineering 191 (11) (2002) 1245–1287.
[16] S. Das, S.S. Mullick, P. Suganthan, Recent advances in differential evolution - an updated survey, Swarm and Evolutionary Computation 27 (2016) 1–30.
[17] S. Das, P.N. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems Tech. rep., Jadavpur University, Nanyang Technological University, Kolkata, 2010.
[18] S. Das, P.N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (1) (Feb 2011) 4–31.
[19] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computation, Springer-Verlag, Berlin, Germany, 2003.
[20] G. Iacca, F. Caraffini, F. Neri, Compact Differential Evolution Light: High Performance Despite Limited Memory Requirement and Modest Computational Overhead, Journal of Computer Science and Technology 27 (5) (2012) 1056–1076.
[21] A.V. Kononova, F. Caraffini, H. Wang, T. Bäck, Can compact optimisation algorithms be structurally biased?, in: T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, H. Trautmann (Eds.), Parallel Problem Solving from Nature – PPSN XVI, Springer International Publishing, Cham, 2020, pp. 229–242.
[22] A.V. Kononova, F. Caraffini, H. Wang, T. Bäck, Can single solution optimisation methods be structurally biased?, in: 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, pp 1–9.
[23] A.V. Kononova, D.W. Corne, P.D. Wilde, V. Shneer, F. Caraffini, Structural bias in population-based algorithms, Information Sciences 298 (2015) 468–490.
[24] Lampinen, J., Zelinka, I., 2000. On stagnation of the differential evolution algorithm. In: Oŝmera, P. (Ed.), Proceedings of 6th International Mendel Conference on Soft Computing. pp. 76–83 .
[25] Liang, J.J., Qu, B.Y., Suganthan, P.N., 2013. Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. Tech. rep., Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore.
[26] J. Liu, J. Lampinen, Population size adaptation for differential evolution algorithm using fuzzy logic, in: A. Abraham, K. Franke, M. Köppen (Eds.), Intelligent Systems Design and Applications, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 425–436.
[27] K.R. Opara, J. Arabas, Differential evolution: A survey of theoretical analyses, Swarm and Evolutionary Computation 44 (2019) 546–558.
[28] G. Pavai, T.V. Geetha, A survey on crossover operators, ACM Comput. Surv. 49 (4) (2016).
[29] A. Piotrowski, Review of differential evolution population size, Swarm and Evolutionary Computation 32 (2017) 1–24.
[30] V.P. Plagianakos, D.K. Tasoulis, M.N. Vrahatis, A review of major application areas of differential evolution, in: Advances in Differential Evolution, Springer, Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 197–238.
[31] K. Price, R. Storn, Differential evolution: A simple evolution strategy for fast optimization, Dr. Dobb's J. Software Tools 22 (4) (1997) 18–24.

[32] K.V. Price, Differential evolution, in: Handbook of Optimization, Springer, Berlin, Heidelberg, 2013, pp. 187–214.
[33] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution, Springer-Verlag, Berlin/Heidelberg, Natural Computing Series, 2005.
[34] Qin, A.K., Suganthan, P.N., 2005. Self-adaptive differential evolution algorithm for numerical optimization. In: 2005 IEEE congress on evolutionary computation. Vol. 2. IEEE, pp. 1785–1791.
[35] A. Qing, Differential evolution: fundamentals and applications in electrical engineering, Wiley-IEEE press, NJ 07030 USA, 2009.
[36] Storn, R., Price, K., 1995. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, ICSI.
[37] P. Suganthan, S. Das, S. Mukherjee, S. Chatterjee, Adaptation methods in differential evolution: A review, 20th International Conference on Soft Computing MENDEL, Vol. 2014, Springer, Brno, Czech Republic, June 2014, pp. 131–140.
[38] R. Tanabe, A.S. Fukunaga, Improving the search performance of shade using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 1658–1665.
[39] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1997) 67–82.
[40] Yaman, A., Iacca, G., Caraffini, F., 2019. A comparison of three differential evolution strategies in terms of early convergence with different population sizes. In: AIP Conference Proceedings. Vol. 2070. AIP Publishing LLC, p. 020002.
[41] J.M. Yeoh, F. Caraffini, E. Homapour, V. Santucci, A. Milani, A clustering system for dynamic data streams based on metaheuristic optimisation, Mathematics 7 (12) (Dec 2019) 1229.
[42] D. Zaharie, Critical values for control parameters of differential evolution algorithm, in: R. Matuŝek, P. Oŝmera (Eds.), Proceedings of 8th International Mendel Conference on Soft Computing, 2002, pp. 62–67.
[43] D. Zaharie, Control of population diversity and adaptation in differential evolution algorithms, in: D. Matousek, P. Osmera (Eds.), Proceedings of MENDEL International Conference on Soft Computing, Springer, Brno, Czech Republic, 2003, pp. 41–46.
[44] D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms, Applied Soft Computing 9 (3) (2009) 1126–1138.
[45] D. Zaharie, F. Micota, Revisiting the analysis of population variance in differential evolution algorithms, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, San Sebastian, Spain, 2017, pp. 1811–1818.