

A Deep Dive into Effects of Structural Bias on CMA-ES Performance along Affine Trajectories

Niki van Stein¹[0000–0002–0013–7969], Sarah L. Thomson²[0000–0001–6971–7817],
and Anna V. Kononova¹[0000–0002–4138–7024]

¹ LIACS, Leiden University, Einsteinweg 55, 2333 CC Leiden, Netherlands

{n.van.stein,a.kononova}@liacs.leidenuniv.nl

² Edinburgh Napier University, United Kingdom
s.thomson4@napier.ac.uk

Abstract. To guide the design of better iterative optimisation heuristics, it is imperative to understand how inherent structural biases within algorithm components affect the performance on a wide variety of search landscapes. This study explores the impact of structural bias in the modular Covariance Matrix Adaptation Evolution Strategy (modCMA), focusing on the roles of various modulars within the algorithm. Through an extensive investigation involving 435 456 configurations of modCMA, we identified key modules that significantly influence structural bias of various classes. Our analysis utilized the Deep-BIAS toolbox for structural bias detection and classification, complemented by SHAP analysis for quantifying module contributions. The performance of these configurations was tested on a sequence of affine-recombined functions, maintaining fixed optimum locations while gradually varying the landscape features. Our results demonstrate an interplay between module-induced structural bias and algorithm performance across different landscape characteristics.

Keywords: Structural bias, benchmarking, performance analysis, algorithm behaviour

1 Introduction

In light of the rapid advancement of the field of heuristic black-box optimisation [1], a remarkable array of algorithms is now available to practitioners. The behaviour of most of these algorithms strongly depends on the settings of numerous hyperparameters, exploding the number of options further and making the choice of a well-performing algorithm configuration for a specific (real-world) problem even harder.

And yet, we still don’t understand these algorithms well enough. One thing we can do is screen (a family of) algorithms or algorithm configurations against some unwanted characteristics. Although it is unrealistic to examine all settings across all characteristics, initial efforts are essential. Such screening is expensive but it not only helps eliminate ineffective configurations but also aids in elucidating the

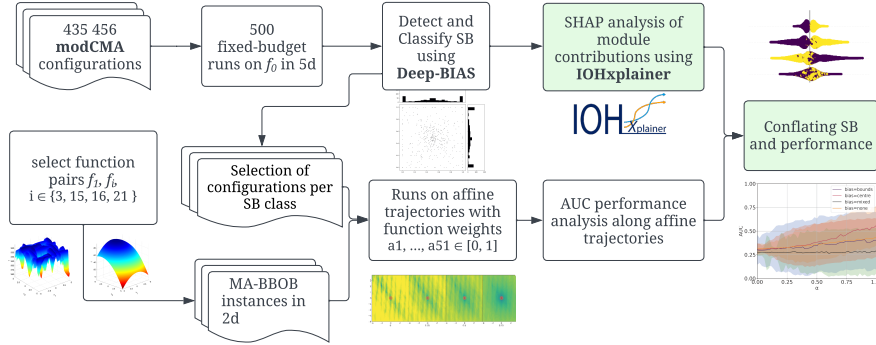


Fig. 1: The summary of the overall methodology used. Full details of all steps are provided in the corresponding sections. Green blocks highlight the contributions of this paper.

internal dynamics that impedes performance. Modular algorithm designs [2,3], where each operator option can be selected independently of the choice for other operators, are particularly well-suited for such analyses.

While in many cases, the unwanted characteristics only manifest within specific function landscapes (and the correspondence between these landscapes and characteristics can be unknown), it is hypothesised that at least some characteristics can be assessed in general. One such aspect that has not been investigated sufficiently is structural bias (SB) [4] and especially its precise causes and influence on the algorithm’s performance. SB is the algorithm’s inherent limitation in locating optima in certain regions of the domain independently of the objective function’s landscape. It stems from the iterative application of a limited set of algorithm operators and their interplay [4]. While some families of algorithms have been screened to some extent [5,2], no clear performance implications have been established so far. Unfortunately, even though such screening is done for very large algorithm configuration spaces, it necessarily remains limited due to the need to discretise numerous continuous hyperparameters, thus potentially overlooking some interactions. This paper is no exception, however, its experimental design is structured to be as comprehensive as computationally feasible.

This paper aims to explore the impact of SB on algorithm performance by addressing the following questions: 1. How does the performance of structurally biased versus unbiased configurations change on sequences of functions where the landscape progressively shifts from rugged to smooth? 2. How does the location of the optima within the domain ¹ of these functions affect the performance depending on the class of structural bias? The complete methodology of our investigation is summarised in Figure 1.

¹ This paper focuses on box-constrained minimisation problems.

2 Background

2.1 Modular CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [6] is a robust, state-of-the-art evolutionary algorithm used for solving non-linear, non-convex optimisation problems [1]. Central to its approach is the adaptation of a covariance matrix which determines the shape and scale of the search distribution, effectively learning the landscape of the problem space. This self-adaptive mechanism allows the algorithm to balance exploration of the search space with exploitation of known good regions, making it particularly effective in a wide range of practical applications, from machine learning parameter tuning to engineering design optimisation. The Modular CMA-ES (modCMA) [3] is a Python and C++ modular implementation of the CMA-ES algorithm and many of its variants, with module options and hyper-parameters that can be switched on and off independently of each other. In this work we investigate the full scale of these module options, given in Table 1, leading to a total of 435 456 different CMA-ES configurations.

2.2 Structural bias

Structural bias in iterative optimisation heuristics [4,7] refers to the tendency of certain algorithms or configurations of algorithms to favour specific regions of the search space over others, despite the absence of initial information indicating where high-quality solutions might reside. Ideally, an optimisation algorithm should explore the defined domain boundaries without preconceived preferences, allowing the data collected from sampled points to guide its progression towards areas with optimal objective function values. However, in practice, some algorithms inherently exhibit a preference, such as a bias towards the centre of the

Table 1: Considered modules of **modCMA** and their configurations.

Module name	Shorthand	Domain
Covariance adaptation	<code>covariance</code>	{false, true}
Elitism	<code>elitist</code>	{false, true}
Active update	<code>active</code>	{false, true}
Base sampler	<code>base_sampler</code>	{Gaussian, Halton, Sobol}
Orthogonal sampling	<code>orthogonal</code>	{false, true}
Threshold convergence	<code>threshold</code>	{false, true}
Sample Sigma	<code>sigma</code>	{false, true}
Bound correction	<code>bound_correction</code>	{off, saturate, mirror, COTN, toroidal, uniform}
Mirrored sampling	<code>mirrored</code>	{off, mirrored, mirrored pairwise}
Recombination weights	<code>weights_option</code>	{default, equal, λ -decay}
Step size adaptation	<code>step_size_adaptation</code>	{CSA, PSR, TPA, MSR, XNES, MXNES, LPXNES}
Local restarts	<code>local_restart</code>	{none, IPOP, BIPOP}

domain, which can limit their effectiveness in universally discovering the best solutions across the entire feasible domain. This phenomenon, known as *structural bias*, can compromise the algorithm’s ability to perform well in general situations. Detecting structural bias is hard, as the objective function and behaviour of the algorithm are always entangled. Using a special objective function (f_0), defined as a completely random fitness landscape, allows one to detect structural bias using statistical tests as introduced in [8]. There are a few related works on Structural Bias that either introduced a different detection method, like the generalised signature test [9], or analysed different groups of algorithms regarding SB [10,11,12].

2.3 SHAP

Shapley Additive Explanations (SHAP), as introduced by Lundberg et al. [13], is a popular *explainable AI* (XAI) method for attributing features in model predictions. SHAP quantifies the impact of a specific feature, f , by comparing model outputs with and without f . The difference in outputs, averaged across models, defines the SHAP value, which can be positive, negative, or zero, representing the feature’s marginal contribution.

However, SHAP’s application to large datasets is computationally intensive. To address this, the TreeSHAP method [14] employs tree-based model structures to streamline computations, using approximation techniques to enhance efficiency in scenarios with extensive feature sets. In this work, we use this XAI method to compute the contributions of different module settings towards specific structural bias classes. This is done by training Xg-boost regression models on the one-hot-encoded algorithm configurations as input and the predicted SB class label from Deep-BIAS as output. Using these models we can approximate the SHAP values of each module option per configuration.

3 Structural Bias Classification

To assess the structural bias (SB) and in the end the interplay of structural bias with performance depending on landscape features and the location of the optima, the first step is to detect and classify structural bias per algorithm configuration. We conduct a configuration sweep for modCMA [3] using the MODCMA package in Python. In this extensive analysis, we use a full grid of all of the categorical module options in modCMA, as specified in Table 1. This resulted in a total of 435 456 configurations. For each of these configurations the population sizes are fixed to $\mu = 5$, $\lambda = 20$. The analysis in this paper is broader and more in-depth than previous analysis of structural bias for modCMA [12] in the sense that it contains not just a subset of modCMA module options but the complete set of all categorical options (and a limited set of continues parameters). In addition, in this work we propose an explainable AI approach, similar to the approach used in [5], to analyze the different contributions of different module options to structural bias and to specific types of structural bias, leading to new insights.

In [5], the XAI approaches were used to analyse the contributions of modules and hyper-parameters on the performance on different function landscapes, here we use it to assess the influence of modules on structural bias, and differently from the approach in [5] we one-hot-encode all categorical module options to see how each option affects the structural bias individually. We also used the approach to look at second order interactions in relation with structural bias, however, these second order interactions were marginal and we therefore do not include these results in this work.

3.1 Methodology

For the assessment and classification of SB, we used the BIAS toolbox [8], available on [15]. The toolbox provides an SB detection mechanism based on the aggregation of the results of 39 statistical tests but also a Deep-learning approach [16] to detect and classify SB based on distributions of final points (found minima) of many independent runs on f_0 . The three bias types we are looking at in this work are: SB towards the **centre** of the search space, towards the **bounds**, and **uniform** (no SB detected). SB is detected by first running an optimisation algorithm several times on the random objective function f_0 . Here we used 100 independent runs with 10.000 function evaluations as budget per run to make the first classification of SB using the Deep-BIAS toolbox. Due to its speed and classification accuracy, we leveraged the Deep-BIAS model instead of the statistical methods in the BIAS toolbox. The Deep-BIAS method classifies the distribution of (in this case 100) final best points found by the algorithm. We do have to note that the Deep-learning model is not a perfect predictor. We therefore also verify the top 20 configurations per SB class, used later in our experiments, by visual inspection of the final point distributions.

Once we have classified each of the configurations automatically, we can use the confidence of each SB class to calculate approximate Shapley values using the TreeSHAP algorithm [14]. The calculated SHAP values for each module option per structural bias class are shown in Figure 2. We can use these SHAP values to gain insights into which module options contribute to what kind of structural bias.

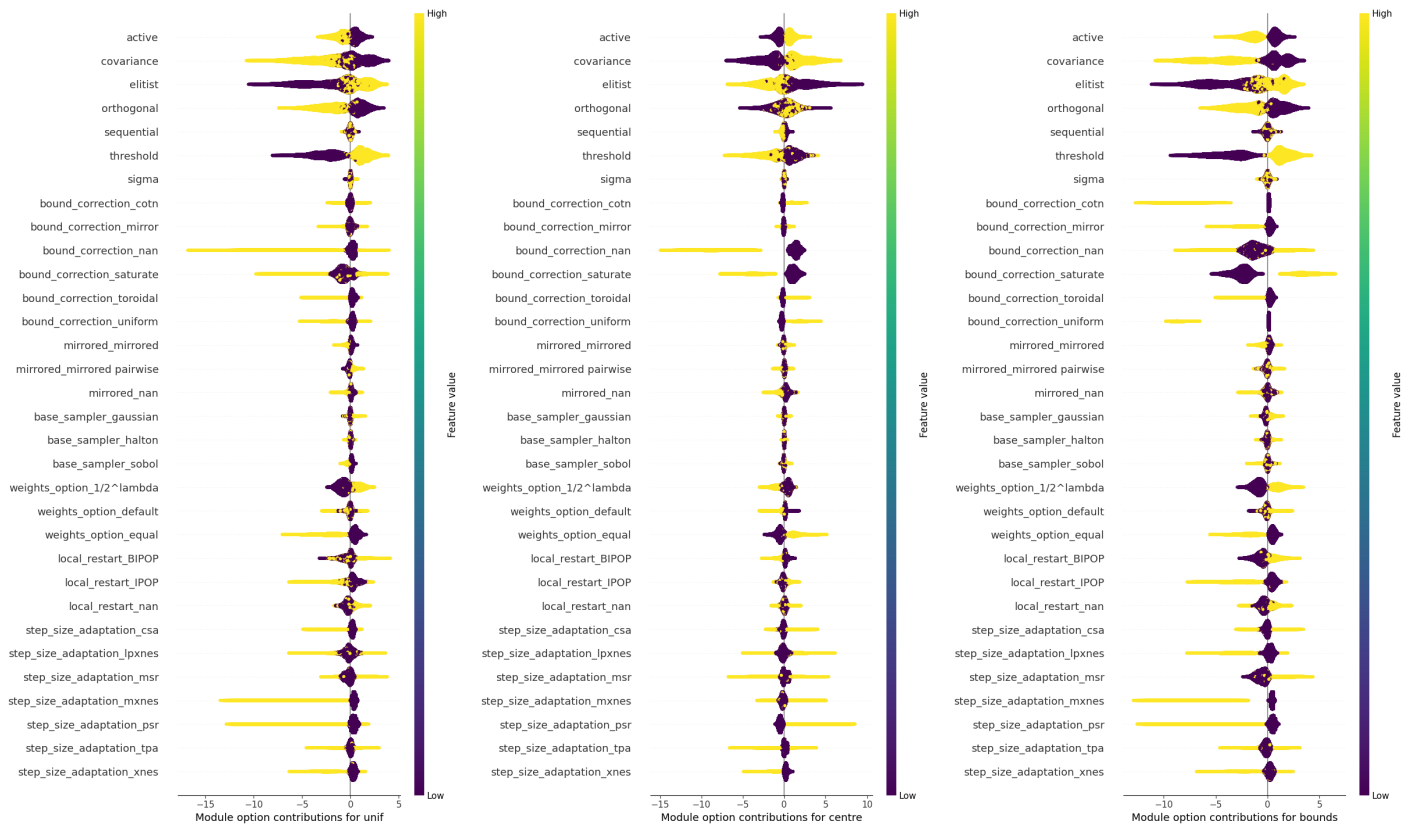


Fig. 2: SHAP values showing module contributions to (from left to right) no structural bias, centre bias and bounds bias classes, respectively. The baseline prediction of these classes are 0.094, 0.559, 0.054 respectively, meaning that a SHAP value of 0 would result in the given baseline class probability.

3.2 Module contributions to SB

Based on the output of the Deep-BIAS package, most of the considered configurations of modCMA (82%) are classified as **Centre** biased, 9% as unbiased (uniform) and 5% as biased towards the bounds. The remaining fraction (3%) was classified as discretization bias, however after visual inspection, those configurations were mostly misclassified and should be either centre or unbiased and therefore we did not take them into account.

Given the SHAP values from Figure 2 and taking into account the base value (mean classification confidence of each class), we see a few interesting patterns. Overall, the covariance, elitism, threshold, bound correction and step size adaptation modules mostly influence the structural bias classification. We can also observe that in general, option contributions are negatively correlated between centre SB and bounds SB, in other words, when a module option causes centre SB it lowers the probability of bounds SB and vice versa. Bounds SB and uniform (no SB) seem roughly aligned except for the bound correction methods. Let us discuss the major modules involved in centre, bounds and unbiased below.

Elitism when turned on, reduces centre SB according to the SHAP data. Since centre SB is the majority class, Elitism seems to reduce SB in general. When looking at the inner workings of modCMA and also the objective function f_0 , this could be explained due to the fact that with elitism the algorithm is more likely to get stuck in a (local) minimum on f_0 early in the optimisation run, effectively dampening the structural bias effects. Elitism by itself is however very likely not to be responsible for any structural biased behaviour.

Threshold convergence when turned on has a similar effect as elitism (though less profound). Again, threshold convergence is likely not causing any biased algorithm behaviour but amplifies (when turned off) or dampens (when turned on) the SB effects.

Bound correction saturate shows to have a large effect on bounds SB, which makes perfect sense and is in line with other research on structural bias [12]. Upon close inspection, all configurations that were classified with high confidence as bounds SB (confidence > 0.45), all used Saturate as the bound correction method.

Covariance matrix adaptation seems to play a large role in centre SB. In the context of a standard CMA-ES (so with the covariance module on), the search distribution is represented by a multivariate normal distribution. This distribution is characterized by its covariance matrix, which determines the shape and orientation of the points (solutions) that are sampled. Geometrically, the shape of this distribution resembles a hyper-ellipse. The search space, on the other hand, is typically a hyper-cube. This causes a mismatch in shapes being explored, likely leading to a structural bias towards the centre of the search space. The hyper-ellipse will naturally avoid sampling close to the edges and especially the corners of the hyper-cube because these areas are outside the maximum reach of the distribution whose radius is limited to the smaller distance from the center to an edge, rather than to a corner. This effect is amplified as the dimensionality of the space increases. In higher dimensions, the corners of the

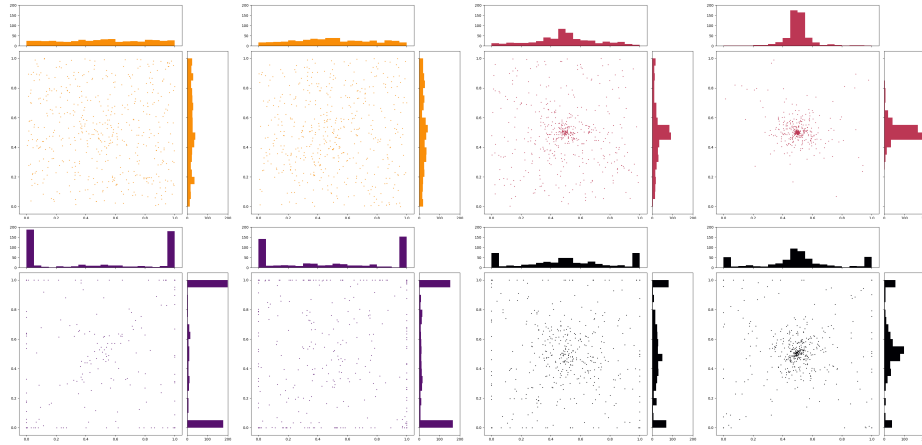


Fig. 3: Examples of the final best point distributions from CMA-ES configurations run on the SB test function f_0 in 2D from 500 independent runs (using different random seed) of configurations (re)classified as **no**, **centre**, **bounds** and **mixed** SB.

hyper-cube are exponentially further away from the centre compared to the edges. Thus, a spherical sampling distribution centred in the hyper-cube will leave vast regions in the corners significantly undersampled. This results in a higher concentration of sample points towards the centre of the search space, and relatively fewer near the boundaries and corners. It could potentially lead to suboptimal exploration of the search space, especially if the global optimum lies near the boundaries or corners of the domain.

Other module options seem to have a limited or mixed effect on structural bias in modCMA.

3.3 Limitations of Deep-BIAS and Mixed SB

While the deep-learning approach of the Deep-BIAS toolbox is very fast, and therefore allows the evaluation of 400 000+ configurations, it is not perfect. First of all, it is known [16] that the SB type ‘Clusters’ is often a misclassified ‘Centre’ SB. As a result of this, after visual inspection of (a large fraction) of the configurations that were initially classified here as Cluster SB, we found out that all of those actually belonged to the Centre class. We therefore discarded the ‘Clusters’ class in our analysis. In addition, after visual inspection of the configurations with highest confidence scores for each of the SB classes, we discovered a *mixed* class between centre and bounds bias. This mix of two bias directions was not discovered in earlier structural bias research and was also not taken into account when developing the (Deep)-BIAS Toolbox. For modCMA, this mixed SB behaviour seems to occur when there is a configuration that is normally centre biased and it also uses the Saturate bound correction method (induc-

ing additional bounds SB). For further analysis of the effects of these different SB classes on performance, we decided to re-classify the top 20 configurations (sorted by confidence) for each SB class as identified by Deep-BIAS by visual inspection of the 2D final distributions into four SB classes: Uniform (no SB), Centre, Bounds and Mixed SB. See Figure 3 for examples of each class of SB we took into consideration. The complete set of configurations and distributions of their final best points across runs can be found in the supplemental material [17].

4 Effects of Structural Bias on Performance

To analyse the effect of SB on algorithm performance, four distinct SB groups of algorithm configurations are evaluated on a range of affine function combinations. By gradually changing the function landscape properties, it is evaluated how the performance of these different groups changes under different conditions.

4.1 Affine function pairs

We include as original problems the SPHERE function (*f1* from BBOB, uni-modal) and four other BBOB functions: *f3* (separable RASTRIGN, multi-modal), *f15* (non-separable Rastrign, multi-modal), *f16* (WEIERSTRASS, multi-modal, adequate global structure), and *f21* (Gallagher’s Gaussian 101-me PEAKS Function, multi-modal, weak global structure). All of these are visualised in 2D in Figure 1 of the supplemental material. The sphere function was chosen because we would like to *tune flatness into the affine combinations*, and the other four were chosen after visual inspection of their ruggedness (we would like to track structural bias along affine trajectories which begin at the flatness of the sphere function and *gradually become more rugged*).

We consider affine combinations between BBOB functions and use the generator proposed by Vermetten *et al.* [18] which facilitates combinations of more than two functions — although we consider only pairs here. Their generator takes three objects as input in order to construct a function: 1. the desired location for the optimum, $\vec{X}_{opt}$; 2. a vector of length 24 — for each of the 24 BBOB functions — indicating proportions, $\vec{W}$; and 3. a vector of length 24 indicating which instances of the BBOB functions should be used, $\vec{I}$. Of course, to obtain pairwise combinations then the proportions of 22 functions can be set to zero and the remaining two have non-zero weight. An affine combination Ξ is constructed by the generator according to the fitness *scaling functions*:

$$R_i(x) = \frac{\max(\log_{10}(x), -8) + 8}{S_i} \quad (1)$$

and its inverse (to reverse back to the original fitness scale):

$$R_i^{-1}(x) = 10^{(S_i \cdot x - 8)} \quad (2)$$

S_i is a scale factor and is set at literature-recommended values [18] depending on the base function: 11.0 for *f1*, 12.3 for *f3* and *f15*, 10.3 for *f16*, and 10.7 for *f21*.

With these defined, we can formally state that Ξ can be obtained by the generator as such:

$$\Xi(\vec{W}, \vec{I}, \vec{X}_{opt}) = R^{-1}(\sum_{i=1}^{24} W_i \cdot R_i(f_1, I_1(x - \vec{X}_{opt} + O_i, I_i) - f_i, I_i(O_i, I_i))) \quad (3)$$

where f_i, I_i is instance i of original function f_i and O_i, I_i is the location of the optimum for instance i of function f_i .

4.2 Experimental setup

We access the 24 noiseless BBOB functions in $2D$ through IOHEXPERIMENTER [19].

Affine combinations The original BBOB functions involved in the affine pairs are all $2D$, to facilitate visualisation. We consider the region of interest $[-4, 4]$ per dimension only. For each function pair, a sequence of 51 values $\alpha \in [0, 1]$ is defined equally spaced with a step of 0.02. We define α as the *proportion of the SPHERE function*, which is BBOB *f1*. For each combination of two functions with a given alpha, we generate four affine combinations which differ only in the location of the global optimum — we use the same instances of the BBOB constituent parts for each of them. We also keep the instance number and optimum location consistent across increasing α within each combination of base function pair and optimum placement strategy. Instances are randomly generated between 1 and 100. The four placement strategies for the optimum are:

1. near to the boundary (within 0.01) in both coordinates,
2. near the centre (between $[-0.01, 0.01]$ in both coordinates),
3. near to the boundary in one coordinate and central in the other,
4. located randomly for both coordinates between $[-2, 2]$.

In total, we generate 816 affine recombination functions (4 original function pairs $\times$ 51 affine weights $\times$ 4 locations for the optimum). The process of affine combination and placement of the optimum is conducted using functions from the IOHEXPERIMENTER package in Python.

Algorithm performance For assessing algorithm performance on the 816 functions we consider the top-scoring modCMA configurations for each bias type after careful visual inspection of the SB distributions (See Section 3.3) (**bounds** (20 configurations), **centre** (19), **mixed** (7), and **none** (10)). In total, this amounts to 56 CMAES variants and 45 696 algorithm and function-pairs. The algorithm configurations for these are available in Tables 1-4 of the supplemental material [17]. Each CMA-ES configuration is instantiated in MODCMA, provided a budget of 5000 evaluations, and is executed 30 times on each of the 816 affine functions. As the performance metric, we use a normalised area under the curve (AUC) with respect to the empirical cumulative distribution function, implemented by Vermetten *et al*². The distribution function considers the default COCO settings with 51 targets spaced logarithmically beginning at 10^{-8} and terminating at 10^2 .

² <https://zenodo.org/records/10376912>

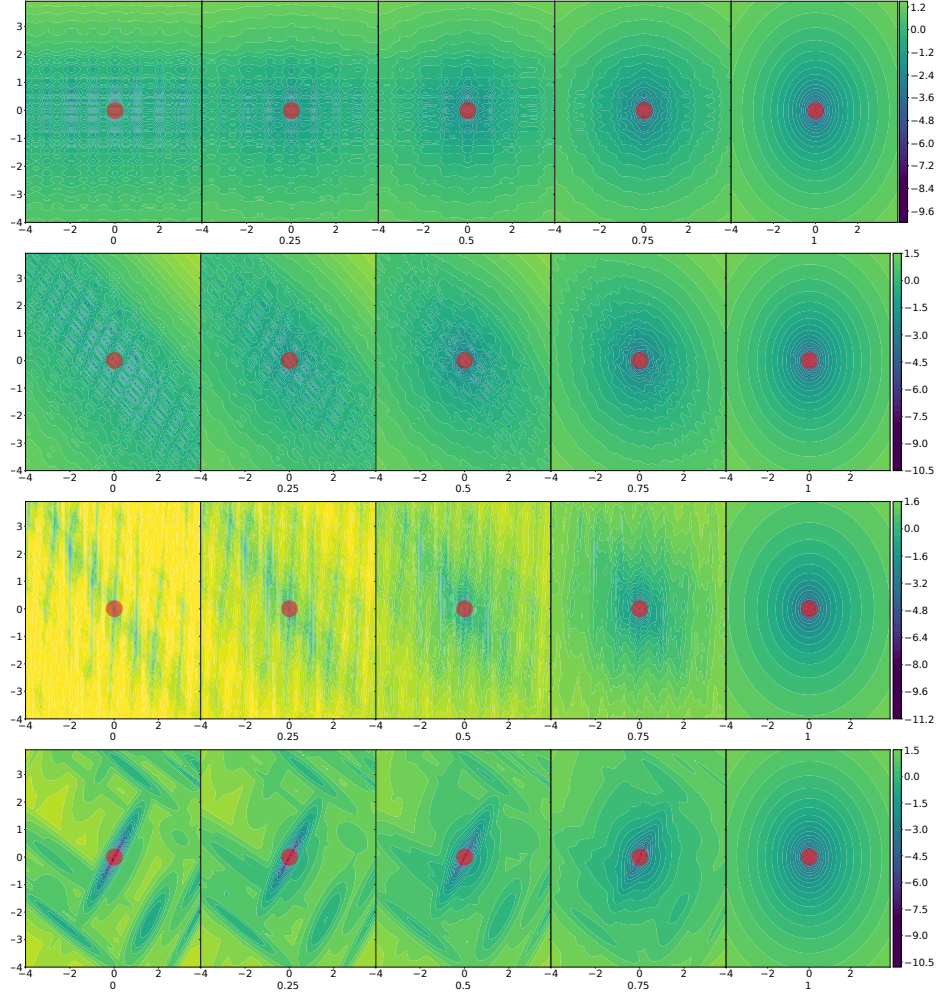


Fig. 4: Example 2D landscapes for affine combinations of functions $f3$, $f15$, $f16$, $f21$ (top to bottom) with $f1$ for 5 affine weights α shown as labels below individual plots, where increasing α corresponds to increasing the proportion of $f1$. On the instances shown here, the location of the global optimum is fixed near the centre of the domain and marked in red.

5 Results

Figure 4 presents, for different base BBOB function pairings, states of affine combination for increasing α (left to right), which is the proportion of the Sphere function $f1$. Colour represents fitness and the global optimum is placed here near the centre.

Notice from the left-most plots that with $\alpha = 0$, the function contains no aspect of $f1$. For Figure 4 rows 1 and 2, their intermediate stages ($0.25 \leq \alpha \leq 0.75$) show

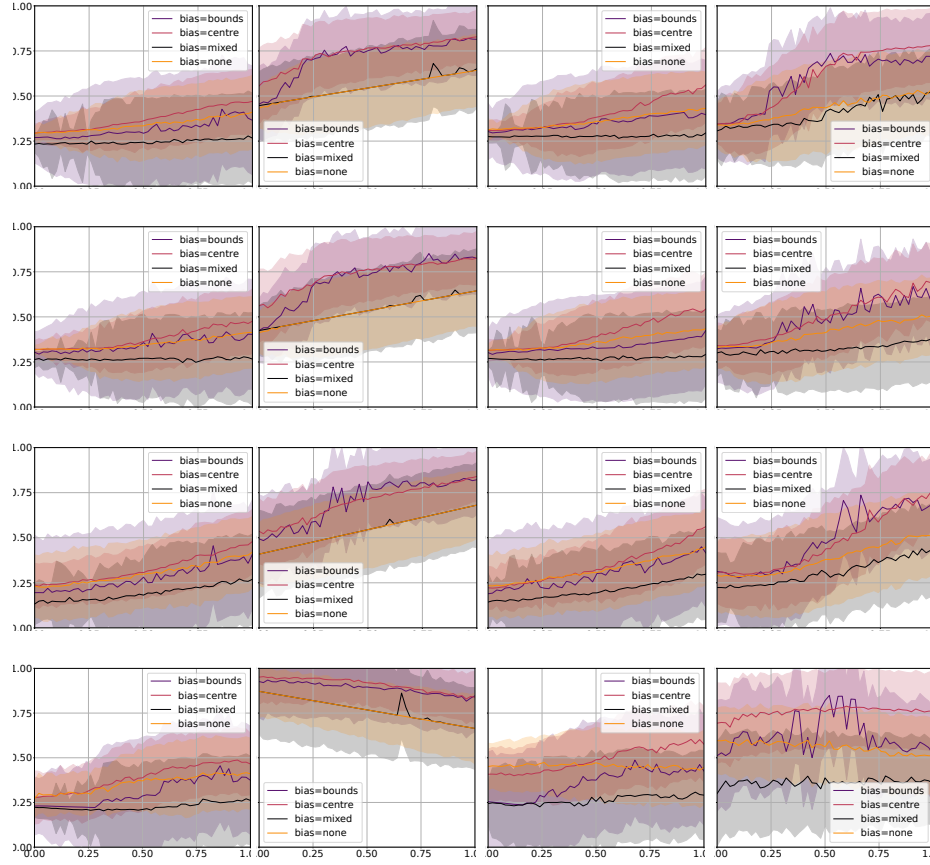


Fig. 5: Median and variance for AUC over 30 runs for CMA-ES variants split into bias classes across increasing α for optimum placements (left to right) bounds, centre, centre of bounds, and random — and for base BBOB function pairs (rows, top to bottom): $f3-f1$; $f15-f1$; $f16-f1$; and $f21-f1$.

the influence of the ‘roundness’ from $f1$ being added into the function. In the case of Figure 4 rows 3 and 4, the effect of increasing α manifests differently, with the bowl shape and concentric structure of $f1$ beginning to appear. We can see that when $\alpha = 1$, the function is entirely $f1$.

Figure 5 presents algorithm performance (AUC median and variance) over 30 runs for the top-scoring CMA-ES variants in each bias class across increasing α (shown on the horizontal axis), split by location of the optimum (left to right: bounds, centre, centre of bounds, and random) and for base function pairs (top to bottom): $f3-f1$, $f15-f1$, $f16-f1$, and $f21-f1$.

The first three rows (that is, the first three function pairs) show similar patterns and trends. Generally, increasing α (proportion of $f1$) is associated with increasing performance. Notice by comparing the median lines of, for example, the second plot of

the first row with the other three plots in the same row that placing the optimum at the centre leads to the best performances by the algorithms, regardless of bias class. Algorithms perform at their worst when the optimum is placed at the bounds in at least one of the two co-ordinates (see the first and third columns of plots).

We now consider how algorithms from the different bias classes compare. Observe from the first and third plot in the first three rows that the *centre* and *none* classes of algorithms perform best (in that order) when the optimum is located near the bounds of the function. This is a curious result: intuitively, the *bounds* algorithms would do the best. We checked some of the algorithm runs and noticed that *centre* algorithms appear to have more freedom of movement — making several small improvements in fitness. On the other hand, *bounds* algorithms seem to sometimes get stuck in these cases when the optimum is on the bounds — struggling to find a fitness improvement and advance towards the optimum location. We leave a statistical analysis of this phenomenon for future study. In the cases where the optimum is placed either centrally or randomly, the best-performing classes of algorithms are *bounds* and *centre* (notice the second and fourth plots in the first three rows). While it makes sense that *centre* algorithms would perform best on these, the high performance of *bounds* is less intuitive. From examination of a sample of performance runs, it seems that although *bounds* algorithms may begin the search as biased towards the bounds, in the specific case where the optimum is centrally located they seem to be able to navigate towards the centre over the course of the search. It seems that the performance of *bounds*-biased algorithms depends on the location of the optimum, but not in the way which might be expected: if the optimum is at the bounds, they may struggle; if it is at the centre, they do better. Note from the Figures that overall, the *mixed* class of algorithms is the worst performing.

The last function pairing, shown in the last row, differs from the other pairings substantially when the optimum is placed centrally (second column). We notice that performance is excellent (with AUC near to 1 in some cases) and that the trend with respect to α has reversed: increasing α is here associated with a decrease in performance. The explanation for this finding can probably be found in the nature of the original base BBOB function *f21*, where the global optimum can be found near the bounds at the bottom of a half-funnel shape. Observe from the lowest-left plot in Figure 4 that when the optimum is placed in the centre, this appears to stretch and mirror the half-funnel structure which leads to the optimum. We therefore speculate that this stretched funnel surrounding the optimum (in the case when it is centrally placed) is the reason for high algorithmic performance.

6 Conclusions

This study has systematically explored the interplay between the performance of configurations of the modular Covariance Matrix Adaptation Evolution Strategy (modCMA) and structural bias within different optimisation landscapes. Through the extensive configuration testing of modCMA, encompassing 435 456 configurations, we have shown that specific modules notably influence the algorithm’s structural bias. Key insights include the significant impact of modules like covariance adaptation and elitism in modulating structural bias towards the centre of the search space and bound correction method Saturate towards the boundaries of the search space.

To investigate the effects of different forms of SB on algorithm performance, we generated pairwise affine recombinations of BBOB functions with varying proportions

of each composite function. For each function we considered four strategies for placing the optimum. The configurations with highest confidence per SB class (predicted by the Deep-BIAS tool) of modCMA, were run 30 times on the affine-combined functions. The results showed that when the optimum is placed at the centre, bounds-biased and centre-biased algorithms perform best. The reason for this is likely that when the optimum is near the centre, bounds-biased algorithms can navigate in the right direction even if they have an inherent bias towards the bounds — the bounds structural bias is mainly caused by the bounds correction method *saturate*, which does not often become active when the search leads away from the bounds. When the optimum is near the bounds, centre-biased and unbiased algorithms are performing better. We believe that centre-biased algorithms have more freedom of movement, and that bounds-biased algorithms with *saturate* bound correction method become early stuck when the optimum is at the bounds.

Future research should focus on extending the analysis of structural bias effects into higher dimensional spaces. As dimensionality increases, the complex interplay between geometry of high-dimensional spaces, structural bias and landscape features may exhibit different characteristics that could bring additional insights. This future work will not only deepen our understanding of structural bias in iterative algorithms but also guide the development of more robust strategies for tackling complex optimisation problems.

References

1. Bäck, T.H.W., Kononova, A.V., van Stein, B., Wang, H., Antonov, K.A., Kalkreuth, R.T., de Nobel, J., Vermetten, D., de Winter, R., Ye, F.: Evolutionary Algorithms for Parameter Optimization—Thirty Years Later. *Evolutionary Computation* 31(2), 81–122 (06 2023)
2. Vermetten, D., Caraffini, F., Kononova, A.V., Bäck, T.: Modular differential evolution. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 864–872. GECCO '23, Association for Computing Machinery, New York, NY, USA (2023)
3. de Nobel, J., Vermetten, D., Wang, H., Doerr, C., Bäck, T.: Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. pp. 1375–1384 (2021)
4. Kononova, A.V., Corne, D.W., Wilde, P.D., Shneer, V., Caraffini, F.: Structural bias in population-based algorithms. *Information Sciences* 298, 468–490 (2015)
5. van Stein, N., Vermetten, D., Kononova, A.V., Bäck, T.: Explainable benchmarking for iterative optimization heuristics (2024)
6. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9(2), 159–195 (2001)
7. Davarynejad, M., van den Berg, J., Rezaei, J.: Evaluating center-seeking and initialization bias: The case of particle swarm and gravitational search algorithms. *Information Sciences* 278, 802–821 (2014)
8. Vermetten, D., van Stein, B., Caraffini, F., Minku, L.L., Kononova, A.V.: BIAS: A toolbox for benchmarking structural bias in the continuous domain. *IEEE Transactions on Evolutionary Computation* 26(6), 1380–1393 (2022)
9. Rajwar, K., Deep, K.: Uncovering structural bias in population-based optimization algorithms: A theoretical and simulation-based analysis of the generalized signature test. *Expert Systems with Applications* 240, 122332 (2024)

10. Kononova, A.V., Caraffini, F., Wang, H., Bäck, T.: Can compact optimisation algorithms be structurally biased? In: *Parallel Problem Solving from Nature – PPSN XVI*. pp. 229–242. Springer International Publishing, Cham (2020)
11. Vermetten, D., van Stein, B., Kononova, A.V., Caraffini, F.: Analysis of Structural Bias in Differential Evolution Configurations, pp. 1–22. Springer Nature Singapore, Singapore (2022)
12. Vermetten, D., Caraffini, F., van Stein, B., Kononova, A.V.: Using structural bias to analyse the behaviour of modular CMA-ES. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. p. 1674–1682. GECCO ’22, Association for Computing Machinery, New York, NY, USA (2022), <https://doi.org/10.1145/3520304.3534035>
13. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. *Advances in Neural information processing systems* 30 (2017)
14. Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.I.: From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence* 2(1), 2522–5839 (2020)
15. van Stein, B., Vermetten, D., Caraffini, F., Kononova V, A.: Deep-bias v1.0.0 (Jan 2023), <https://doi.org/10.5281/zenodo.7614586>
16. Van Stein, B., Vermetten, D., Caraffini, F., Kononova, A.V.: Deep bias: Detecting structural bias using explainable ai. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. pp. 455–458 (2023)
17. van Stein, N., Thomson, S., Kononova, A.V.: Supplemental Material for ”A Deep Dive into Effects of Structural Bias on CMA-ES Performance along Affine Trajectories” (Apr 2024), <https://doi.org/10.5281/zenodo.10994149>
18. Vermetten, D., Ye, F., Bäck, T., Doerr, C.: Ma-bbob: A problem generator for black-box optimization using affine combinations and shifts. *arXiv preprint arXiv:2312.11083* (2023)
19. de Nobel, J., Ye, F., Vermetten, D., Wang, H., Doerr, C., Bäck, T.: Iohexperi-
menter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation* pp. 1–6 (2024)