Infeasibility and structural bias in Differential Evolution

Fabio Caraffini[a,*], Anna V. Kononova[b,c], David Corne[b]

[a]*Institute of Artificial Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester, LE1 9BH, UK*
[b]*Department of Computer Science, Heriot-Watt University, Edinburgh, EH14 4AS, UK*
[c]*School of Communication, Media and Information Technology, Hogeschool Rotterdam, The Netherlands*

Abstract

This paper investigates a range of popular differential evolution (DE) configurations to identify components responsible for emergence of structural bias – a recently identified tendency of algorithms to prefer some regions of search space over others, for reasons unrelated to objective function values. Previous work has explored this tendency for genetic algorithms (GA) and particle swarm optimisation (PSO), finding a relationship between population size and extent of structural bias, hence highlighting potential weaknesses of those algorithms. In current article, we focus on DE, extend the investigation to include consideration of an algorithmic component that is often overlooked – constraint handling mechanism. Towards this end, a wide range of DE configurations was tested here. Results suggest that DE is generally robust to structural bias. Unlike the case with GA and PSO, population size seems to have no influence on DE structural bias. Only one of variants studied – DE/current-to-best/1/bin – shows clear signs of bias, however, we show that this effect is mitigated by a judicious choice of constraint handling technique. These findings contribute towards explaining widespread success of DE variants in algorithm comparison studies; its robustness to structural bias represents the absence of a factor that may confound other algorithms.

Keywords: structural bias, algorithmic design, differential evolution, population-based algorithms, optimisation

1. Introduction

The modern world is replete with optimisation tasks of varying size and difficulty. For the majority of these problems that are faced in commerce and industry, the task of solving them *exactly*, and in an acceptable timescale, is unachievable. The latter may either be due to high dimensionality, the presence of tight constraints, the computationally expensive nature of evaluating the objective function(s), or due to some

*Corresponding author
Email addresses: fabio.caraffini@dmu.ac.uk (Fabio Caraffini), anna.kononova@gmail.com (Anna V. Kononova), d.w.corne@hw.ac.uk (David Corne)
URL: www.cse.dmu.ac.uk/~fcaraf00 (Fabio Caraffini)

combination of these and other features. Thus, the techniques used to address these problems are invariably drawn from the area of stochastic, approximate optimisation, which offers a very wide range of algorithms that, while never guaranteeing exact solutions, will nevertheless find good solutions, sometimes even near-optimal, in reasonable time. Among the available options, a common choice for solving such problems is to use algorithms from the fields of Evolutionary Computation (EC) and Swarm Intelligence (SI). The most successful EC and SI algorithms happen to be population-based metaheuristics, where certain principles of natural systems are taken as a source of inspiration to design the operators that select, mix and manipulate individuals that make up a population of candidate solutions. These algorithms broadly operate by simulating an evolutionary process which iteratively improves the quality of candidate solutions as the algorithm progresses, leading towards optimal or near-optimal solutions. The most commonly used EC and SI frameworks, now regularly deployed in several engineering applications [23, 32, 47, 48], in system control and other industrial processes [11, 18], in robotics [19, 26], and in many other subjects and real world scenarios, are Genetic Algorithms (GA), Evolution Strategies (ES), Particle Swarm Optimisation (PSO) and Differential Evolution (DE).

However, metaheuristics are certainly not flawless; a number of undesirable algorithmic behaviours commonly surface during the optimisation process. For example, the well known phenomenon of “premature convergence” can prevent population-based algorithms from exploring the search space to an adequate extent. Such behaviour is quite common in GAs [1, 28] as a consequence of employing overly exploitative adaptive operators [49] or due to the selection mechanism not being able to maintain adequate diversity in the evolving population. Similarly, stagnation can interrupt the exploratory process even in the presence of high diversity, as frequently occurs in DE [25]. In fact, convergence is not generally guaranteed for the majority of the aforementioned stochastic algorithms: only a few have a general proof of convergence, see e.g. [44, 50, 53], meanwhile for other algorithms such proofs exist only under rather restrictive hypotheses [6, 17].

Researchers are yet to find good explanations for the aforementioned undesired behaviours, thus leaving practitioners with little or no guidance for choosing the most appropriate optimisation method for the problem at hand. In other words, reliable “off-the-shelf” solvers for specific classes of real-world problem do not currently exist. The challenges a practitioner encounters when faced with a real-world optimisation problem are therefore not necessarily limited to those related to the nature or complexity of the optimisation problem itself. Often, lack of theoretical insights into the dynamics inside general purpose algorithms, together with scant empirical indications on how to select and tune the most appropriate algorithms available in the literature, make it very difficult for practitioners to be effective at solving general optimisation problems, and for computer scientists to be efficient in designing novel methods.

Currently, a widely varied list of factors conspire to confound the design of “ready-to-use” algorithms. From a *technical* point of view, we can make the following observations:

- the No Free Lunch Theorem (NFLT) by Wolpert and Macready [54] states that a universal algorithm for black-box optimisation cannot exist; it follows that problem specific information needs to be available to tailor algorithms to a given problem;

- even though algorithm design for given classes of problem can be partly automated by techniques such as hyperheuristics [5] and memetic computing (MC) [36] (paradigms for which the NFLT seems not to hold [43]), it is still a partially blind process due to the lack of theoretical knowledge regarding the internal dynamics of EC and SI algorithms;
- the exploration/exploitation conundrum is still unsolved, especially given to the necessity of performing the optimisation process under a limited computational budget in real-world applications.

Moreover, from a *practical* point of view, we can mention the following:

- the current tendency to incrementally improve upon previous algorithms, e.g. by employing multiple search strategies and self-adaptive operators as in [3, 12], has led to the emergence of over-complicated optimisation frameworks carrying a high algorithmic overhead, and thus being less suitable for implementation in devices with limited memory, inadequate for real-time problems and large-scale tasks [18], and meanwhile also being more difficult to tune;
- hybrid approaches, from the fields of Memetic Computing or hyperheuristics in particular, are full of examples of heavy algorithmic skeletons, such as in [14, 33, 39], where multiple algorithms are arbitrarily merged and therefore requiring “meta-optimisation” over a high number of artificial and, thus difficult-to-interpret, parameters [35, 30].

To overcome these issues, researchers have recently started investigating possible ways for tuning parameters, either off-line [15] or on-the-fly.

To shed light on these open research questions, recent studies have focused their attention on the variability of individuals inside populations, and have found *structural biases* in popular metaheuristics such as GA and PSO [24]. The presence of such bias in the search process has been established to be intrinsic to the algorithmic structure and the search logic, as well as correlated with the common parameters, first and foremost being the population size parameter. This finding has many implication, not least of which is a potential positive impact on the algorithm design and configuration process. Recently, a preliminary study started to explore structural bias in the context of a limited selection of Differential Evolution (DE) algorithms [7]. The latter study uncovered interesting links between structural bias and particular combinations of the DE operators. In this paper we report on a comprehensive and further extended investigation of structural bias in DE, covering a full range of popular DE algorithm variants.

The remainder of the paper is structured as follows:

- Section 2 summarises the literature on algorithmic bias, presenting the methods used to investigate structural bias, summarising previous results, and discussing constraint handling strategies (including a treatment of the fact that correction strategies may themselves introduce bias);

- Section 3 briefly presents the Differential Evolution framework and four of the most popular schemes used in this field;
- Section 4 clarifies the objectives of the present study, and gives details on the method and experimental setup deployed to address these objectives, leading to the later presented results;
- Section 5 comments on the results;
- Section 6 concludes the study, outlining its main message, and highlighting strengths and weaknesses of the proposed approach.

2. Background

An algorithm is said to *possess structural bias* when it is unable to explore all areas of the search space to a uniform extent, *irrespective of the fitness function* [24]. When faced with the task of optimising a given function, a general-purpose optimisation algorithm is expected to be able to locate the optima regardless of where they are located in the search space. Following the general idea of iterative optimisation algorithms, values of the objective function (of the points that are the current candidate solutions) effectively operate as a gradient that pulls the search in the direction of improvement as prescribed by the algorithm at hand. However, as has been established in [24], the combined action of the algorithm's operators may conspire to introduce an additional "pull", independent of that associated with the objective function. Thus, the trajectory of the population ends up being influenced by a complex *superposition* of two forces that are not necessarily in agreement: the unknown "evolutionary" pull originating from the incremental re-sampling of candidate solutions and their fitnesses, and the unknown pull stemming from the "sum" of individual biases of the algorithm's operators when applied to the current candidate solutions. Theoretically, in this superposition of pulls, the first is dominated by the objective function, while the second is predominantly defined by the algorithm's structural bias.

For an algorithm not to exhibit structural bias, its generating and mixing operators should be able to reach every part of the search space without imposing any preferences on some regions of the domain over others. If this is not the case, the search process will suffer from structural bias, and researchers or practitioners may need to perform a thorough analysis to identify the configuration of operators that cause it. Clearly, different combinations of functions, domains, constraints and corresponding constraint handling strategies may have different impacts on the extent of bias, making it very difficult to tease out the primary triggering mechanisms. Therefore, a suitable protocol must be defined which allows algorithmic biases to be detected at the same time as allowing the causes to be identified reliably. After a discussion of previous results on structural bias in Section 2.1, the method previously established to produce the results presented in this article is described in Section 2.2.

2.1. Existing results on bias

A first attempt at demonstrating the presence of structural biases was made by the PSO community. The empirical observation that some PSO algorithms were incapable of exploring too far from the origin of the search space, i.e. so called "centre-seeking bias" (CSB), and also too far from the initial swarm of candidate solutions, i.e. "initialisation-region bias" (IRB), have attracted the attention of researchers and paved the way for more thorough analyses. Initially, a great deal of attention was paid to finding algoirthmic mechanisms to avoid these biases. The study in [2] introduced selection operators to relax the rigid "one-to-one-spawning" logic that at the time has been thought to be a source of such undesired phenomena. However, little effort has been made to understand and define its actual nature – a proper formalisation for CSB and IRS was finalised later on in [13]. In the meantime, controversial studies, such as [34], which speculates that CSB exists in all population based algorithms, have started looking at different optimisation paradigms. The latter, refuted in [22], has been followed by further investigations on PSO, such as [21], where it has been shown that, unlike DE [9, 10], PSO seems sensitive to rotations of the objective function. In this light, it is worth mentioning that an "angular" bias can also be defined for PSO [51].

Not long after the latest results in [13], a generalised definition of "structural bias", applicable to all population-based metaheuristics, was given in [24], in which the PSO paradigm was shown to be plagued by an intrinsic structural bias, with both theoretical and empirical evidence. Further, the latter study also considered a simple GA, thus countering the assumption made in [34] (that "all population-based algorithms have centre-seeking bias") and showing how also optimisation algorithms equipped with selection and genetic operators can display a biased search logic. The latter finding shed some doubt on the approach proposed in [2], where selection was employed to reduce the effect of biases, with no particular justification for this choice. Most importantly, it was shown theoretically in [24] that, under certain conditions, structural bias in GAs correlates positively with population size. This was also validated empirically. Since population size is a "common denominator" among population-based algorithms for real-valued global optimisation, this discovery can be considered significant, and confronts the common belief that a large population size is beneficial for a number of reasons (e.g. a summary is given in [46]), and is key to tackling large-scale problems efficiently [29].

Finally, a graphical approach to visualising structural bias, as a non-uniform clustering of the population over time, was developed. The visual approach for representing structural bias in [24] has been adopted by other researchers in the field, who have recently adopted this method to, for example, identify and then remove structural biases from the so called JADE and SHADE-based algorithms [42]. It has also been used to stress that current state-of-the-art algorithms tend to be over-complicated, and should instead be designed via a more informed algorithmic design process that considers structural bias as a key factor to avoid [40, 41]. Finally we note that the latter articles each focus on particular DE variants; meanwhile, comprehensive analysis of structural bias in a range of DE variants was initiated in a preliminary study [7], and is significantly extended here.

2.2. *Testing for bias*

The most suitable choice of testbed for the identification of structural bias, in terms of its effects on the distribution of the final best solutions over multiple runs, is a series of experiments, with a pre-selected fixed dimensionality, on the function

$$f_0 : [0,1]^n \subset \mathbb{R}^n \rightarrow [0,1] \subset \mathbb{R}, \quad f_0(x) = \text{Uniform}\,(0,1) \tag{1}$$

In words, function f_0 simply implements a non-deterministic mapping, whereby the fitness of any solution vector x is a uniform random scalar between zero and one, in no way actually depending on x. As rigorously explained in [24], with f_0 as the objective function, over a series of independent runs an ideal unbiased algorithm should return a uniform distribution of best final solutions. In this sense, f_0 serves as a reference problem which *by design allows a decoupling between artefacts of the objective function and artefacts arising from iterative application of algorithmic operators*. The value of f_0 at any point has no correlation with either the values within its neighbourhood, or past evaluations of this point. This is achieved by effectively eliminating the influence of the local positions of the candidate points, but retaining the underlying algorithmic artefacts.

It is worth noting that the discussion above holds for all population-based algorithms, regardless of the differences in their algorithmic components. This includes, for example, a Differential Evolution variant in which parent selection is missing (since it employs the "one-to-one-spawning" mechanism, as does PSO, which seems to be biased), but individuals are necessarily selected at the mutation level either at random, or according to a stochastic rank-ordering over a specific set of values of the objective function in the current population.

The procedure for testing an optimiser on f_0 consists in repeating optimisation over a pre-established fixed number of independent runs and studying the resulting distribution of the final best solutions per run. A value of 50 runs has been used in our earlier studies to promote statistically significant results. A practical visual approach, originally proposed in [24] and successfully re-employed in [7], consists in displaying the obtained final distribution of solutions in "parallel coordinates" [20, 24]: coordinates of a vector in an n-dimensional space are marked correspondingly on n equally spaced parallel lines. Position of a marker on this line indicates the position of point in the domain in this dimension with respect to the boundaries. In details, each vertical line would display 50 markers representing solutions from a series of 50 runs of the algorithm on f_0. For this reason, the horizontal axis will span from 1 up to 30 dimensionality values (as $n = 0$ in this study), meanwhile the vertical axis in the interval $[0, 1]$ (since each design variable is constrained in such domain in definition of f_0).

Thus, an unbiased algorithm would yield a figure with points homogeneously filling the entire interval in each dimension. Conversely, in the presence of a strong structural bias, clusters will appear as best solutions would tend to accumulate in one or more segments of each or some parallel lines. As amply demonstrated in [24], it is exactly the independence of the value of f_0 in one point on values of neighbouring points that allows decoupling of the landscape of the objective function and cumulative undesired effect of algorithm's operators on the population referred here as structural bias. It is worth highlighting that landscape of f_0 has random nature and, thus, different realisations of f_0 would generate different images, all of them being extremely rugged.

2.2.1. *Random generator effects*

It is worth mentioning that an extensive study has been carried out in [24] to demonstrate that structural bias cannot be attributed to the artefacts of the random generator used. The essence of this study consisted in a “reductio ad absurdum“ approach. This began by assuming that there exists a correlation between random numbers used to generate coordinates of the two subsequent points examined by the algorithm, and that correlation is significant enough to measure. To find such correlations between elements of the pseudorandom sequences, thus proving the impact of the random number generator on the structural bias, three tests were designed and executed. The first test was run to examine the correlation between consecutive pairs of random values used to generate some specific dimension values of points in the search space. Similarly, the second test was run to examine the correlation between all dimensions simultaneously. Finally, the third test was run to track the correlation between consecutive values (as in the first test) in the whole pseudorandom string. Full details are available in [24]. No evidence of correlation was found in the tests, and the conclusions of the study unequivocally suggested that observations of structural bias on f_0 do not originate from the random generator but rather represent artefacts from the iterative application of the algorithmic operators.

To conclude, it has to be remarked that all algorithms discussed in the current publication employ the same Java $48-$bit pseudorandom generator as in [24], which is based on the linear congruential generator (LCG) [27] with a period of $2^{48} \approx 2.8 \times 10^{14}$ and the seed automatically generated by means of the system time routine *System.currentTimeMillis()*[1]. Thus, the authors find unnecessary to redo such analysis for the current paper – this question is considered settled unless a different random generator is used.

2.3. *Constrained problems*

To complicate things further, a great deal of optimisation problems are explicitly or implicitly constrained: ranging from simple inequalities that produce a hypercube domain through to the complex disconnected sets defined through simulations or as solutions of large systems of complex equations. Most constraint optimisation problems can be considered as ambiguously defined since function values outside the domain usually tend not to be specified. Unfortunately, constraint handling is not straightforward in EAs as traditional variation operators are blind to constraints [16]. This means that feasibility of parents does not guarantee feasibility of solutions they generate – numerical variation operators are generally unaware of the boundaries of the search domain. Therefore, unless a very specific restrictive operator or encoder which somehow exploits regularities of the feasible search space [31] is used inside the algorithm to ensure feasibility, a strategy must be chosen to guide how to deal with solutions generated outside the domain at any stage of the algorithm. Thus, reduction of the amount of time spent generating infeasible solutions becomes an additional (possibly, implicit) *objective for the algorithmic design*.

[1]For reference, the total length of DNA molecules in all cells of an adult human body is of the order of 10^{14} meters.

Over the years, a variety of strategies has been developed – a summary of general state-of-the-art constraint handling techniques is given below [31]:

- **Penalise**: the fitness value of a newly generated solution outside the domain is substituted by a predefined penalty value (binary, distance-based, time-dependent or some other adaptively calculated value);
- **Dismiss** (death penalty): if a newly generated solution is outside the domain, it is dismissed and either re-generated or replaced by one of the parent solutions;
- **Correct** (repair): given a newly generated solution, z, outside the domain, generate a new feasible solution that is a function of z, via one of several specific methods (e.g. saturation, toroidal, local search, ...);

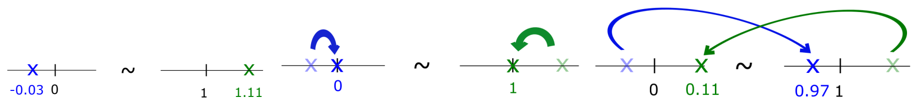
Another approach that has been explored are probabilistic formulations, in which constraint handling techniques are used only for a given proportion of out-of-domain solutions that enter the population. However such approaches have largely fallen out of fashion [31, 37]. Finally, the advantages and disadvantages of the aforementioned techniques are summarised in Table 1; it should be noted that constraint handling operators, as an integral factor in an algorithm's design, may clearly contribute towards the algorithmic bias. However, our current level of understanding is not sufficient to pinpoint their impact.

Table 1: Advantages and disadvantages of common constraints handling techniques used to deal with solutions generated outside the domain

Dismiss	• Unaltered objective function	✓
	• Wasteful of computational resources • Can easily lead to no result	✗
Penalise	• Traditional and ”cleanest” way to deal with constraints	✓
	• Allows unfeasible individuals (problematic in applications) • Can result in an infeasible result • Can be viewed as a distortion of the original objective function • Introduces extra design choices requiring investigation and justification[2] • Requires knowing the range of values of the objective function	✗
Correct	• Very straightforward • Provides additional moves to the algorithm facilitating diversity	✓
	• Distorts objective function if the correction operator is biased • Introduces extra decisions that need to be taken by algorithm designer, e.g.: which particular correction method to choose; whether or not to inherit the corrected genotype; in case of using the local search, whether or not to use the greedy algorithm and in what order to examine the variables.	✗

A further way in which all *correction strategies* can be classified is as either *superficial* or *complete*. *Superficial correction strategies* limit their impact to only reassigning coordinates of out-of-domain points, transforming them to points within the domain *without* re-evaluating the resulting corrected point. Solutions corrected with such methods become in some sense “alien” to the original objective function – they are the artefacts of the design choice (a choice of correction strategy) rather then the objective function itself – and comparison of two algorithms employing drastically different correction strategies is akin to comparing apples and so-called "Chinese apples".

A graphic elaboration of examples of superficial correction strategies popular in the field of EAs is shown in Figure 1.



(a) Solution out of domain (b) saturation correction result (c) toroidal correction result

Figure 1: Schematic explanation of correction strategies for domain $[0, 1]$

Meanwhile, *complete correction strategies* actively search for other feasible inside-the-domain points in the (relative) vicinity of the currently unfeasible point. This search is typically performed via some local search routine with a prescribed complexity and budget of function evaluations. Thus, complete strategies ensure that corrected solutions fully represent the original objective function, unlike superficial strategies that only assign artificially fitness values to moved-previously-unfeasible points. However, complete correction strategies create "holes" in the domain, filled with infeasible points, for which no quality information is provided to the optimisation method. This stands in opposition to a long-standing principle of EAs, which sees them as operating via combining partial information from the entire population [31]. Use of complete correction strategies introduces further difficulties, such as being very time-consuming, and the possibility of transferring limitations of the chosen local search algorithm into a failure to find suitable solutions by the overall method. Moreover, there is a risk that domain boundaries end up being over-explored due to the locality of the search concentrated around the boundary region[3]. In practice, mostly due to time limitations and unwillingness to introduce additional design choices, the impact of employing an adequate correction strategy is often underestimated. Moreover, *under a widely spread assumption that the correction strategy has limited importance*, justification of choice of constraint handling method is either omitted or vaguely mentioned. We argue that such an assumption is, in fact, *erroneous*.

We also argue that, in practice, *significantly more solutions* end up requiring correction than is usually assumed by an algorithm designer. The overall number of corrections required during the optimisation provides a (rough) estimate for the degree to which the actual function being optimised corresponds to the "true" real-world quality measurement that it aims to approximate, thus validating the use of local information. In other words, *an overly high number of corrections deprives the algorithm from using information about the local structure of the landscape under investigation in most*

[3]This highlights a subtle connection between the choice of correction strategy and structural bias. Results presented in this paper suggest that, potentially, the correction strategy plays an important role in the formation of structural bias of the method

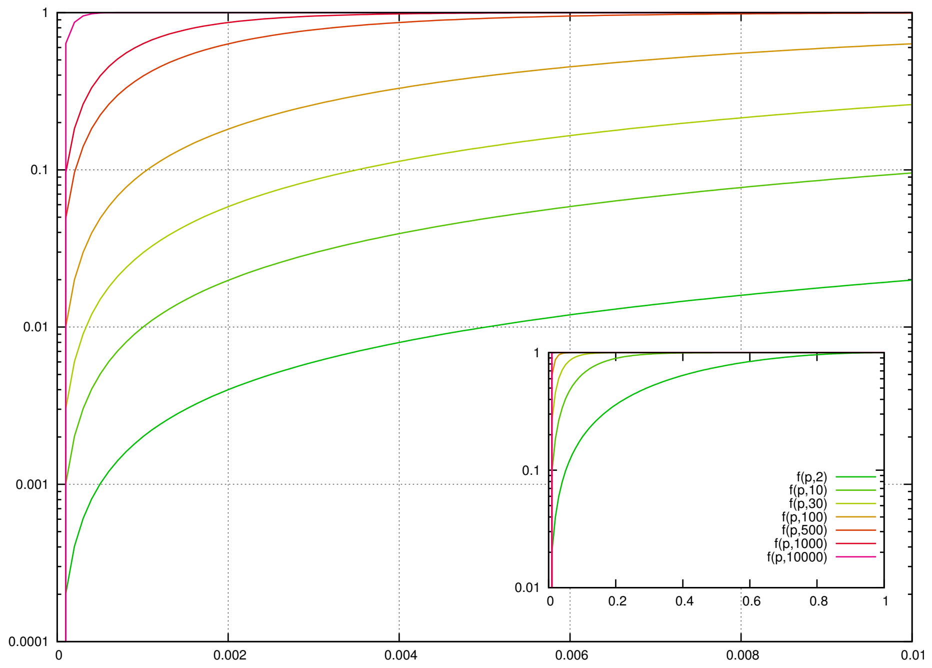


Figure 2: Tabulation of the probability of a solution requiring a correction in at least one dimension, as expressed by $f(p, n) = 1 - (1 - p)^n$. Multiple curves correspond to different values of dimensionality n defined for values of the constant rate p at which corrections become necessary, independently for different dimensions (shown in horizontal axis). Note how fast probability gets close to 1 for values of dimensionality over 30. Zoom out to $[0, 1]$ is shown in smaller figure in the same colours.

critical boundary regions.

Clearly, the number of corrected solutions in the population grows with the dimensionality of the problem, since a solution with only one coordinate outside the domain already requires handling/correction. For a simplified experiment, suppose jumps outside the domain in different dimensions occur independently and with a constant rate $p \in [0, 1]$[4]. The probability that a solution requires correction in at least one dimension is then trivially expressed as $f(p, n) = 1 - (1 - p)^n$, where n is problem's dimensionality. To help with visualisation, Figure 2 shows tabulations of this formula for various values of dimensionality (values of p are shown on the horizontal axis and $f(p, n)$ on the vertical axis). It is easy to see that $f(p, n)$ attains high values for relatively low values of p and this effect rapidly increases with dimensionality n. For a rather modest (by modern standards) value of dimensionality of $n = 100$, the probability that at some iteration a correction is required in at least one dimension exceeds 0.99 for $p > 0.045$. As dimensionality increases further, virtually all solutions end up requiring corrections

[4]Obviously, this is a simplification of the real situation as this rate is not constant and there might be dependencies between dimensions.

for majority of reasonable values of p. This clearly demonstrates the importance of the correction strategy choice for high dimensional problems – *an arbitrarily chosen correction strategy may transform the original optimisation problem into a trivial one which structurally has little to do with the original function*.

Depending on the nature of the problem, some constraint handling technique choices might even distort the search in feasible regions; for example, physical limitations of some equipment used to compute objective function values might result in unnecessary corrections. Thanks to design, some strategies may lead to identical behaviour for some algorithms (e.g. penalise and dismiss in DE, as explained in Section 4). For other strategies, the performance of an algorithm can turn out to be highly sensitive to the constraint handling strategy, as clearly demonstrated by results presented later in this paper. However, despite these observations, the choice of constraint handling technique is usually not investigated thoroughly during the design of an optimisation algorithm. We aim to draw the attention of algorithm designers and practitioners to the importance of this choice. *It turns out that, as usual, it matters what happens on domain boundaries*.

3. Differential Evolution

Differential Evolution is a powerful yet simple metaheuristic for global real-valued optimisation which only requires three parameters to function efficiently [52]: the scale factor $F \in [0, 2]$, the crossover ratio $Cr \in [0, 1]$ and the population size NP. The DE framework consists of the iteration of only three steps, as shown in Algorithm 1. In view of its "one-to-one-spawning" selection mechanism, it resembles Swarm Intelligence methods such as PSO but, at the same time it is similar to Evolutionary Algorithms such as the GA, as it requires a "mutation" operator, followed by a "crossover" strategy to produce a new solution. Conventionally, the mutation operator in EC tends to follow crossover. However, in DE, the term "mutation" arises from the fact that the operator must be applied, to some individuals of the population (see e.g. Formulas 2 to 5), to generate the so called "mutant vector". The latter, here indicated with $\mathbf{x_m}$, then participates in crossover with an individual $\mathbf{X}$ (see Algorithm 1) to produce an offspring. Thus, if GA terminology is used for DE, the mutant vector and $\mathbf{x_m}$ and the current individual $\mathbf{X}$ represent the parents.

Algorithm 1 Differential Evolution

```
$g \leftarrow 1$                                                      ▷ First generation
$\mathbf{Pop^g} \leftarrow$ randomly sample $NP$ individuals within the search space $\boldsymbol{D} \subset \mathbb{R}^n$
$\mathbf{x_{best}} \leftarrow$fittest individual$\in \mathbf{Pop^g}$
while condition on budget do
    for each $\mathbf{x} \in \mathbf{Pop^g}$ do
        $\mathbf{x_m} \leftarrow$Mutation                              ▷ e.g. Formula 2 in [52]
        $\mathbf{x_{offspring}} \leftarrow$CrossOver$(\mathbf{x}, \mathbf{x_m})$          ▷ e.g. Algorithm 2 in [52]
        if $f(\mathbf{x_{offspring}}) \leq f(\mathbf{x})$ then
            $\mathbf{Pop^{g+1}} \leftarrow \mathbf{x_{offspring}}$
        else                                 ▷ Fill the new population for the next iteration
            $\mathbf{Pop^{g+1}} \leftarrow \mathbf{x}$
        end if
    end for
    $g \leftarrow g + 1$                          ▷ Replace the old with the new generation
    $\mathbf{x_{best}} \leftarrow$ fittest individual $\in \mathbf{Pop^g}$            ▷ update best individual
end while
Output Best Individual $\mathbf{x_{best}}$
```

Despite the absence of fitness-informed selection and a fixed order for the operators, a number of different algorithmic behaviours can be obtained from DE by changing and combining mutation and crossover operators. The most commonly used mutation operators are:

- rand/1:
$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) \tag{2}$$
- rand/2:
$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) + F\left(\mathbf{x_{r_4}} - \mathbf{x_{r_5}}\right) \tag{3}$$
- best/1:
$$\mathbf{x_m} = \mathbf{x_{best}} + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) \tag{4}$$
- current-to-best/1:
$$\mathbf{x_m} = \mathbf{x} + F\left(\mathbf{x_{best}} - \mathbf{x}\right) + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) \tag{5}$$

Other configurations of DE exist and are described in the literature [45]. With reference to equations (2) to (3), indices $r_1 \neq r_2 \neq r_3$ are randomly sampled in order to pick the random individual from the populations.

In modern terminology, the initial version of DE proposed in 1995 is DE/rand/1/bin; it includes a "binomial" crossover operator where each variable has a fixed probability Cr of being exchanged. In subsequent implementations, a new crossover strategy was proposed which exchanges bursts of consecutive components whose length depends on the Cr value and the dimensionality of the problem. As the probability of exchanging one more coordinate in the burst follows the geometric progression, and thus decays exponentially, this variant is commonly referred to as DE/rand/1/exp. Implementation details of bin and exp crossover strategies are given in Algorithms 2 and 3 respectively.

Algorithm 2 Binomial crossover

```
Input two parents $\mathbf{x_1}$ and $\mathbf{x_2}$                    ▷ $\mathbf{x_1}, \mathbf{x_2} \in \mathbf{D} \subset \mathbb{R}^n$
$\mathbf{x_{offspring}} \leftarrow \mathbf{x_1}$
Index $\leftarrow \mathcal{I}$                          ▷ $\mathcal{I}$ is uniformly sampled in $[1, n] \subset \mathbb{N}$
for $i = 1, \ldots, n$ do
    if $\mathcal{U} \leq CR$ or $i =$ Index then        ▷ $\mathcal{U}$ is uniformly sampled in $[0, 1] \subset \mathbb{R}$
        $\mathbf{x}^{(i)}_{\mathbf{offspring}} \leftarrow \mathbf{x}^{(i)}_{\mathbf{2}}$        ▷ exchange the $i^{th}$ component
    end if
end for
Output $\mathbf{x_{offspring}}$
```

Algorithm 3 Exponential crossover

```
Input two parents $\mathbf{x_1}$ and $\mathbf{x_2}$                    ▷ $\mathbf{x_1}, \mathbf{x_2} \in \mathbf{D} \subset \mathbb{R}^n$
$\mathbf{x_{offspring}} \leftarrow \mathbf{x_1}$
$i$, Index $\leftarrow \mathcal{I}$                     ▷ $\mathcal{I}$ is uniformly sampled in $[1, n] \subset \mathbb{N}$
do
    $\mathbf{x}^{(i)}_{\mathbf{offspring}} \leftarrow \mathbf{x}^{(i)}_{\mathbf{2}}$            ▷ exchange the $i^{th}$ component
    $i \leftarrow i + 1$
    if $i > n$ then
        $i \leftarrow 1$
    end if
while $\mathcal{U} \leq CR$ and $i \neq$ Index        ▷ $\mathcal{U}$ is uniformly distributed in $[0, 1] \subset \mathbb{R}$
Output $\mathbf{x_{offspring}}$
```

In modern terminology, any particular scheme can be obtained by combining the chosen mutation and crossover strategies, which is reflected in the "DE/a/b/c" notation. The first member, "a", refers to the vector being mutated (namely the one to which difference vectors are added), "b" is the number of difference vectors used and "c" indicates the crossover. For example, "a" could be "rand" (Formulas 2 and 3), "best" (Formulas 4) or even a combination of two vectors as in "current-to-best" (Formula 5).

The various mutation operators provide a set of different strategies for moving across the search domain to handle different landscape scenarios. The current-to-best/1 strategy, for example, can be of help in speeding up convergence when dealing with less complex fitness functions (functions with plateau-like regions as opposed to highly multimodal or ill-conditioned functions) [9, 10]. However, it would normally be inadequate for highly multimodal functions, since it gives high priority to the direction of the current best solution – a basic rand/1 would be preferred in this case.

Some simple observations can be made about the DE framework, which is indeed quite distinct from its closest relatives in the EC and SI fields. First, it can be noted that some level of individual selection is present in the mutation operator, which requires a number of candidate solution to be selected from the population. Second, it can be argued that DE mutation operators, since they performs a linear combination of individuals, are identical to the arithmetic crossover operators used in real-valued GAs. In this light, DE can be seen as crossover-driven (crossover occurs twice per iteration), with no mutation at all. This distinguishes DE from ES algorithms, which are mainly mutation-driven, and leads to further considerations as DE and ES are both successfully used despite the common belief that complete EAs need both mutation and crossover operators. Also, it suggests that too much attention is often paid by the EC community towards pursuing a metaphor-based algorithmic design at the expenses of a more informed one. For the latter to take place, better understanding is needed of the population dynamics induced by a given set of operators and their associated rates.

Despite the low number of parameters in DE, their tuning plays a major role, since performance can be quite sensitive to parameter settings. A skilled algorithm designer may be able to exploit parameter settings to achieve desired overall behaviour. For example, a well-chosen combination of scale factor and crossover rate can theoretically be used to control the nature of the search (hence the term “control parameters”) by positioning it in a desired spot along the “exploitative/explorative” continuum. Unfortunately, this is not always achievable due to the lack of precise indications in the context of arbitrary objective functions. The scale factor F, which is used to control the extent of exploration in mutations operators, was initially thought to have a large range of potentially effective values, specifically $F \in (0, 2]$, where the the value $F = 0$ is usually not considered as it will nullify the effect of the difference vector of the mutation operator (see Formulas 2 to 5). In contrast, the received wisdom concerning crossover rate has been that it should lie in $[0, 1]$, probabilistically, with extreme values typically avoided, to prevent the crossover operator either replacing only 1 component(see Algorithm 2 and 3) or returning an exact copy of one of the parents. However, it later became clear that only values of F in $(0, 1]$ were used in practical applications. Moreover, the study in [29] recommended $F \in [0.5, 0.9]$ and $Cr \in [0.8, 1]$. In practice, users of DE tend to settle on values close to 0.7 for the scale factor, and several DE algorithms have been proposed in which such parameters are self-adapted to the problem during the optimisation process [3], or picked up from a pool of promising values [12]. A key study in [56] has instead dug into the inter-relationship among parameters F and Cr, and discovered an optimal tuning of the crossover rate as a function of the scale factor.

While the research literature is rich of studies on the setting of F and Cr, significantly fewer studies have investigated the setting of NP. In one case that we have found, where [29] recommended that the population size should be set at 10 times the value of dimensionality of the problem, the recommendations seem questionable. In (for example) problems with dimensionality in the '000s, this would usually be an infeasible population size, leading to undesirably slow convergence [38]. This recommendation also runs counter to the observation that shrinking the population size can be beneficial to avoid stagnation [4].

Further, while large population size is known to promote diversity, thus reducing the risk of premature convergence [55], an overly large population size may amplify the deleterious effect of structural bias. As demonstrated theoretically in [24] for the case of GAs, the spread of the population across the domain is directly affected by the number of points in the population: unfortunately enough, as GA population size grows, so does the strength of structural bias. However, the situation is less clear for other optimisation frameworks and, in particular, for DE. According to the preliminary study in [7] such correlation is only partially confirmed. Surprisingly, it turned out that while the “current-to-best/1” mutation operator seems to carry a visible structural bias, a simpler “rand/1” operator is to some extent capable of mitigating the bias regardless of the chosen crossover strategy. Thus, [7] is extended here for a wider range of aspects, such as different parameter settings, correction strategies and DE schemes.

4. Objectives, methods and experimental setup

The main objective of this paper is to analyse a wide range of popular DE configurations and try to identify a combination of mutation and crossover operators responsible for the existence of structural bias in DE algorithms, as well as to observe to what extent the bias can be mitigated by judicious choice of control parameters F, Cr, constraint handling strategy and population size NP. This knowledge can be exploited to achieve a more informed algorithmic design process and to give practitioners guidance in tuning DE’s control parameters for real-world optimisation problems.

4.1. Choice of DE configurations

As explained in Section 2.2, identification of structural bias is based on the minimisation of function in Equation (1) [5]. Dimensionality is kept at $n = 30$; this has the advantage of being sufficiently high-dimensional to be comparable with many complex real-world problems, while at the same time being low enough to allow for extensive experimentation on modest computing resources in reasonable time. Additionally, $n = 30$ corresponds to our previous work, and thus enabled us to form judgements about the relative amount of any discovered bias in comparison therewith. How structural biases may or may not vary with dimensionality is an interesting potential area for future work, but one which would may complicate analyses in the current effort.

Meanwhile, to leave nothing to chance, the effect of different constraint handling strategies is also taken into consideration in this study. Experiments in this article have been carried out with the following implementations of popular correction methods introduced in Section 2.3:

1. **penalise**: $f_0 : [0,1]^n \to [0,1]$ is extended via a surjective penalty function $f_P : \mathbb{R}^n \to [0,1] \cup \{c\}$ mapping solutions outside the $[0,1]^n$ domain with a pre-fixed real valued constant $c \notin [0,1]$:

$$f_P(\mathbf{x}) = \begin{cases} f_0 & \text{if } \mathbf{x} \in [0,1]^n \\ 2 & \text{otherwise} \end{cases}; \tag{6}$$

[5]without loss of generality, conclusions here also hold true for the maximisation case

2. **saturation**: keeping the original fitness value, modify those coordinates outside the domain as shown in Figure 1(b) and Algorithm 4 (superficial correction);
3. **toroidal**: keeping the original fitness value, modify those coordinates outside the domain as shown in Figure 1(c) and Algorithm 5 (superficial correction).

It must be pointed out that a popular "*dismiss*" correction strategy is omitted from this study as it would be equivalent to using a penalty function, due to the one-to-one spawning logic in DE. Indeed, in the most general case this is implemented as in Algorithm 6, in which the infeasible solution is simply discarded and replaced with a parent chosen according to any convenient logic. However, the same replacement necessarily occurs in DE for penalised solutions as they are discarded based on a direct comparison based on their fitness function value. In contrast, it makes sense to differentiate between "penalise" and "dismiss" methods in EA paradigms such as e.g. some GAs for which selection takes place stochastically.

Algorithm 4 Saturation correction

```
Input a solution $\mathbf{x}$ and problem's boundaries    ▷ $\mathbf{x} \in \mathbb{R}^n$ but domain is $\mathbf{D} \subset \mathbb{R}^n$
for $i = 1, \ldots, n$ do
    l←$i^{th}$ lower bound
    u←$i^{th}$ upper bound
    if $\mathbf{x}^{(i)} > u$ then
        $\mathbf{x}^{(i)}_{\mathtt{saturated}} \leftarrow$ u    ▷ saturate the $i^{th}$ component to the upper-bound
    else if $\mathbf{x}^{(i)} < l$ then
        $\mathbf{x}^{(i)}_{\mathtt{saturated}} \leftarrow$ l    ▷ saturate the $i^{th}$ component to the lower-bound
    else
        $\mathbf{x}^{(i)}_{\mathtt{saturated}} \leftarrow \mathbf{x}^{(i)}$    ▷ keep the original $i^{th}$ component
    end if
end for
Output $\mathbf{x}_{\mathtt{saturated}}$
```

Algorithm 5 Toroidal correction

```
Input a solution $\mathbf{x}$ and problem's boundaries    ▷ $\mathbf{x} \in \mathbb{R}^n$ but domain is $\mathbf{D} \subset \mathbb{R}^n$
for $i = 1, \ldots, n$ do
    l←$i^{th}$ lower bound
    u←$i^{th}$ upper bound
    $x_N \leftarrow \frac{\mathbf{x}^{(i)} - l}{u - l}$    ▷ normalise $i^{th}$ component of $\mathbf{x}$ in $[0, 1]$
    $x_R$ ←rounds $\mathbf{x}^{(i)}$ to the nearest integer towards 0
    if $x_N > 1$ then
        $x_N \leftarrow x_N - x_R$
    else if $x_N < 0$ then
        $x_N \leftarrow 1 - |x_N - x_R|$
    end if
    $\mathbf{x}_{\mathbf{corrected}}{}^{(i)} \leftarrow l + X_N \cdot (u - l)$    ▷ scale the corrected value back to $\mathbf{D}$
end for
Output $\mathbf{x}_{\mathbf{corrected}}$
```

Algorithm 6 Discard correction

```
Input a solution $\mathbf{x}$ and problem's boundaries    ▷ $\mathbf{x} \in \mathbb{R}^n$ but domain is $\mathbf{D} \subset \mathbb{R}^n$
if $\mathbf{x} \notin \mathbf{D}$ then
    $\mathbf{x} \leftarrow$ a selected parent
end if
Output $\mathbf{x}$    ▷ No fitness functional call required
```

The mutation strategies presented in Section 3, Formulas 2 to 5, were combined with both bin and exp crossovers, i.e. Algorithms 2 and 3 respectively, thus, forming 8 DE schemes. Each one was equipped with the three aforementioned correction schemes (i.e. Formula 6, Algorithm 4 and 5) thus generating 24 different DE configuration variants.

Finally, it is worth emphasising that this study only considers "box-constrained" problems [6]. Thus, only constraints due to the presence of upper and lower bounds for each design variable are to be handled. These problem are often erroneously referred to as "unconstrained continuous problems" as there are no equality and inequality constraints other than the domain boundaries, and, despite the fact that metaheurstics are key in dealing with discontinuous real-valued problems, they cannot be optimised analytically.

4.2. *Choice of DE parameters*

To decide on the most appropriate parameter settings for DE variants selected in the previous section, extensive preliminary experimentation has been carried out. Keeping the number of runs and population sizes consistent with [24, 7], DE control parameters were *investigated in the light of the percentage of points corrected throughout the whole optimisation process*. The logic behind the latter approach was to focus on the algorithmic behaviour, and stick as much as possible to optimising the true f_0, thus minimising potential side effects introduced by the constraint handling strategy. In this sense, percentage of corrections over the run can be considered a *metric* that, under the right conditions, captures how aggressive, or badly-suited, the correction strategy is. On the other hand, the choice of DE control parameters is balanced empirically by the fact that too few corrections potentially mean under-exploitation of the domain boundaries, since generating operators have relatively local nature. Thus, the most reliable F-Cr pair was determined by looking into 25 pre-selected combinations of control parameters F and Cr, i.e. those generated with $F \in \{0.05, 0.2, 0.4, 0.7, 0.9\}$ and $Cr \in \{0.05, 0.4, 0.7, 0.9, 0.99\}$. Such tabulation of the intervals of control parameters is empirically motivated by regions of particular interest.

This means that all 24 configurations of DE were considered with three population sizes, each producing 25 histograms (one per $F - Cr$ pair) – resulting in 1800 setups of full optimisation runs, each being executed 50 times independently, each for a budgeted number of fitness evaluations, keeping track of the percentage of corrections. From distributions of these percentages of corrections, mean and standard deviation surface plots were produced to support visual processing of this large amount of data. Values of $0.1 - 0.2$ for the $F - Cr$ control parameter pair have been chosen for all DE configurations to force the algorithms under study to operate, as far as possible, within the problem's boundaries. To avoid an unnecessarily long list of graphs here, all these results are made available online [8].

For demonstration purposes only, one example for the DE/rand/1/bin with penalty correction case and three different population sizes (5, 20 and 100) is shown here in Figure 3. Detailed analysis of the percentage of corrections in different configurations of DE has proved to be particularly interesting and slated for further study. To clarify, subsfigures in Figure 3 depict distributions of the percentage of occurred corrections, while optimising f_0, for each combination F and Cr (i.e. 25 sub-diagrams per subfigure). Each sub-diagram carries two layers of information. The first layer, shown in red, represents the distribution of correction percentages in a series of 50 runs. Percentages can be read on the y-axis (which points upwards and whose range is always $[0, 1]$), while the number of runs is reported on x-axis (which points to the right). Values shown on x-axis are less of interest than the shape of distribution. The second layer, shown in blue, indicates the employed values of F and Cr for particular subdiagram. Also for this layer, the y-axis points upwards and the x-xis to the right, but they report values for F and Cr respectively, both within $[0, 1]$. For both layers, origin is in the lower left corner. To summarize, this Figure attempts to draw a three-dimensional figure in projections in two dimensions: distributions shown in red in each small subfigure should be placed on the page in a perpendicular fashion towards the reader, in points marked in blue circles. Choice of number of bins in histograms is based on the investigation explained in the next section.

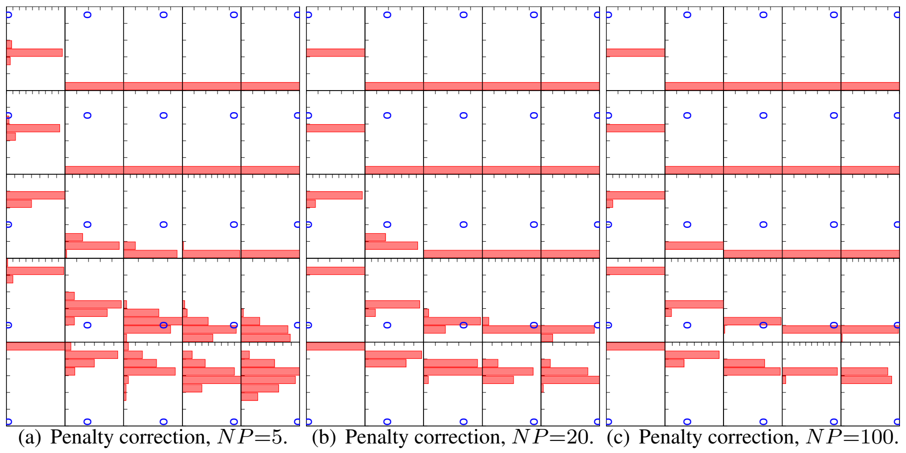
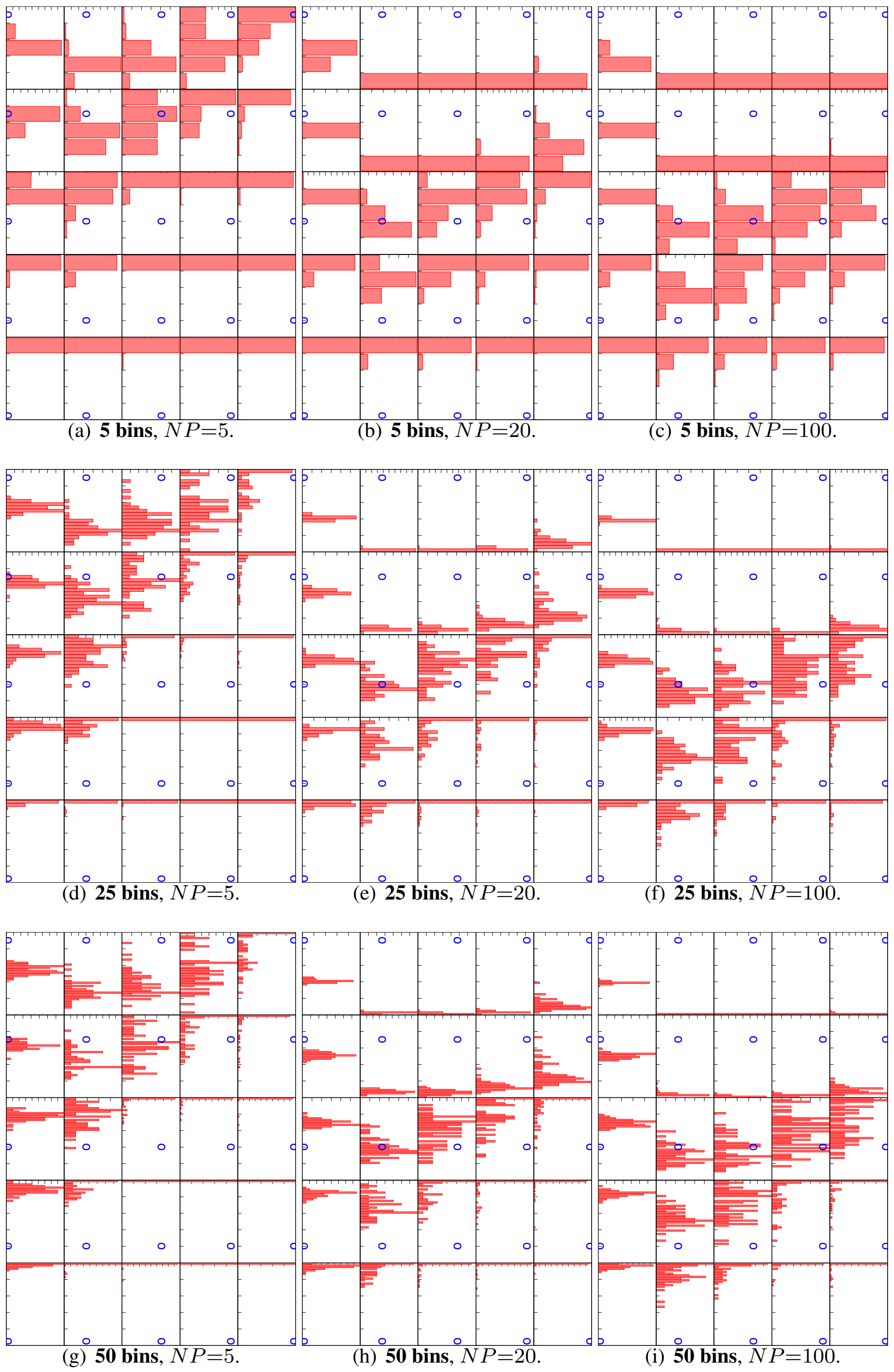

(a) Penalty correction, NP=5. (b) Penalty correction, NP=20. (c) Penalty correction, NP=100.

Figure 3: Example of distributions of percentage of corrections for DE/rand/1/bin for various values of F and Cr. For explanation of axes and colours, see the Section 4.2.

To summarise, having chosen appropriate values of DE control parameters, for the purpose of studying structural bias in DE configurations, all aforementioned 24 DE configuration variants were each run 50 times on f_0 for the $n = 30$ dimensional case with a standard computational budget of $10000 \times n = 300000$ fitness functional calls with $F = 0.1$, $Cr = 0.2$ and population size of $NP \in \{5, 20, 100\}$. Thus, in total, $24 \times 3 = 72$ optimisation processes were repeated for 50 runs to generate the results shown in Section 5.

4.3. Choice of granularity

To generate meaningful distribution figures such as Figure 3, it has become clear that bin size needs careful consideration. Choosing a number of bins for a histogram is a balancing act which depends on the number of points and features of the underlying distribution. Too few bins, and the resulting histogram potentially smooths over significant characteristics of the distribution. Too many bins, and key characteristics of the distribution are potentially drowned in noise. Clearly, the number of bins should not exceed the number of data points. As no educated *a priori* assumptions could be made regarding properties of the distribution of the percentage of corrections, an initial study was made to decide on the appropriate number of bins in distribution plots. The DE/best/1/bin configuration with saturation correction has been chosen as representative, and distributions of the percentage of corrections have been visualised for three population sizes $(5, 20, 100)$ for three different numbers of bins (5, 25, 50), see Figure 4 where plot layout is identical to that of Figure 3.


(a) **5 bins**, NP=5. (b) **5 bins**, NP=20. (c) **5 bins**, NP=100.

(d) **25 bins**, NP=5. (e) **25 bins**, NP=20. (f) **25 bins**, NP=100.

(g) **50 bins**, NP=5. (h) **50 bins**, NP=20. (i) **50 bins**, NP=100.

Figure 4: Experiments for choosing the number of bins in distributions of the percentage of corrections for various values of F, Cr and populations sizes for DE/best/1/bin with saturation correction on f_0. For explanation of axes and colours, see the Section 4.2.

Clearly, histograms with 50 bins (last row in Figure 4) are too noisy to interpret. Histograms with 25 bins (middle row in Figure 4) provide no discernible improvement, while 5 bins (top row in Figure 4) loses too much information. Thus, it has been decided to use 10 bins for all histograms of the percentage of corrections as is done in Figure 3.

4.4. Further results on distributions of the number of corrections

To help with interpreting multidimensional data shown in Figure 3, Figures 5 depicts for DE/rand/1/bin example over 50 runs, for three population sizes, and for every of 25 $F - Cr$ combinations *only the small portion of the data* – the mean percentage of corrections.

More specifically, in each subfigure of Figure 5, the vertical axes show values of F, while the horizontal axis shows values of Cr - both parameters run here in $[0, 1]$ and origin is in the left bottom corner. Colour encodes the value of the *a posteriori* computed average percentage of corrections for experiments with given F and Cr values – experimentally obtained values of averages are marked with small dots, meanwhile averages for the remaining $F-Cr$ couples are interpolated and numerically smoothed. Black curves are interpolated contour curves (isocurves) for the average percentage of corrections. The colour axis is the same for all subfigures – it runs from 0 values in blue to 1 values in gold. The pair of $F - Cr$ values chosen for experiments in this paper $(0.2, 0, 1)$ is marked with a circle. *In other words*, each subdiagramm of Figure 3 is represented in Figure 5 only by its average – selection of experimentally obtained a posteriori average values are indicated with small circles in terms of $F - Cr$ values meanwhile average values for other $F-Cr$ combinations are interpolated; average percentage of corrections values themselves are shown in colour according to the uniform scale.

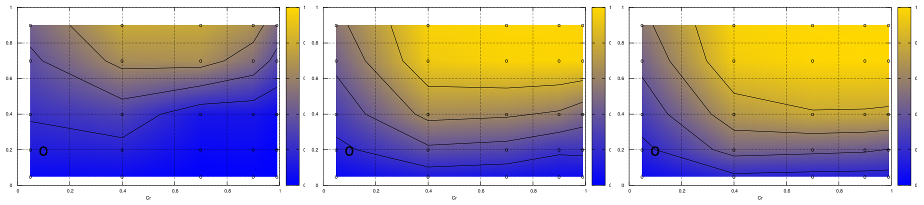

(a) $NP = 5$ (b) $NP = 20$ (c) $NP = 100$

Figure 5: Example of interpolated surfaces of average percentage of corrections for DE/best/1/bin with toroidal correction for three population sizes for various values of F (vertical axis $[0, 1]$) and Cr (horizontal axis $[0, 1]$). For full explanation about axes and colours, see Section 4.4.

Following similar approach, Figure 6 shows the standard deviation of the percentage of corrections for DE/rand/1/bin over 50 runs, for three population sizes, for every of the 25 $F - Cr$ combinations – each subdiagramm of Figure 3 is represented in Figure 5 only by its standard deviation – selection of experimentally obtained a posteriori standard deviation values are indicated with small circles in terms of $F - Cr$ values meanwhile standard deviation values for other $F - Cr$ combinations are interpolated;

standard devitation of percentage of corrections values themselves are shown in colour according to the uniform scale.

More specifically, as before, in each subfigure, vertical axes shows values of F, while the horizontal axis shows values of Cr. Both parameters run in $[0, 1]$ with origin in the left bottom corner. Colour encodes the value of the *a posteriori* computed standard deviation of percentage of corrections for experiments with given F and Cr values – experimentally obtained values of standard deviation are marked with small dots, while standard deviations for the remaining $F - Cr$ pairs are interpolated and numerically smoothed. Black curves are interpolated contour (iso-)curves for the standard deviation of the percentage of corrections. Colour axis is the same for all subfigures – it runs from 0 values in green to 0.31 values in violet. The pair of $F - Cr$ values chosen for experiments in this paper $(0.2, 0, 1)$ is marked with a circle.

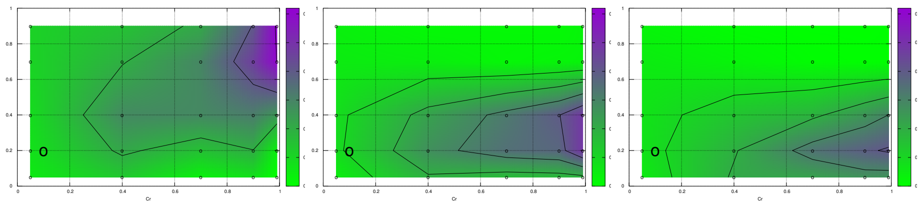

(a) NP=5. (b) NP=20. (c) NP=100.

Figure 6: Example of interpolated surfaces of standard deviation of the percentage of corrections for DE/best/1/bin with toroidal correction for three population sizes for various values of F (vertical axis $[0, 1]$) and Cr (horizontal axis $[0, 1]$). For full explanation about axes and colours, see Section 4.4.

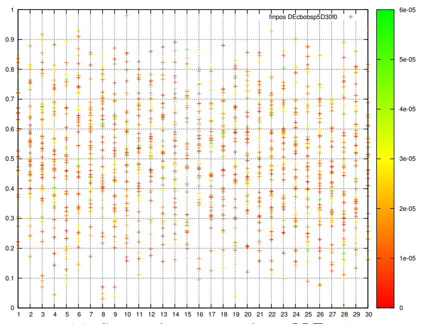
As mentioned previously, detailed analysis of the distributions, averages and standard deviations of the percentage of corrections for different configurations of DE will appear shortly as a companion publication. However, it is worth mentioning here that, unexpectedly, at least from the point of view of the analysis of the percentages of corrections, control parameters of DE seem to have different and, moreover, usually opposite effects in different configurations of DE. In other words, our results seem to show that, from the point of view of search dynamics, DE effectively transforms into a different algorithm with changes in its configuration – *dynamics inside DE are highly sensitive to the detailed specification of its operators*.

Apart from other conclusions, this also suggests that it is rarely sensible to talk about ”tuning control parameters for DE”. For example, high values of F do not necessarily always mean large steps of the algorithm, as popular research suggests [25] – if it was so, the number of corrections would consistently be high for all DE configurations, regardless the chosen value of Cr, since a higher number of larger steps correlates with a higher numbers of corrections. Our results suggest that this is not the case – higher numbers of corrections are indeed attained for higher values of F, but for some configurations this consistently happens for higher values of Cr and for some – for low values of Cr. Clearly, more research is required in this direction.

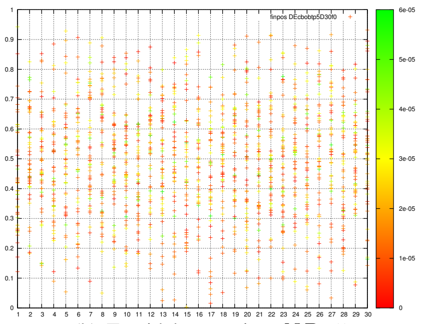
5. Results on structural bias

Graphical results have been generated as explained in Section 2.2 and grouped according to the specific DE scheme in the file [8] made available online, for visual inspection. To facilitate their analysis, the nine possible configurations of correction strategies and population size are positioned there on the same page, thus giving a general view of each one of the eight DE configurations considered in this study.

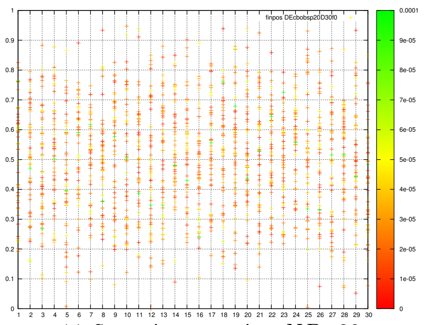
A glance at these results immediately reveals clear differences between DE/current-to-best/1/bin and the other schemes under examination, as it is the only configuration to display a clear structural bias. All biased patterns have been grouped in Figure 7 in contrast to Figure 8 where some of the remaining unbiased patterns are showed.



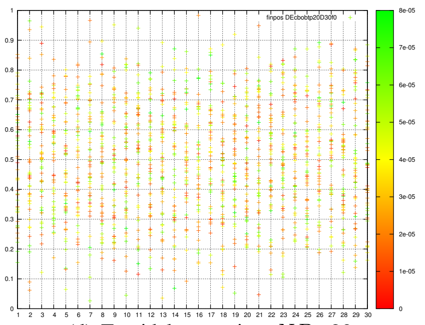
(a) Saturation correction, NP=5.



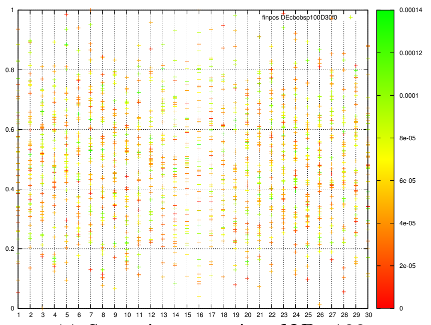
(b) Toroidal correction, NP=5.



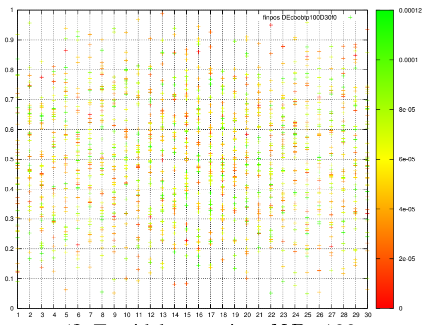
(c) Saturation correction, NP=20.



(d) Toroidal correction, NP=20.



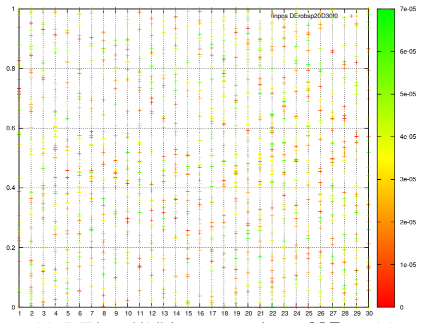
(e) Saturation correction, NP=100.

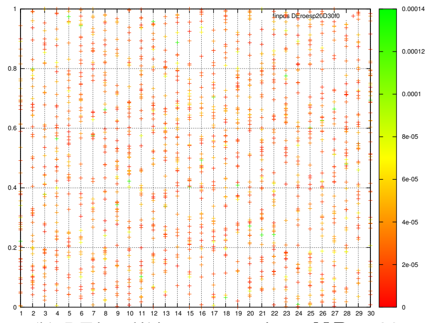


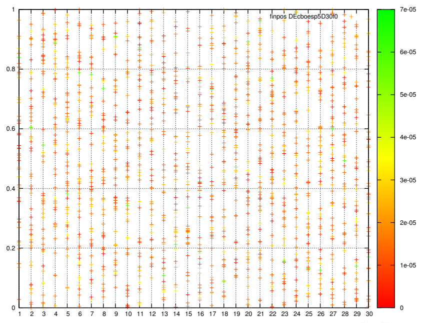
(f) Toroidal correction, NP=100.

Figure 7: Structural bias results in parallel coordinates for DE/current-to-best/1/bin with saturation correction (left column) and with toroidal correction (right column) with population size NP equal to 5 (top row), 20 (middle row) 100 (bottom row). Horizontal axis show dimensions from 1 to 30, meanwhile vertical axis shows the position of final solution in this dimension in the domain. Colour of points encodes their objective function values, according to the bar on the right. Each vertical line has 50 markers representing solutions from a series of 50 runs of the algorithm on f_0.

Thus, after the first observation, one must conclude that Differential Evolution is a robust and less biased optimisation paradigm than certain others, such as GA and PSO, which were examined in [24]. Furthermore, it appears that simpler DE variants, e.g. DE/rand/1/exp, which uses a randomised mutation, are robust to structural bias, while more complex ones, e.g. DE/current-to-best/1/bin, are more prone to carry a bias. This observation is consistent with what was observed in [42] for a popular self-adaptive DE variant referred to as the JADE algorithm (which displays a bias), and also supports the emerging belief that optimisation heuristics should, on the whole, be simplified [41]. Most importantly, it is interesting to note that simpler mutation schemes, as DE/rand/1, DE/rand/2 and DE/best/1, do not seem to carry significant structural biases, neither when the binomial crossover is employed, nor when it is replaced with the exponential one (see top row of Figure 8). Moreover, they are not sensitive to the choice for the correction strategy and, despite what was observed in [24] – where the strength of the structural bias increases with increasing population sizes in GA and PSO based optimisation – the number NP of individuals seems not to have a direct impact on the biases of these algorithmic structures (see bottom row of Figure 8).


(a) DE/rand/1/bin - saturation - $NP = 20$


(b) DE/rand/1/exp - saturation - $NP = 20$


(c) DE/current-to-best/1/exp - saturation - $NP = 5$

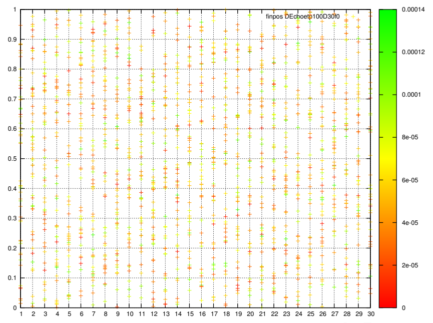
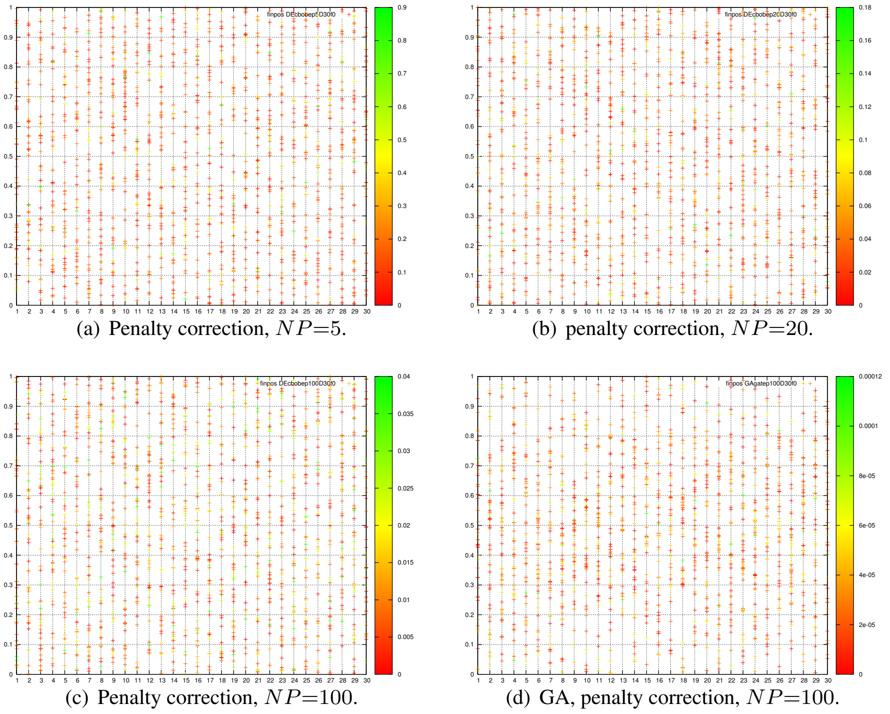

(d) DE/current-to-best/1/exp - toroidal - $NP = 100$

Figure 8: Structural bias results in parallel coordinates for several unbiased DE configurations with different population sizes and different combinations of mutation, crossover, and correction strategy. Colour of points encodes their objective function values, according to the bar on the right. See Figure 7 for explanation of the layout.

Conversely, it is evident that DE/current-to-best/1/bin tends to prioritise specific regions as the best individuals accumulate towards the centre of the search space at the end of the optimisation process. Once again, in contrast to what was previously observed in [24], the displayed bias seems to only marginally depend on the population size, as can be seen from Figure 7. It must be pointed out that, even though this was already noted in the preliminary study in [7], in which saturation and toroidal corrections were employed, a peculiar behaviour arose in this examination while trying the penalty approach of Formula 6, so eliminating its structural bias (see Figure 9). It is worth remarking that a similar result cannot be easily obtained with other optimisation paradigms. This is graphically shown in Figures 9(d) which displays a visible, albeit not strong, structural bias for a GA equipped with penalty correction.



(a) Penalty correction, NP=5.
(b) penalty correction, NP=20.
(c) Penalty correction, NP=100.
(d) GA, penalty correction, NP=100.

Figure 9: Structural bias results in parallel coordinates for DE/current-to-best/1/bin with penalty correction and population size $NP = 5$ in 9(a), $NP = 20$ in 9(b), and $NP = 100$ in 9(c). For comparison, Subfigure 9(d) shows the effect of the application of penalty correction to a Genetic Algorithm with population size $NP = 100$, Arithmetic crossover, Gaussian mutation, and tournament selection. As previously, colour of points encodes their objective function values, according to the bar on the right. See Figure 7 for explanation of the layout.

Interestingly enough, results obtained with three different population sizes and three different correction strategies further confirmed the "mitigating" effect of the crossover operator for the structural bias of DE/current-to-best/1 [7]. Indeed, see the bottom row of Figure 8, DE/current-to-best/1/exp unexpectedly appears not to be biased at all, according to visual inspection. An explanation for this discrepancy may lay in the fact that, for a given Cr value, the average number of exchanged design variables is higher if binomial crossover is used, whereas it is lower in presence of exponential crossover. The latter logic is indeed based on a different working principle according to which the probability of exchanged components decays exponentially and depends on the dimension n of the problem – a fair comparison between two DE algorithms employing bin and exp crossovers should consider the ad-hoc crossover rates $Cr_{bin} = Cr$ and $Cr_{exp} = \frac{1}{nCr\sqrt{2}}$ respectively [10]. This means that, for the considered dimension value $n = 30$, only a few components are in fact exchanged. One can therefore speculate that such a low value is simply not enough to manifest a visible structural bias. However, this should not be the case as it would implies that only the crossover operator, and in particular the binomial one, introduces biases into the DE framework while all considered mutation strategies are unbiased. At the current state of this investigation, this does not seem to be reasonable considering that both the bin and exp options have been used with other mutation schemes in this study, but clear biases, visible to the naked eye, were not apparent.

The latter considerations suggest that the structural bias of an optimisation algorithm is not simply the result of the sum of biases of its operators and can also arise under particular mutation-crossover combinations, or in general, under specific combinations of both biased and unbiased operators. In this light, structural bias can also be seen as a non-linear phenomenon where the biases of two biased operators do not necessarily add up if they are combined within an algorithm, but rather generate spurious contributions biasing the search further and differently.

To gain further insight, an additional complete correction strategy, which we call "one tailed normal" correction, was investigated. In this strategy, out-of-domain points were mapped non-deterministically back into the domain, enabling a more insightful analysis of the distribution of infeasible solutions when corrected near to the problem's boundaries.

In detail, this additional strategy simply re-samples design variables smaller than the lower bound (0 for each design variable of f_0) from $|\mathcal{N}(0, \frac{1}{3})|$, where $\mathcal{N}$ is a two tailed normal distribution. Conversely, design variables greater than the upper bound (1 for each design variable of f_0) are drawn from $1 - |\mathcal{N}(0, \frac{1}{3})|$. The strategy is implemented as a recursive method, because newly generated values could still fall outside the search domain thus requiring further correction rounds. However, recursion was rare in practice due to the chosen small value for $\sigma = \frac{1}{3}$, that tends to keep corrected values within the domain and close to the boundaries (since $3\sigma = 1$).

This strategy is not from the literature and thus was not mentioned in Section 4.1, but was tested with the identical experimental setup discussed in section 2.2. Results confirmed what was observed with popular correction strategies, with a moderate bias arising only in the case of DE/current-to-best/1/bin. For this reason, graphical results are made available in [8] but omitted in this manuscript to avoid an unnecessarily extended gallery of similar images. However, it is worth noting that this latter experiment further confirmed the observation that a structural bias arises as a result of the combination of the operators forming the algorithm, and not as the simple sum of the biases carried by the individual operators. Indeed, the One Tailed Normal correction is intrinsically biased by design (as it is likely to correct infeasible solutions close to the problem's bounds), but does not add additional biases to DE/rand/1/exp - which is unbiased. Moreover, DE/rand/1/bin - which is biased - seems to display a less aggressive structural bias when equipped with such a correction strategy.

6. Conclusions

This paper systematically analyses a wide range of popular DE configurations in the light of structural bias. The latter is a phenomenon that manifests during the optimisation process as the algorithm “prefers” some regions of the search space to others, independently of the objective function. Such behaviour has been clearly demonstrated in other EA algorithms and shown to stem from iterative application of algorithmic operators, each potentially plagued with their own shortcomings. These shortcomings are exacerbated by the algorithm’s structure and could thus limit the overall performance of the algorithm due to uneven exploration of the search space.

Compared to previous studies, a new aspect, usually overlooked during the algorithmic design stage, is considered as a potential contributor to the formation of structural bias, namely, the choice of constraint handling technique. In the present work, we find that the constraint handling technique can be crucial in mitigating the effects of structural bias on an otherwise highly biased DE configuration. This is evident in DE/current-to-best/1/bin, whose bias can be eliminated by simply employing the penalty constraint handling strategy. In this light, practitioners are recommended to use DE/current-to-best/1/bin in combination with such a constraint handling method, and also algorithm designers are urged to pay more attention to the often too-briefly considered design details which could, surprisingly, make a big difference.

The triggering mechanisms for structural bias in DE has also turned out to be different from those in GA and PSO – in particular, population size in DE has no clear effect on the strength of structural bias. Our main results suggest that the structural bias of an optimisation algorithm is really a superposition and not a sum of effects of individual operators, as in general it can arise from a specific combination of both biased and unbiased operators generating spurious contributions, and causing further distortion of the desired search dynamics.

In the future, promising areas of investigations are then two-fold: first, to examine the contribution of algorithmic operators (in support of our observation that the system as a whole can be biased regardless of the innocence of its individual components), their structural bias has to be detected and quantified; second, alternative ways to visualise/quantify bias, e.g. clustering, can be designed to flag critical cases and confirm the absence/presence of structural bias. These would clarify unsolved matters and help understand the triggering mechanism for the bias, not only in DE.

Finally, an additional intriguing direction for further research is suggested by aspects of the approach we have used in attempt to understand the dynamics of DE populations, both herein and in [24]. To interrogate such dynamics, metrics were crafted – such as percentage of corrections (and its associated statistics) – and models were then built (see section 4.4) to help interpret those metrics. When the relationship between an algorithm’s performance and its design/parameter settings is very complex – as seems to be the case for the majority of metaheuristics – the use of such proxy models may be a valuable tool for taming this complexity. However, at the same time we must also avoid artefacts that arise purely from the simplifications inherent in any modelling process.

References

[1] Andre, J., Siarry, P., Dognon, T., 2001. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. Advances in engineering software 32 (1), 49–60.

[2] Angeline, P. J., May 1998. Using selection to improve particle swarm optimization. In: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360). pp. 84–89.

[3] Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V., Dec 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10 (6), 646–657.

[4] Brest, J., Maučec, M. S., 2008. Population size reduction for the differential evolution algorithm. Applied Intelligence 29 (3), 228–247.

[5] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J. R., 2019. A classification of hyper-heuristic approaches: Revisited. In: Handbook of Metaheuristics. Springer, pp. 453–477.

[6] Caraffini, F., 5 2016. Algorithmic issues in computational intelligence optimization: from design to implementation, from implementation to design. No. 243 in Jyväskylä studies in computing. University of Jyväskylä.
URL `http://urn.fi/URN:ISBN:978-951-39-6742-0`

[7] Caraffini, F., Kononova, A. V., 2019. Structural bias in differential evolution: A preliminary study. AIP Conference Proceedings 2070 (1), 020005.

[8] Caraffini, F., Kononova, A. V., Corne, D., 2019. Infeasibility and structural bias in differential evolution. `www.cse.dmu.ac.uk/~fcaraf00/BIAS/INFEASIBILITY_BIAS_DE_EXTENDED_RESULTS.pdf`.

[9] Caraffini, F., Neri, F., 2018. Rotation invariance and rotated problems: An experimental study on differential evolution. In: Sim, K., Kaufmann, P. (Eds.), Applications of Evolutionary Computation. Springer International Publishing, Cham, pp. 597–614.

[10] Caraffini, F., Neri, F., 2018. A study on rotation invariance in differential evolution. Swarm and Evolutionary Computation.

[11] Chen, Ti-Hung, Yeh, Ming-Feng, Lee, Wen-Yo, 2017. Particle swarm optimization based networked control system design with uncertainty. MATEC Web Conf. 119, 01051.

[12] Cheng, J., Zhang, G., Caraffini, F., Neri, F., 2015. Multicriteria adaptive differential evolution for global numerical optimization. Integrated Computer-Aided Engineering 22 (2), 103–107.

[13] Davarynejad, M., van den Berg, J., Rezaei, J., 2014. Evaluating center-seeking and initialization bias: The case of particle swarm and gravitational search algorithms. Information Sciences 278, 802 – 821.

[14] de Oca, M. A. M., Stützle, T., Birattari, M., Dorigo, M., 2009. Frankenstein's PSO: A composite particle swarm optimization algorithm. IEEE Transactionon on Evolutionary Computation 13 (5), 1120–1132.

[15] Eiben, A., Smit, S., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1 (1), 19 – 31.

[16] Eiben, A. E., Smith, J. E., 2003. Introduction to Evolutionary Computation. Springer-Verlag, Berlin, Germany.

[17] Grippo, L., Rinaldi, F., 2015. A class of derivative-free nonmonotone optimization algorithms employing coordinate rotations and gradient approximations. Comp. Opt. and Appl. 60, 1–33.

[18] Iacca, G., Caraffini, F., Neri, F., 2013. Memory-saving memetic computing for path-following mobile robots. Applied Soft Computing 13 (4), 2003 – 2016.

[19] Iacca, G., Caraffini, F., Neri, F., Mininno, E., April 2012. Robot base disturbance optimization with compact differential evolution light. In: et al., D. C. C. (Ed.), Applications of Evolutionary Computation. EvoApplications 2012. Lecture Notes in Computer Science, vol 7248. Springer, Berlin, Heidelberg, pp. 285–294.

[20] Inselberg, A., Aug 1985. The plane with parallel coordinates. The Visual Computer 1 (2), 69–91.

[21] Janson, S., Middendorf, M., April 2007. On trajectories of particles in pso. In: 2007 IEEE Swarm Intelligence Symposium. pp. 150–155.

[22] Kennedy, J., April 2007. Some issues and practices for particle swarms. In: IEEE Swarm Intelligence Symposium SIS 2007. pp. 162–169.

[23] Kim, H., Chong, J., Park, K., Lowther, D. A., April 2007. Differential evolution strategy for constrained global optimization and application to practical engineering problems. IEEE Transactions on Magnetics 43 (4), 1565–1568.

[24] Kononova, A. V., Corne, D. W., Wilde, P. D., Shneer, V., Caraffini, F., 2015. Structural bias in population-based algorithms. Information Sciences 298, 468–490.

[25] Lampinen, J., Zelinka, I., 2000. On stagnation of the differential evolution algorithm. In: Oŝmera, P. (Ed.), Proceedings of 6^{th} International Mendel Conference on Soft Computing. pp. 76–83.

[26] Larik, A. S., Haider, S., Nov 2016. On using evolutionary computation approach for strategy optimization in robot soccer. In: 2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI). pp. 11–16.

[27] L'Ecuyer, P., 1999. Tables of linear congruential generators of different sizes and good lattice structure. Mathematcs of computation 68 (225), 249–260.

[28] Leung, Y., Gao, Y., Xu, Z.-B., 1997. Degree of population diversity- a perspective on premature convergence in genetic algorithms and its markov chain analysis. IEEE Transactions on Neural Networks 8 (5), 1165–1176.

[29] Liu, J., Lampinen, J., Jun 2005. A fuzzy adaptive differential evolution algorithm. Soft Computing 9 (6), 448–462.

[30] Mason, K., Duggan, J., Howley, E., 2018. A meta optimisation analysis of particle swarm optimisation velocity update equations for watershed management learning. Applied Soft Computing 62, 148 – 161.

[31] Michalewicz, Z., 1995. A survey of constraint handling techniques in evolutionary computation methods. Evolutionary programming 4, 135–155.

[32] Miettinen, K. (Ed.), 1999. Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, GE. John Wiley & Sons, Inc., New York, NY, USA.

[33] Molina, D., Lozano, M., García-Martínez, C., Herrera, F., mar 2010. Memetic algorithms for continuous optimisation based on local search chains. Evolutionary Computing 18 (1), 27–63.

[34] Monson, C. K., Seppi, K. D., 2005. Exposing origin-seeking bias in PSO. In: GECCO'05. pp. 241–248.

[35] Nannen, V., Eiben, A., 2006. A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. GECCO '06. ACM, New York, NY, USA, pp. 183–190.

[36] Ong, Y., Lim, M. H., Chen, X., May 2010. Memetic computationpast, present amp; future [research frontier]. IEEE Computational Intelligence Magazine 5 (2), 24–31.

[37] Orvosh, D., Davis, L., 1993. Shall we repair? genetic algorithmscombinatorial optimizationand feasibility constraints. In: Proceedings of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 650–.

[38] Parsopoulos, K. E., 2009. Cooperative micro-differential evolution for high-dimensional problems. In: Proceedings of the conference on Genetic and evolutionary computation. pp. 531–538.

[39] Peng, F., Tang, K., Chen, G., Yao, X., 2010. Population-based algorithm portfolios for numerical optimization. IEEE Transactionon on Evolutionary Computation 14 (5), 782–800.

[40] Piotrowski, A. P., Napiorkowski, J. J., Dec 2016. Searching for structural bias in particle swarm optimization and differential evolution algorithms. Swarm Intelligence 10 (4), 307–353.

[41] Piotrowski, A. P., Napiorkowski, J. J., 2018. Some metaheuristics should be simplified. Information Sciences 427, 32 – 62.

[42] Piotrowski, A. P., Napiorkowski, J. J., 2018. Step-by-step improvement of jade and shade-based algorithms: Success or failure? Swarm and Evolutionary Computation.

[43] Poli, R., Graff, M., 2009. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (Eds.), Genetic Programming. Vol. 5481 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 195–207.

[44] Powell, M. J. D., Jan. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7 (2), 155–162.

[45] Price, K. V., Storn, R., Lampinen, J., 2005. Differential Evolution: A Practical Approach to Global Optimization. Springer.

[46] Prügel-Bennett, A., 2010. Benefits of a population: Five mechanisms that advantage population-based algorithms. IEEE Transactions on Evolutionary Computation 14 (4), 500–517.

[47] Quagliarella, D., Périaux, J., Poloni, C., Winter, G., 1997. Genetic algorithms and evolution strategies in engineering and computer science. Jhon Wiley & Sons.

[48] Rahmat-Sami, Y., Jin, N., Sept 2007. Particle swarm optimization (pso) in engineering electromagnetics: A nature-inspired evolutionary algorithm. In: 2007 International Conference on Electromagnetics in Advanced Applications. pp. 177–182.

[49] Rudolph, G., 2001. Self-adaptive mutations may lead to premature convergence. IEEE Transactions on Evolutionary Computation 5 (4), 410–414.

[50] Spall, J. C., 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control 372, 332–341.

[51] Spears, W., Green, D., Diana F., S., 11 2010. Biases in particle swarm optimization. International Journal of Swarm Intelligence Research (IJSIR) 1, 34–57.

[52] 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.

[53] Stutzle, T., Dorigo, M., Aug 2002. A short convergence proof for a class of ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 6 (4), 358–365.

[54] Wolpert, D., Macready, W., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1, 67–82.

[55] Yaman, A., Iacca, G., Caraffini, F., 2019. A comparison of three differential evolution strategies in terms of early convergence with different population sizes. AIP Conference Proceedings 2070 (1), 020002.

[56] Zaharie, D., 2002. Critical values for control parameters of differential evolution algorithm. In: Matuŝek, R., Oŝmera, P. (Eds.), Proceedings of 8th International Mendel Conference on Soft Computing. pp. 62–67.