

From benchmarking optimization heuristics to dynamic algorithm configuration

Vermetten, D.L.

Citation
Vermetten, D. L. (2025, February 13). *From benchmarking optimization heuristics to dynamic algorithm configuration*. Retrieved from https://hdl.handle.net/1887/4180395

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4180395

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

From Benchmarking Optimization Heuristics to Dynamic Algorithm Configuration

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op donderdag 13 februari 2025
klokke 11:30 uur

door

Diederick Lambertus Vermetten
geboren te Maastricht, Nederland
in 1996

Promotor:
Prof.dr. T.H.W. Bäck

Co-promotores:
Dr.-ing. HDR C. Doerr (CNRS and Sorbonne Université, France)
Dr. H. Wang

Promotiecommissie:
Prof.dr. M.M. Bonsangue
Prof.dr. A. Plaat
Dr. J.N. van Rijn
Dr. A.V. Kononova
Prof.dr. K. Smith-Miles (University of Melbourne, Australia)
Prof.dr. J. Branke (University of Warwick, United Kingdom)

Copyright © 2025

ISBN:

Het onderzoek beschreven in dit proefschrift is uitgevoerd aan het Leiden Institute of Advanced Computer Science (LIACS).

"He said that most men were in their lives like the carpenter whose work went so slowly for the dullness of his tools that he had not time to sharpen them."

- Cormac McCarthy, 'The Crossing'

Contents

1 Introduction **1**

1.1 Research Questions 2

1.2 Other Contributions by the Author 3

1.3 Thesis Outline 4

2 Preliminaries **5**

2.1 Iterative Optimization Heuristics 5

2.2 Benchmark Problems: BBOB 6

2.3 Core Algorithms 8

2.4 Performance Indicators 11

2.5 Exploratory Landscape Analysis 16

2.6 Reproducibility 17

3 Benchmarking Optimization Algorithms **19**

3.1 IOHprofiler 20

3.2 Benchmark Suites: BBOB 43

3.3 Benchmarking Algorithm Behavior: Structural Bias 53

4 Algorithm Configuration and Selection **69**

4.1 Modular Algorithm Design 71

4.2 Algorithm Configuration for Modular Algorithms 80

4.3 Selected Challenges in Algorithm Configuration 94

5 Dynamic Algorithm Configuration and Selection **109**

5.1 Complementarity in Anytime Performance 110

5.2 Switching Between Algorithm Variants 118

5.3 Per-run Dynamic Algorithm Selection 122

5.4 When to Switch? 132

6 Testing Generalizability: MA-BBOB **145**
6.1 Many-Affine BBOB 146
6.2 Pairwise Affine Combinations 150
6.3 Combining Multiple Functions: Testing Generalizability 160

7 Conclusions **163**
7.1 Key Findings 165
7.2 Future Work 167

Bibliography **171**

Acronyms **197**

Samenvatting **199**

Summary **201**

Acknowledgements **203**

Curriculum Vitae **205**

Publications **207**

Chapter 1

Introduction

Optimization problems can be found in a wide variety of contexts, from complex industrial problems where we aim to increase the crashworthiness of a car body to daily life where we want to choose the shortest route home. While we are always looking for better solutions, it is usually impossible to try every possible option. Instead of an exhaustive evaluation, we can introduce a heuristic search procedure to efficiently explore the solution space. At the cost of potentially missing the 'perfect' solution, these types of methods prove to be rather effective in practice.

Because of the prevalence and variety of optimization problems, heuristic optimization algorithms are continuously being developed to solve subsets of problems more effectively than before. This development has led to a thriving research area, and the resulting algorithms have significantly benefited a wide range of real-world applications.

The variety of available optimization heuristics is a clear indication of the strengths of this class of algorithms. However, with so many methods to choose from, it can be hard to gauge which algorithms excel in a given setting. In order to support the continued development and understanding of heuristic optimization algorithms, standardized practices for testing and comparing their results are a necessity. *Benchmarking tools* support this aim by providing access to sets of problems with known properties and fixed pipelines for logging the optimization process. The resulting data can then be compared fairly to data from existing algorithms in a variety of ways, including aggregated visualizations and statistical testing procedures.

In this thesis, we will focus on the IOHprofiler benchmarking environment, and show how this tool supports robust research in a variety of manners. In particular,

we highlight the ways in which we can use benchmarking in combination with the modularization of algorithms to gain insight into the benefits of individual algorithmic components, as well as shine a light on the interplay between pairs of components. This shows the inherent complementarity between different algorithmic principles, where some combinations of components outperform others on one type of problem, while the reverse is true for problems with different overarching properties. This notion of complementarity between algorithms lies at the core of *algorithm selection*, where we aim to first identify some functional properties of the problems to be optimized, to then determine which algorithm would be most appropriate to use for the optimization.

By combining the ideas of algorithm selection with the detailed insights gained from tracking the full performance trajectory during benchmarking, we finally get into the domain of *dynamic algorithm selection*. Here, we don't merely exploit differences between algorithms on different types of problems, but instead, we aim to utilize the information about the local state of the optimization algorithm to decide whether to continue optimizing the same way or to switch to a completely different algorithm. Depending on the way in which this problem is framed, we can also consider the combination with *algorithm configuration*, where we tune the parameters of the optimization algorithm, leading to an even larger space of potential choices and corresponding benefits.

1.1 Research Questions

The overarching question of this thesis is: *How can benchmark data be used to gain insights into, and subsequently exploit, different levels of algorithm complementarity?* Since this question encompasses a wide range of topics, we further specify a set of underlying research questions which shine a light on a selected set of components ranging from benchmarking software to dynamic algorithm selection.

RQ1 *How can robust benchmarking pipelines be made accessible and resulting data be made usable by the wider community?* This is the core focus of Chapter 3 (based on [255, 53, 177, 43, 136, 150, 244, 236]), where we introduce *IOHprofiler*, a benchmarking environment with a modular design which aims to lower the barrier of entry to robust experimental design. By examining the wide range of problems made available in IOHprofiler, we show the way in which benchmark design influences the types of conclusions which can be drawn from studies which use these types of problems. In addition to the common performance-oriented

benchmarks, we also discuss *behavior-based benchmarking*, illustrating the ways in which this difference of perspective can change the ways in which we look at the differences between iterative optimization heuristics.

RQ2 *How can a modular design aid in the exploration of interactions between different algorithmic ideas?* Most new algorithmic ideas are not proposed in isolation, but instead build upon existing algorithm families. While these modifications are usually compared to a baseline implementation of the original algorithm, their potential interactions with other modifications are usually hard to judge. In Chapter 4 (based on [51, 240, 247, 246]), we explore how the usage of modular design principles can be combined with algorithm configuration techniques to explore the strengths of different modifications within an algorithm family.

RQ3 *To what extent can we exploit performance complementarity between different algorithms by switching between them?* While performance complementarity is classically exploited by algorithm selection or algorithm configuration techniques, we expand these ideas to account for the potential benefit of switching between algorithms *during the search process*. By switching algorithms in this way, we can potentially speed up convergence by combining algorithms' strengths in different phases of the optimization. In Chapter 5 (based on [245, 243, 110, 135, 248]), we explore this potential from a data-driven perspective, as well as by performing switching both within a fixed algorithm family and between completely different optimizers.

RQ4 *How can we fairly judge the performance of meta-learning methods in the context of black-box optimization?* While the previously discussed research questions focus on methods for exploiting algorithm complementarity, it is important to note how the efficacy of these meta-learning techniques depends on the used benchmark suites. In particular, the generalizability of results from algorithm selection and configuration remains an open problem. In Chapter 6 (based on [250, 251, 249]), we discuss MA-BBOB, a new benchmark problem generator, which we introduce to create larger sets of training and testing functions for these kinds of meta-learning methods.

1.2 Other Contributions by the Author

While this thesis covers a variety of aspects of data-driven benchmarking and its role in dynamic algorithm selection, a set of other tangentially related research directions

have been omitted for reasons of space. A full list of publications can be found at the end of this thesis.

1.3 Thesis Outline

In Chapter 2, we provide information on a variety of background topics related to the work in the thesis. In particular, we discuss continuous optimization and ways in which we can characterize these types of problems. Chapter 3 discusses benchmarking optimization algorithms, with a focus on the benchmarking platform IOHprofiler. This section also discusses more behaviour-based benchmarking in the form of structural bias detection. Chapter 4 focuses on the problems of algorithm configuration and selection, highlighting the challenges in configuring the inherently stochastic optimization algorithms discussed throughout this thesis. In particular, we show how taking a modular approach to algorithm design can lead to significant improvement in performance when algorithm configuration methods are used. Chapter 5 then introduces dynamic algorithm selection and configuration, highlighting the potential benefits to be gained by switching between optimization algorithms. This section shows several use cases, including the promising per-run trajectory-based selection method. Finally, Chapter 6 discusses the problem of generalizability in the context of continuous optimisation and proposes a new test suite on which this aspect can be further investigated. The core results from the thesis are then summarized in Chapter 7, where we finally discuss future research directions based on the insights obtained.

Chapter 2

Preliminaries

In this chapter, we provide background information on several key aspects related to benchmarking and designing optimization algorithms. In particular, we outline many of the techniques we use throughout the remainder of the thesis to evaluate, analyze and configure optimization heuristics.

2.1 Iterative Optimization Heuristics

Throughout this thesis, we study optimization (minimization unless stated otherwise) of problems of the type $f \colon \mathcal{S} \to \mathbb{R}$, i.e., we assume our problem to be a single-objective, real-valued objective function (i.e., problems for which the quality of possible solutions is rated by real numbers), defined over a search space $\mathcal{S}$. While some of the methods discussed are independent of $\mathcal{S}$ (it can be discrete or continuous, constrained or unconstrained), we consider the case where $\mathcal{S} \subset \mathbb{R}^d$ unless stated otherwise. Generally, when dealing with benchmark problems, we assume $\mathcal{S} = [lb_i, ub_i]^d$, which we refer to as a *box-constrained* problem where lb_i and ub_i are the lower and upper bound of the i-th decision variable respectively.

These problems are often considered to be *black-box* problems: we don't assume any information about the underlying structure of the problem. For this type of problem, the only way to get information about it is to evaluate the quality of points $\mathbf{x} \in \mathcal{S}$. This type of problem is common in many real-world problems, e.g. when dealing with simulations of complex processes or physical experiments.

For some problems, some intermediate information about the problem structure is available, in which case we speak of *gray-box* optimization. This can be the case when

Algorithm 1 Blueprint of an iterative optimization heuristic (IOH) optimizing a function $f : \mathcal{S} \to \mathbb{R}$.

procedure IOH
  $t \leftarrow 0$ ▷ iteration counter
  $\mathcal{H}^{(0)} \leftarrow \emptyset$ ▷ search history information
  choose a distribution $\Lambda^{(0)}$ on $\mathbb{N}$ ▷ distribution of the number of samples
  while termination criterion not met **do**
    $t \leftarrow t + 1$
    sample $\lambda^{(t)} \sim \Lambda^{(t-1)}$ ▷ number of points to be evaluated
    Based on $\mathcal{H}^{(t-1)}$, choose a distribution $D^{(t)}$ on $\mathcal{S}^{\lambda(t)}$
    sample $\left(\mathbf{x}^{(t,1)}, \ldots, \mathbf{x}^{(t,\lambda(t))}\right) \sim D^{(t)}$ ▷ candidate generation
    evaluate $f\left(\mathbf{x}^{(t,1)}\right), \ldots, f\left(\mathbf{x}^{(t,\lambda(t))}\right)$ ▷ function evaluation
    choose $\mathcal{H}^{(t)}$ and $\Lambda^{(t)}$ ▷ information update
  end while
end procedure

information about variable interactions can be extracted from the problem formulation [259]. To ease notation, we will still consider these problems as *black-box* in this thesis. We emphasize that the sampling-based optimization algorithms studied in this thesis can be competitive even when the problem f is explicitly known. In pseudo-Boolean optimization, the low auto-correlation binary sequence (LABS) problem is a good example of such a problem that can be defined in two lines, but for which the best-known solvers are sampling-based [182].

The class of algorithms that we are interested in are *Iterative Optimization Heuristics* (IOHs). IOHs are entirely sampling-based, i.e., they sample the search space $\mathcal{S}$ and use the function values $f(x)$ of the evaluated samples x to guide the search. Algorithm 1 provides a blueprint for IOHs. Classical examples for IOHs are local search algorithms (this class includes Simulated Annealing [128] and Threshold Accepting [66] as two prominent examples), genetic and evolutionary algorithms [71], Bayesian Optimization and related global optimization algorithms [113], Estimation of Distribution algorithms [139], and Ant Colony Optimization algorithms [64]. In Section 2.3, we discuss the algorithms which this thesis focuses on.

2.2 Benchmark Problems: BBOB

This thesis is focused on continuous, single-objective, noiseless black-box optimization. As such, we make extensive use of the Black-Box Optimization Benchmarking (BBOB) suite. Originally developed as part of the COCO platform [95], BBOB is one of

the most used benchmark suites for continuous optimization. This suite contains 24 functions, which can be separated into five core classes based on their global properties, which are listed in Table 2.1.

While the suite is originally intended to be used for unconstrained optimization, in practice however, black box optimization functions like this are often considered to be box-constrained [8], in the case of BBOB with domain $[-5, 5]^d$.

Table 2.1: Classification of the noiseless BBOB functions based on their properties (multi-modality, global structure, separability, variable scaling, homogeneity, basin-sizes, global to local contrast). Predefined groups are separated by horizontal lines [96]. Table taken from [164].

	Function	multim.	gl.-struc.	separ.	scaling	homog.	basins	gl.-loc.
1	Sphere	none	none	high	none	high	none	none
2	Ellipsoidal separable	none	none	high	high	high	none	none
3	Rastrigin separable	high	strong	none	low	high	low	low
4	Bueche-Rastrigin	high	strong	high	low	high	med.	low
5	Linear Slope	none	none	high	none	high	none	none
6	Attractive Sector	none	none	high	low	med.	none	none
7	Step Ellipsoidal	none	none	high	low	high	none	none
8	Rosenbrock	low	none	none	none	med.	low	low
9	Rosenbrock rotated	low	none	none	none	med.	low	low
10	Ellipsoidal high-cond.	none	none	none	high	high	none	none
11	Discus	none	none	none	high	high	none	none
12	Bent Cigar	none	none	none	high	high	none	none
13	Sharp Ridge	none	none	none	low	med.	none	none
14	Different Powers	none	none	none	low	med.	none	none
15	Rastrigin multi-modal	high	strong	none	low	high	low	low
16	Weierstrass	high	med.	none	med.	high	med.	low
17	Schaffer F7	high	med.	none	low	med.	med.	high
18	Schaffer F7 mod. ill-cond.	high	med.	none	high	med.	med.	high
19	Griewank-Rosenbrock	high	strong	none	none	high	low	low
20	Schwefel	med.	deceptive	none	none	high	low	low
21	Gallagher 101 Peaks	med.	none	none	med.	high	med.	low
22	Gallagher 21 Peaks	low	none	none	med.	high	med.	med.
23	Katsuura	high	none	none	none	high	low	low
24	Lunacek bi-Rastrigin	high	weak	none	low	high	low	low

For each BBOB function, arbitrarily many problem instances can be generated by applying transformations to both the search space and the objective values [96] – such mechanism is implemented internally in BBOB and controlled via a unique identifier (usually referred to as instance identifier or IID) which defines the applied transformations. For most functions, the search space transformation is made up of rotations and translations (moving the optimum, usually uniformly in $[-4, 4]^d$). Since the objective values are also transformed (through shifting the function values), the performance measures used are generally relative to the global optimum value to allow for comparison of performance between instances, typically in a logarithmic scale.

While this instance generation method is certainly useful for many applications, it has not been without critique. In particular, the stability of low-level features under the used transformations might not be guaranteed [175]. In Section 3.2, we analyze the features of the BBOB instance generation mechanism in more detail.

2.3 Core Algorithms

For continuous optimization, there are several popular algorithm families that have been shown to be effective in the decades since they have been introduced. In this thesis, we focus on two evolutionary algorithms in particular: Differential Evolution (DE) [219] and Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [98]. In addition to these two families, we often make use of a wider set of algorithms, many of them accessed via the Nevergrad framework [200], which are briefly introduced as well.

2.3.1 CMA-ES

One of the most commonly used types of evolutionary algorithms is the CMA-ES [98]. As with most ES, its sampling is based on a d-dimensional normal distribution, from which we can sample as follows:

$$\mathbf{y}^{(t+1,i)} = \mathcal{N}(\mathbf{0}, \mathbf{C}^{(t)}) \tag{2.1}$$

$$\mathbf{x}^{(t+1,i)} = \mathbf{m}^{(t)} + \sigma^{(t)}\mathbf{y}^{(t+1,i)} \tag{2.2}$$

Where $\mathbf{m}$ is the center of mass, $\mathbf{C}$ is the covariance matrix, σ is the step-size and i is the index of the individual in the population. The CMA-ES adapts these parameters based on the relative success of the previously sampled points. For the center of mass, an intermediate recombination strategy is used: the new mean is placed according to a weighted sum of the μ best individuals from the previous generation as follows:

$$\mathbf{m}^{(t+1)} = \mathbf{m}^{(t)} + \sum_{i=0}^{\mu} w_i(\mathbf{x}^{(t+1,i)} - \mathbf{m}^{(t)}) \tag{2.3}$$

Several variants for the recombination weights w_i have been proposed, but the most common variants use exponential decaying weights.

To update the covariance matrix, the CMA-ES incorporates not just information from the current step, but a weighted combination of information collected in the search so far, in the form of the rank-μ update:

$$\mathbf{C}^{(t+1)} = (1 - c_\mu)\mathbf{C}^{(t)} + c_\mu \sum_{i=0}^{\mu} \mathbf{y}^{(t+1,i)}(\mathbf{y}^{(t+1,i)})^T \tag{2.4}$$

Incorporating this previous information allows for a more robust estimation of the

covariance matrix which is most likely to lead to improving steps. However, the used weighted sum causes signs to be lost, which have to be reintroduced to preserve directionality. This is done in the form of the rank-one update, which utilizes the evolution path $\mathbf{p}_c$, which is an aggregation of the search path (selected steps) of the population through a number of successive generations. In practice, $\mathbf{p}_c$ is computed as an exponential moving average of the mean of the search distribution:

$$\mathbf{p}_c^{(t+1)} = (1 - c_c)\mathbf{p}_c^{(t)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}}\frac{\mathbf{m}^{(t+1)} - \mathbf{m}^{(t)}}{\sigma^{(t)}} \tag{2.5}$$

Where c_c is the learning rate for the exponential smoothing of $\mathbf{p}_c$, and μ_{eff} denotes the variance effective selection mass.

The full update of the covariance matrix then reads:

$$\mathbf{C}^{(t+1)} = (1 - c_1 - c_\mu \sum_{i=0}^{\mu} w_i)\ \mathbf{C}^{(t)} \tag{2.6}$$

$$+ c_1 \mathbf{p}_c^{(t+1)} (\mathbf{p}_c^{(t+1)})^T \qquad \text{rank-one update} \tag{2.7}$$

$$+ c_\mu \sum_{i=0}^{\mu} w_i \mathbf{y}^{(t+1,i)} (\mathbf{y}^{(t+1,i)})^T \qquad \text{rank-}\mu\text{ update} \tag{2.8}$$

Finally, the stepsize σ has to be adapted. This is one of the most commonly modified aspects of the CMA-ES procedure, since the covariance adaptation procedure described above does not directly produce a corresponding stepsize update. However, implementations of CMA-ES usually employ the Cumulative Step length Adaptation (CSA), which also employs a form of evolution path: the conjugate evolution path $\mathbf{p}_\sigma^{(t+1)}$, which is constructed via:

$$\mathbf{p}_\sigma^{(t+1)} = (1 - c_\sigma)\mathbf{p}_\sigma^{(t)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\mathbf{C}^{(t)^{-\frac{1}{2}}}\frac{\mathbf{m}^{(t+1)} - \mathbf{m}^{(t)}}{\sigma^{(t)}} \tag{2.9}$$

Here, the inverse square root of the covariance matrix is used, which can be calculated via eigendecomposition. However, this inversion is still computationally expensive, and thus it is recommended to only update this inverse every $\max(1, \lfloor 1/(10n(c_1 + c\mu)) \rfloor)$ generations [90].

To update σ, the length of the conjugate evolution path is compared with its expected length under random selection, i.e.: $\mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||$. If the evolution path is too short, this means that single steps are not making enough progress and cancel each other out, thus the step-size should be decreased. If the evolution path is long,

the progress made in previous steps is correlated, and thus could have been made with fewer steps, and requires an increase in step-size. With an additional dampening parameter d_σ, this allows $\sigma^{(t+1)}$ to be computed as:

$$\sigma^{(t+1)} = \sigma^{(t)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\mathbf{p}_\sigma^{(t+1)})}{\mathbb{E}||\mathcal{N}(\mathbf{0}, \mathbf{I})||} - 1\right)\right) \tag{2.10}$$

Throughout the years, many other step-size adaptation strategies have been proposed [72, 91]. In fact, given its popularity, a wide range of modifications and additional components for the CMA-ES have been published [3, 254, 155, 5]. This explosion of algorithm variants has been the catalyst for the development of modular versions of CMA-ES, starting with modEA [234, 232], which transformed into modCMA [51] and will be discussed in Chapter 4.1.2.

2.3.2 DE

As opposed to CMA-ES, DE does not make use of normal distributions for its candidate generation at all [219]. Instead, DE relies on differences between individuals in its population in the creation of its offspring. In the context of DE, each individual in the population undergoes crossover with a mutant individual, which is in turn created by adding one or more difference vectors to a reference vector. The most basic mutation operator is *rand/1*, where a mutant vector is created as follows:

$$\mathbf{v} = \mathbf{x}_1 + F \cdot (\mathbf{x}_2 - \mathbf{x}_3) \tag{2.11}$$

Where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ are three random (distinct from $\mathbf{x}$ and each other) individuals from the population, and F is the mutation strength. This dependence on differences within the current population enables DE to implicitly scale its mutation without explicitly relying on a step-size mechanism.

Based on the created mutant vector, crossover is performed with the current individual $\mathbf{x}$ as follows:

$$\mathbf{x}'_i \leftarrow \begin{cases} \mathbf{v}_i & \text{if } \mathcal{U}(0,1) \leq Cr \text{ or } i = i_{rand} \\ \mathbf{x}_i & \text{otherwise} \end{cases} \tag{2.12}$$

Where Cr is the crossover rate, $i \in \{0 \dots d\}$ and i_{rand} is a randomly selected index to ensure at least one component from the mutant vector is used. After crossover, the created *trial* solution competes directly against its parent to determine which

individual is added to the next generation.

The structure of DE is modular by design, with the mutation and crossover operators functioning completely separately. This has led to the proposal of many different versions of these operators [266, 73, 106, 49, 37], in addition to a wide range of adaptation mechanisms for their corresponding parameters [224, 266]. In Chapter 4.1.1 we propose a modular version of DE that incorporates a large selection of these modifications.

2.3.3 Other Common Algorithms

PSO Particle Swarm Optimization (PSO) [123] is population-based method where individuals are moved in the search space based on a velocity that is updated according to an individual's search history and the history of its connected individuals.

BFGS Broyden-Fletcher-Goldfarb-Shannon (BFGS) [29, 77, 85, 212] is a Quasi-Newton method that approximates the Jacobian or the Hessian instead of actually computing them.

MLSL Multi-Level Single Linkage (MLSL) [147, 183] is an algorithm that combines global search phases based on clustering with more focused, local search procedures.

Cobyla Constrained optimization by linear approximation (Cobyla) [191] is an algorithm that iteratively optimizes linear approximations of the objective function using trust regions.

EMNA Estimation of Multivariate Normal Algorithm (EMNA) [140] is an estimation of distribution algorithm that uses a multivariate normal distribution.

NGopt An algorithm selection wizard implemented in the Nevergrad platform that selects the optimizer to use based on high-level features such as available budget and problem dimensionality [166].

2.4 Performance Indicators

Since the iterative optimization heuristics only query the objective function through sampling, we use function evaluations as the basis for most performance measures. While this is different from many other disciplines in which solutions are generated

constructively, it allows for much easier ways to compare performance than measures such as CPU time, since function evaluation counts are independent of the used hardware [92].

Generally, we have two axes in which we can consider performance. The first is to consider a *fixed-budget* setting: we have a limited number of function evaluations available and want to find the best possible solution within this budget. This is orthogonal to the *fixed-target* setting, where we have a given quality target and want to reach this threshold with the fewest possible function evaluations.

As discussed earlier, many state-of-the-art IOHs are randomized in nature, therefore yielding random performance traces even when the underlying problem f is deterministic. The performance space is spanned by the number of evaluations, by the quality of the assessed solutions, and by the probability that the algorithm has found within a given budget of function evaluations a solution that is at least as good as a given quality threshold.

Basic Notation To define the performance measures used in this thesis, we use the following notation.

- $\mathcal{F}$ denotes the set of problems under consideration. Each problem (or problem instance, depending on the context) $f \in \mathcal{F}$ is assumed to be a function $f : \mathcal{S} \to \mathbb{R}$. The *dimensionality* of $\mathcal{S}$ is denoted by d. We often consider scalable functions that are defined for several or all dimensions $d \in \mathbb{N}$. In such cases, we make the dimension explicit.

- $\mathcal{A} = \{A_1, A_2, \ldots\}$ is the set of algorithms under consideration. $\mathcal{A}$ can be finite or infinite. Often, $\mathcal{A}$ is a configurable meta-algorithmic framework, which allows users to specify parameters such as the degree of parallelism, the intensity of the local perturbations, the memory size, the use (or not) of recombination operators, etc.

- We denote by r the number of independent runs of an algorithm $A \in \mathcal{A}$ on problem $f \in \mathcal{F}$ in dimension d.

- $T(A, f, d, B, v, i) \in \mathbb{N} \cup \{\infty\}$ is a *fixed-target measure*. It denotes the number of function evaluations that algorithm A performed, in its i-th run and when minimizing the d-dimensional variant of problem f, to find a solution x satisfying $f(x) \leq v$. When A did not succeed in finding such a solution within the maximal allocated budget B, $T(A, f, d, B, v, i)$ is set to ∞. Several ways to deal with

such failures are considered in the literature, as we shall discuss in the next paragraphs.

- Similar to the above, $V(A, f, d, t, i) \in \mathbb{R}$ is a *fixed-budget measure.* It denotes the function value of the best solution that algorithm A evaluated within the first t evaluations of its i-th run, when minimizing the d-dimensional variant of problem f.

Descriptive Statistics We next recall some basic descriptive statistics.

- The average function value over r runs given a budget value t is simply

$$\bar{V}(t) = \bar{V}(A, f, d, t) = \frac{1}{r} \sum_{i=1}^{r} V(A, f, d, t, i).$$

  As we do with all other measures, we omit explicit mention of A, f, and d when they are clear from the context.

- The *Penalized Average Runtime* (PAR-c score, where $c \geq 1$ is the penalty factor) for a given target value v is defined as

$$\text{PAR-c}(v) = \text{PAR-c}(A, f, d, B, v) = \frac{1}{r} \sum_{i=1}^{r} \min \{T(A, f, d, B, v, i), cB\}, \quad (2.13)$$

  i.e., the PAR-c score is identical to the sample mean when all runs successfully identified a solution of quality at least v within the given budget B, whereas non-successful runs are counted as cB. In IOHanalyzer, we typically study the PAR-1 score, which, in abuse of notation, we also refer to as the mean runtime.

- Apart from mean values, we are often interested in quantiles, and in particular in the *sample median* of the r values $\{T(A, f, d, B, v, i)\}_{i=1}^{r}$ and $\{V(A, f, d, t, i)\}_{i=1}^{r}$, respectively.

- We also study the *sample standard deviation* of the running times and function values, respectively.

- The *empirical success rate* is the fraction of runs in which algorithm A reached the given target v within the maximal number B of allowed function evaluations.

That is, in the case of a minimization problem,

$$\widehat{p}_s = \widehat{p}_s(A, f, d, B, v) \tag{2.14}$$

$$= \frac{1}{r} \sum_{i=1}^{r} \mathbb{1}(V(A, f, d, B, i) < v) \tag{2.15}$$

$$= \frac{1}{r} \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, B, v, i) < \infty), \tag{2.16}$$

where $\mathbb{1}(\mathcal{E})$ is the characteristic function of the event $\mathcal{E}$.

Expected Running Time An alternative to the PAR-c score is the *expected running time (ERT)*. ERT assumes independent restarts of the algorithm whenever it did not succeed in finding a solution of quality at least v within the allocated budget B. Practically, this corresponds to sampling indices $i \in \{1, \ldots, r\}$ (i.i.d. uniform sampling with replacement) until hitting an index i with a corresponding value $T(A, f, d, B, v, i) < \infty$. The running time would then have been $mB + T(A, f, d, B, v, i)$, where m is the number of sampled indices of unsuccessful runs. The average running time of such a hypothetically restarted algorithm is then estimated as

$$\begin{aligned} \mathrm{ERT}(A, f, d, B, v) &= \frac{\sum_{i=1}^{r} \min\{T(A, f, d, B, v, i), B\}}{r\widehat{p}_s} \\ &= \frac{\sum_{i=1}^{r} \min\{T(A, f, d, B, v, i), B\}}{\sum_{i=1}^{r} \mathbb{1}(T(A, f, d, B, v, i) < \infty)}. \end{aligned} \tag{2.17}$$

Note that ERT can take an infinite value when none of the runs was successful in identifying a solution of quality better than v.

Cumulative Distribution Functions For the fixed-target and fixed-budget analysis, we can also estimate probability density (mass) functions and compute *empirical cumulative distribution functions* (ECDFs). For the fixed-budget function value, its probability density function is estimated via the well-known *Kernel Density Estimation* (KDE) method [100], which approximates the density function by a superposition of kernel functions (e.g., Gaussian functions with a fixed width) centred at each data point. Intuitively, a set of crowded data points would lead to a very peaky empirical density due to massive superpositions of the kernel, while a set of distant points can only generate a relatively flat curve. For the fixed-target running time (an integer-

valued random variable), we estimate its probability mass function by treating it as a real value and applying the KDE method. For a set $\{T(A, f, d, v, i)\}_{i=1}^{r}$ of fixed-target running times, its ECDF is defined as the fraction of runs that successfully found a solution of quality at least as good as v within a budget of at most t function evaluations. That is,

$$\mathrm{ECDF}(A, f, d, v, t) = \frac{1}{r} \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, v, i) \leq t).$$

ECDF values are most typically used in aggregated form. We use the following two aggregations:

- The aggregation over a set $\mathcal{V}$ of *target values*:

$$\mathrm{ECDF}(A, f, d, \mathcal{V}, t) = \frac{1}{r|\mathcal{V}|} \sum_{v \in \mathcal{V}} \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, v, i) \leq t), \tag{2.18}$$

  i.e., the fraction of (run,target value) pairs (i, v) for which algorithm A has identified a solution of quality at least v within a budget of at most t function evaluations.

- Given a set of functions $\mathcal{F}$ and a mapping $\mathcal{V}\colon \mathcal{F} \to 2^{\mathbb{R}}$ that specifies the target values to consider for each function, the ECDF can be further aggregated by the following definition:

$$\mathrm{ECDF}(A, \mathcal{F}, d, \mathcal{V}, t) = \frac{1}{r \sum_{f \in \mathcal{F}} |\mathcal{V}(f)|} \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}(f)} \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, v, i) \leq t). \tag{2.19}$$

For the BBOB suite in particular, we have a default setting for the set of targets to consider: $\mathcal{V} = \{10^{2-\frac{x}{5}}\}_{x \in 0 \ldots 51}$.

Area Over/Under the ECDF (AOC/AUC) The ECDF can be further aggregated by taking the area under the ECDF (AUC), which results in a measure of *anytime performance* over the included (function, target) pairs included in the ECDF [93]. By taking the difference between the maximal budget and the AUC, we get the area over the ECDF (AOC) instead, which is useful in two ways. First, it allows us to stick to minimization, simplifying visualizations. Second, the AOC is an approximation of the geometric average running time. As such, we often make use of (normalized) AOC in this thesis.

Area Over the Convergence Curve (AOCC) The AOCC is an anytime performance measure, which is equivalent to the area under the cumulative distribution curve (AUC) given infinite targets for the construction of the ECDF [78]. This measure is thus slightly more precise than the AUC, and does not require the selection of the targets to use. Instead, we only need to define the lower (f_l) and upper (f_u) bounds for the function values, as well as an optional scaling function $\mathcal{S}_f$.

$$AOCC(A, f, d, B) = \frac{1}{B \cdot r} \sum_{i=1}^{r} \sum_{t=1}^{B} \left(1 - \frac{clip((\mathcal{S}_f(V(A, f, d, t, i))), f_l, f_u) - f_l}{f_u - f_l}\right)$$

, where the *clip* function caps the function value to the range $[f_l, f_u]$. To remain consistent with the values used for AUC, and analysis of results on BBOB in general, we generally use 10^2 and 10^{-8} as the bounds for our function values, and perform a log-scaling after subtracting the global optimum before calculating the AOCC. We thus calculate the normalized AOCC as follows:

$$AOCC(A, f, d, B) = \frac{1}{B \cdot r} \sum_{i=1}^{r} \sum_{t=1}^{B} \left(1 - \frac{clip((\log_{10}(V(A, f, d, t, i) - f(x*)), -8, 2) + 8}{10}\right)$$

2.5 Exploratory Landscape Analysis

Since the performance of optimization algorithms relies on the problems which are being solved, it is important to focus our analysis not only on the algorithm, but also to look at the structure of the problems themselves. In our black-box context, we can not rely on any outside information about the problem, and thus we can only gain insights through sampling. The field of Exploratory Landscape Analysis [164] (ELA) aims to use information obtained through sampling to estimate the complexity of an optimization problem, by capturing its topology or landscape characteristics. More precisely, the high-level landscape characteristics of optimization problems, such as multi-modality, global structure and separability, are numerically quantified through classes of manually designed low-level features. These landscape characteristics, commonly referred to as ELA features, can be cheaply computed based on a Design of Experiments (DoE), consisting of some samples and their corresponding objective values.

In recent years, ELA has gained increasing attention in the *landscape-aware algorithm selection problem (ASP)* tasks, where the correlation between landscape characteristics and optimization algorithm performances has been intensively researched.

In fact, previous works have revealed that ELA features are indeed informative in explaining algorithm behaviors and can be exploited to reliably predict algorithm performances, e.g., using machine learning approaches [18, 107, 126]. Apart from ASP tasks, ELA has shown promising potential in other application domains, for instance, classification of the BBOB functions [204] and instance space analysis of different benchmark problem sets [216].

One of the most popular implementations of ELA is the Flacco package [127] (and its Python version pFlacco [192]), which incorporates several hundred features. We focus on the set of features which do not require additional sampling after the DoE. In particular, we consider features from six different classes: y-distribution, level set, meta-model, local search, curvature and convexity [164, 165]. While we are fully aware that these ELA features are highly sensitive to sample size [175] and sampling strategy [202, 253], they remain the best tool available at the moment for analyzing the low-level features of black-box problems.

2.6 Reproducibility

Reproducibility is an important aspect in any scientific domain, and evolutionary computation is no different [151]. In this field, many new algorithms are proposed or benchmarked without accompanying code being made public. At best, this hinders the usage of the method by other researchers, but more often it is combined with an underspecification of the algorithm, leading to ambiguities in its working principles.

To ensure the reproducibility of the results presented in this thesis, most publications discussed are linked to a repository on Zenodo. While the individual references to these repositories have been removed from this thesis, they can be found under the Zenodo profile "Diederick Vermetten".[1] In addition to these repositories, many of the datasets discussed throughout this thesis can be accessed directly on the IOHanalyzer website.[2]

[1] https://tinyurl.com/39v642nw
[2] https://iohanalyzer.liacs.nl

Chapter 3

Benchmarking Optimization Algorithms

With the ever-increasing number of available optimization heuristics, insights into which algorithm effectively tackles which kinds of problem characteristics can be hard to come by. Theoretical analysis is often limited to relatively small sets of problems and algorithms, so researchers are forced to rely on empirical results to judge their algorithm's effectiveness. *Benchmarking* aims to bridge the gap between theory and practice by collecting sets of problems with known characteristics, such that an algorithm's performance on these problems can be interpreted and compared to other algorithms.

Benchmarking optimization algorithms in a robust and reproducible manner is a key challenge in the field. In any given study, design choices regarding the used problems (including instances of the same problem), number of runs, evaluation budget, performance criteria etc. have to be made, since each of these variables influences the types of comparisons which are available, as well as how they can be interpreted.

To ease the barrier to robust, yet flexible benchmarking, we propose the IOHprofiler framework in Section 3.1 and illustrate how this tool can contribute to multiple parts of the benchmarking pipeline. Since rigorous benchmarking can result in large amounts of performance data, we also discuss a project related to the creation of an optimization ontology, OPTION, in Section 3.1.4. The core of this section is based on [255, 53, 136], while the problem sets were presented in [43, 177, 242].

As one of the most critical parts of the benchmarking pipeline, we pay particular

attention to the choice of optimization problems used in benchmarking algorithms for continuous optimization: the BBOB suite. In Section 3.2, we discuss an in-depth analysis of the BBOB functions and the corresponding instance generation process, originally published in [150].

Finally, in Section 3.3 we note that benchmarking is not limited to the performance-oriented viewpoint by summarizing several papers on behavior-based benchmarking in the form of structural bias detection [244, 236].

3.1 IOHprofiler

In this section, we introduce IOHprofiler, a tool developed as a collaboration between Leiden University and Sorbonne University (France), with additional input and design help from Tel-Hai College (Isreal). IOHprofiler is a benchmarking platform that aims to integrate elements of the entire benchmarking pipeline, ranging from problem (instance) generators and modular algorithm frameworks over automated algorithm configuration techniques and feature extraction methods to the actual experimentation, data analysis, and visualization [63, 255, 53]. An illustration of the interplay between these different components is provided in Figure 3.1. Notably, IOHprofiler provides the following components:

- **IOHproblems:** a collection of benchmark problems. This comprises pseudo-Boolean, discrete, and continuous problem suites [63, 95], as well as several parameterized problem generators [256]. Section 3.1.3 discusses this component in more detail.

- **IOHalgorithms:** a collection of IOHs. For the moment, the algorithms used for the benchmark studies presented in [63, 7, 51] are available. This subsumes textbook algorithms for pseudo-Boolean optimization, an integration to the object-oriented algorithm design framework ParadisEO [121], and the modular algorithm framework for CMA-ES variants originally suggested in [233] and extended in [51] (discussed further in Chapter 7). Further extensions for both combinatorial and numerical solvers are in progress.

- **IOHdata:** a data repository for benchmark data. This repository currently comprises the data from the experiments performed in several benchmarking studies [63, 134, 208], a sample data set used to illustrate IOHanalyzer functionality in Section 3.1.2, and all compatible data sets from the single-objective,

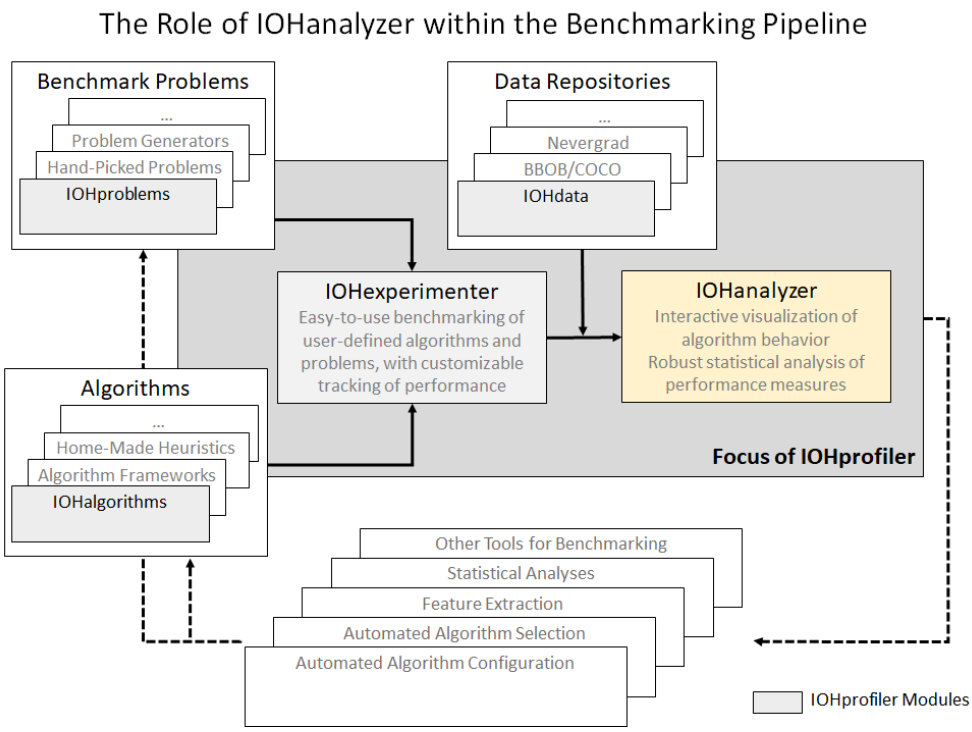


Figure 3.1: Schematic overview of the IOHprofiler, where the focus lies on the IOHanalyzer and IOHexperimenter.

noiseless version of the COCO repository [4]. **IOHdata** also contains performance data from Facebook's Nevergrad benchmarking environment [200], which is updated periodically.

- **IOHexperimenter:** the experimentation environment that executes IOHs on **IOHproblems** or external problems and automatically takes care of logging the experimental data, which will be discussed in Section 3.1.1.
- **IOHanalyzer:** the data analysis and visualization tool presented in Section 3.1.2.

Related Benchmarking Environments

As argued above, benchmarking IOHs is an essential task towards a better understanding of IOHs. It is therefore not surprising that a large number of different tools

have been developed for this purpose. We summarize a few of the tools that come closest to IOHprofiler in terms of functionality and scope.

In evolutionary computation, the arguably best established benchmarking environment is the already mentioned **COCO platform** [95]. Originally designed to compare derivative-free optimization algorithms operating on numeric optimization problems [94], the tool has seen several extensions in the last years, e.g., towards multi-objective optimization [228], mixed-integer optimization [227], and large-scale optimization [238]. COCO consists of an experimentation part that produces data files with detailed performance traces, and an automated data analysis part in which a fixed number of standardized analyses are automatically generated. The by far most reported performance measures from the COCO framework are *empirical cumulative distribution function* (ECDF) curves, see Section 2.4 for definitions. The COCO software has a strong focus on *fixed-target performances* [92], i.e., on the time needed to find a solution of a certain quality.

COCO has been a major source of inspiration for the development of IOHprofiler. What concerns the performance assessment, the key difference between COCO and our IOHanalyzer is in the interactive interface that allows users of IOHanalyzer to study different performance measures, to change their ranges, and granularity. As mentioned, COCO performance files can be conveniently analyzed by IOHanalyzer.

Another important software environment for benchmarking sampling-based optimization heuristics is the **Nevergrad framework** [200]. As with COCO, Nevergrad implements functionalities for both experimentation and performance analysis, accommodating continuous, discrete, and mixed-integer problems. It has a strong focus on noisy optimization, but also comprises several noise-free optimization problems. In addition to studying IOHs, Nevergrad has a special suite to compare one-shot optimization techniques, i.e., non-iterative solvers. The current focus of Nevergrad is to be seen on the problem side, as it offers several new benchmark problems, ranging from modified versions of BBOB to problems optimizing adversarial attacks of image detectors. Nevergrad also provides interfaces to the following benchmark collections: LSGO [143], YABBOB [145], Pyomo [99], MLDA [199], and MuJoCo [226]. The performance evaluation, however, is much more basic than those of COCO or IOHanalyzer, in that only the quality of the finally recommended point(s) is stored, but no information about the search trajectory. That is, apart from taking a *fixed-budget perspective*, Nevergrad does not store performance traces, but only the final output.

3.1.1 IOHexperimenter

To systematically collect performance data from IOHs, a robust benchmarking setup has to be created that allows for rigorous testing of algorithms. For this purpose, we introduce IOHexperimenter. This section is based primarily on [53].

Numerous benchmark problems have been proposed within the evolutionary computation community, and these are often implemented many times over, without an overarching structure or proper maintenance [143, 144, 221]. The importance of using overarching frameworks to facilitate the benchmarking process has been gaining increasing traction within the community in the last decade, especially after [89] showed the benefits that these kinds of tools can provide. Since then, two of the most popular benchmarking tools have been COCO [95] and Nevergrad [200]. While these tools enable users to benchmark their algorithms with relative ease, their overall design has some drawbacks.

In the case of COCO, the enforced design of a suite-based structure allows for very robust benchmarking on problems made available by the developers. However, this simultaneously restricts users to using only that set of available problems and adds a complexity barrier for benchmarking algorithms on other problems. In addition, the logging of performance data follows a fixed framework, and extending it, e.g., to keep track of dynamic algorithm parameters, is not straightforward. Nevergrad, in contrast, offers great flexibility with respect to adding new benchmark problems but is severely limited in terms of the information that is tracked about algorithm performance and behavior. As mentioned above, it essentially only stores the final solution quality after exhausting a user-defined optimization budget.

With IOHexperimenter, we offer a benchmarking module that emphasizes extendability and customizability, allowing users to easily add new problems while providing a comprehensive set of built-in defaults. The logging of performance data is flexible and allows users to customize the content and frequency of the data collected. To improve ease of use, several out-of-the-box storage structures are made available, one of which can be used to collect the same type of data as COCO.

Within the IOHprofiler pipeline, IOHexperimenter can be considered the interface between algorithms and problems, allowing consistent collection of performance data and algorithmic data such as the evolution of control parameters that change during the optimization process. To perform the benchmarking, three components interact with each other: *problems*, *loggers*, and *algorithms*. Within IOHexperimenter, an interface is provided to ensure that any of these components can be modified without

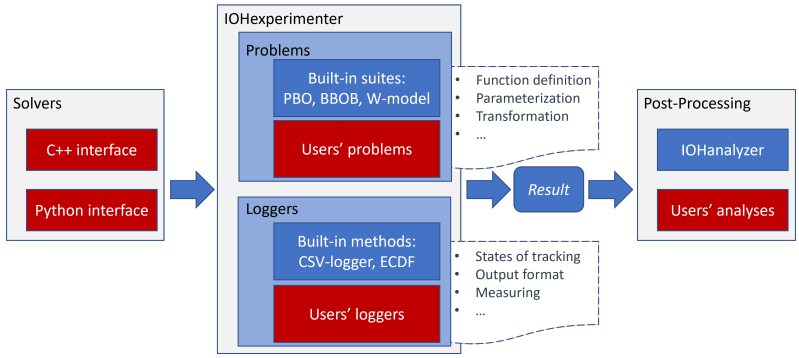


Figure 3.2: Workflow of IOHexperimenter.

impacting the behavior of the others, in the sense that any changes to their setup will be compatible with the other components of the benchmarking pipeline.

Functionality

At its core, IOHexperimenter provides a standard interface towards expandable benchmark *problems* and several *loggers* to track the performance and the behavior (internal parameters and states) of *algorithms* during the optimization process. The logger is integrated into a wide range of existing tools for benchmarking, including *problem* suites such as PBO [63] and the W-model [256] for discrete optimization, COCO's noiseless real-valued single-objective BBOB problems [95] for the continuous case, and submodular problems for constraint optimization [176]. On the *algorithms* side, IOHexperimenter has been connected to several algorithm frameworks, including ParadisEO [121], a modular genetic algorithm [264], a modular CMA-ES [51], and the optimizers in Nevergrad [200]. In [150, 135], the flexibility of IOHexperimenter was demonstrated by generating interfaces between two aforementioned benchmarking tools to execute algorithms from the Nevergrad framework on the BBOB problems from COCO.

Figure 3.2 shows how IOHexperimenter can be placed in a typical benchmarking workflow. The key factor here is the flexibility of its design. IOHexperimenter can be used with any user-provided solvers and problems given a minimal overhead. It also ensures that the output of experimental results follows conventional standards. Because of this, the data produced by IOHexperimenter is compatible with post-processing frameworks like IOHanalyzer [255], enabling an efficient path from algorithm design to performance analysis. In addition to the built-in interfaces to existing software, IOHexperimenter aims at providing a user-friendly, easily accessible way to customize the

benchmarking setup. IOHexperimenter is built in C++, with an interface to Python. In this section, we describe the functionality of the package on a high level, without going into implementation details.[1] In the following, we introduce the typical usage of IOHexperimenter, as well as how it can be customized to fit different benchmarking scenarios.

Problems

Single-Objective Optimization. IOHexperimenter is developed with a focus on single-objective optimization problems, i.e., instances defined as $F = T_y \circ f \circ T_x$, in which $f\colon X \to \mathbb{R}$ is a benchmark problem (e.g., for OneMax $X = \{0,1\}^d$ and the sphere function $X = \mathbb{R}^d$), and T_x and T_y are automorphisms supported on X and $\mathbb{R}$, respectively, representing transformations in the problem's domain and range (e.g., translations and rotations for $X = \mathbb{R}^d$). To generate a problem instance, one needs to specify a tuple of a problem f, an instance identifier $i \in \mathbb{N}_{>0}$, and the dimension d of the problem. Any problem instances that reconcile with this definition of F, can easily be integrated into IOHexperimenter, using the C++ core or the Python interface.

The transformation methods are particularly important for robust benchmarking, as they allow for the creation of multiple problem instances from the same base function. They also allow the user to check algorithm invariance to transformations in search and objective space. Built-in transformations are available for pseudo-Boolean functions [63] and for continuous optimization, implementing the transformations used by [95]. Problems can be combined in a *suite*, which allows the user to easily run solvers on collections of selected problem instances.

Constrained Optimization. Similar to benchmark problems, constraints are defined as free functions that compute a value on an evaluated solution, i.e.; $C\colon X \to \mathbb{R}$, that is non-zero in the case the constraint is violated. IOHexperimenter supports both *hard constraints* C_h and *soft constraints* C_s, of which multiple can be added to any given problem. The single-objective constrained problems are defined by $F_c = F \circ C_h \circ C_s$, which evaluates to ∞ when one of the hard constraints C_h is violated (given minimization). Otherwise, $F_c = F + \sum_{i=0}^{k-1} w_i (C_s^i)^{\alpha_i}$, where k is the number of soft constraints. The weight $w_i \geq 0$ and exponent α_i of a constraint C_s^i can be used by the user to customize a penalty for a constraint violation. In this fashion, arbitrary functions can be added as constraints (thus allowing for both equality and

[1] Technical documentation, a getting-started, and several use-cases are available for both C++ and Python on the IOHexperimenter docs at https://iohprofiler.github.io/IOHexperimenter/.

inequality constraints) to the benchmark problems in IOHexperimenter, allowing the conversion of existing unconstrained problems into constrained problems.

Data Logging

IOHexperimenter provides *loggers* to track the performance of algorithms during the optimization process. These *loggers* can be tightly coupled with the problems: when evaluating a solution, the attached loggers will be triggered to store relevant information. Information about solution quality is always recorded, while the algorithm's control parameters are included only if specified by the user. The events that trigger a data record are customized by the user; e.g., via specifying a frequency at which information is stored, or by choosing quality thresholds that trigger a data record when met for the first time.

A default logger makes use of a two-part data format: meta-information such as function ID, instance, and dimension, written to `.json`-files, and the performance data that gets written to space-separated `.dat`-files. A full specification of this format can be found in [255]. Additional loggers to store the data in memory or use different file structures are available. In addition to the built-in loggers, users can also create their own custom logging functionalities. For example, a logger storing only the final calculated performance measure was created for algorithm configuration tasks [65].

3.1.2 IOHanalyzer

In this section, we present IOHanalyzer, a versatile, user-friendly, and highly interactive platform for the assessment, comparison, and visualization of IOH performance data. IOHanalyzer is designed to assess the empirical performance of sampling-based optimization heuristics in an algorithm-agnostic manner. Our **key design principles** are 1) an easy-to-use software interface, 2) interactive performance analysis, and 3) convenient export of reports and illustrations.

Several other tools have been developed for displaying performance data and/or the search behavior in decision space. However, all tools that we are aware of allow much less flexibility with respect to the performance measures, the ranges, and the granularity of the analysis or focus on selected aspects of performance analysis only (e.g., [33, 68] study statistical significance, whereas [78, 209] aim to visualize performance with respect to multiple objectives). The ability of IOHanalyzer to link the evolution of algorithms' parameters to the evolution of solutions' quality seems to be unique.

IOHanalyzer takes as input benchmarking data sets, generated, e.g., by IOHexperimenter, through the COCO platform, or through the Nevergrad environment. Of course, users can also use their own experimentation platform (IOHanalyzer has a flexible interface for uploading custom `csv`-files). IOHanalyzer provides an evaluation platform for these performance traces, which allows users to choose the performance measures, the ranges, and the precision of the displayed data according to their needs. In particular, IOHanalyzer supports both a fixed-target and a fixed-budget perspective, and allows various ways of aggregating performances across different problems (or problem instances). In addition to these performance-oriented analyses, IOHanalyzer also offers statistics about the evolution of non-static algorithmic components, such as, for example, the hyperparameters suggested by a self-adjusting parameter control scheme.

To illustrate the functionality of IOHanalyzer, we highlight a selected subset of the available functionality (which is listed in more detail in Tables 3.1 and 3.2).

Fixed-Target Results ▶ Single Function ▶ Data Summary: This setting provides basic statistics on the distribution of the fixed-target running time, which are grouped in 3 different tables:

- Table `Data Overview:` This table provides a high-level summary of the currently loaded data set. It simply summarizes the range of *function values* observed in the data set, offering users a quick overview of the quality of the solutions that were evaluated by the algorithms by showing summary statistics of the function values found for the selected function.

- Table `Runtime Statistics at Chosen Target Values:` A screenshot of this table is given in Figure 3.3. The user can set the range and the granularity of the results in the box on the left. The table shows fixed-target running times for evenly spaced target values.[2] More precisely, the table provides the success rate and the number of successful runs as defined in Eq. (2.14), the sample mean, median, standard deviation, the sample quantiles: $Q_{2\%}, Q_{5\%}, \ldots, Q_{98\%}$, and the *expected running time* (ERT) as defined in Eq. (2.17). The user can download this table in `csv` format, or as a LaTeX table.

[2] These target values are evenly spaced between the user-specified minimum and maximum values (whose default values are set to be the extreme values found in the data) on a linear or log scale, based on the difference in order of magnitude between the extreme values found for the specified function. This same principle is used in all similar tables and plots where both a minimum and maximum target can be chosen by the user. A notable exception are the cumulative distribution functions, where *arbitrary sets* of target values can be chosen by the user.

Table 3.1: *Fixed-budget* functionality of IOHanalyzer (v0.1.7).

Section	Group	Functionality	Description
Single Function	Data Summary	*Data Overview*	The minimum and maximum of running times for selected algorithms.
		Target Value Statistics	The mean, median, quantiles of the function value at a sequence of budgets controlled by $B_{\min}, B_{\max}$ and ΔB.
		Target Value Samples	The function value samples at an evenly spaced sequence of budgets controlled by $B_{\min}, B_{\max}$ and ΔB.
	Expected Target Value	*Expected Target Value: single function*	The progression of expected function values over budgets, whose range is controlled by the user.
	Probability Density Function	*Histogram*	The histogram of the function value a user-chosen budget.
		Probability Density Function	The probability density function (using the Kernel Density Estimation) of the function value at a user-chosen budget.
	Cumulative Distribution	*ECDF: single budget*	On **one** function, the ECDF of the function value at **one** budget specified by the user.
		ECDF: single function	On **one** function, ECDFs aggregated over **multiple** budgets.
		Area Under the ECDF	On **one** functions, the area under ECDFs of function values that are aggregated over **multiple** budgets.
	Algorithm Parameters	*Expected Parameter Value*	The progression of expected value of parameters over the **budget**, whose range is controlled by the user.
		Parameter Statistics	The mean, median, quantiles of recorded parameters at an evenly spaced sequence of budgets controlled by $B_{\min}, B_{\max}$ and ΔB.
		Parameter Sample	The sample of recorded parameters at an evenly spaced sequence of budgets controlled by $B_{\min}, B_{\max}$ and ΔB.
	Statistics	*Hypothesis Testing*	The two-sample Kolmogorov-Smirnov test applied on the running time at a target value for each pair of algorithms. A partial order among algorithms is obtained from the test
Multiple Functions	Data Summary	*Multi-Function Statistics*	Descriptive statistics for all functions at a single target value.
		Multi-Function Hitting Times	Raw hitting times for all functions at a single target value.
	Expected Target Value	*Expected Target Value: all functions*	The same as above expect that the expected function values are grouped by functions and the range of budgets are automatically determined.
		Expected Target Value: Comparison	The expected function value at the largest budget found on each function is plotted against the function ID for each algorithm.
	Deep Statistics	*Ranking per Function*	Per-function statistical ranking procedure from the Deep Statistical Comparison Tool (DSCTool) [69].
		Omnibus Test	Use the results of the per-function ranking to perform an omnibus test using DSC.
		Posthoc comparison	Use the results of the omnibus test to perform the post-hoc comparison.
	Ranking	*Glicko2-based ranking*	For each pair of algorithms, a function value at a given budget is randomly chosen from all sample points in each round of the comparison. The glicko2-rating is used to determine the overall ranking from all comparisons.

Table 3.2: *Fixed-target* functionality of IOHanalyzer (v0.1.7).

Section	Group	Functionality	Description
Single Function	Data Summary	*Data Overview*	The best, worst, mean, median values and success rate of selected algorithms.
		Runtime Statistics	The mean, median, quantiles, success rate and ERT at an evenly spaced sequence of targets controlled by $f_{\min}, f_{\max}$ and Δf.
		Runtime Samples	The running time sample at an evenly spaced sequence of targets controlled by $f_{\min}, f_{\max}$ and Δf.
	Expected Runtime	*ERT: single function*	The progression of ERT over targets, whose range is controlled by the user.
		Expected Runtime Comparisons	Comparing the ERT values of selected algorithms at pre-computed targets across all problem dimensions on a chosen problem.
	Probability Mass Function	*Histogram*	The histogram of the running time at a target specified by the user on **one** function.
		Probability Mass Function	The probability mass function of the running time at a target specified by the user on **one** function.
	Cumulative Distribution	*ECDF: single target*	On **one** function, the ECDF of the running time at **one** target specified by the user.
		ECDF: single function	On **one** function, ECDFs aggregated over **multiple** targets.
	Algorithm Parameters	*Expected Parameter Value*	The progression of expected value of parameters over **targets**, whose range is controlled by the user.
		Parameter Statistics	The mean, median, quantiles of recorded parameters at an evenly spaced sequence of targets controlled by $f_{\min}, f_{\max}$ and Δf.
		Parameter Sample	The sample of recorded parameters at an evenly spaced sequence of targets controlled by $f_{\min}, f_{\max}$ and Δf.
	Statistics	*Hypothesis Testing*	The two-sample Kolmogorov-Smirnov test applied on the running time at a target value for each pair of algorithms. A partial order among algorithms is obtained from the test.
Multiple Functions	Data Summary	*Multi-Function Statistics*	Descriptive statistics for all functions at a single target value.
		Multi-Function Hitting Times	Raw hitting times for all functions at a single target value.
	Expected Runtime	*ERT: all functions*	The progress of ERTs are grouped by functions and the range of targets are automatically determined.
		Expected Runtime Comparisons	The ERTs at the best target found on each function (one fixed dimension) is plotted against the function ID for each algorithm.
	Cumulative Distribution	*ECDF: all functions*	On **all** functions, ECDFs aggregated over **multiple** targets.
	Deep Statistics	*Ranking per Function*	Per-function statistical ranking procedure from the Deep Statistical Comparison Tool (DSCTool) [69].
		Omnibus Test	Use the results of the per-function ranking to perform an omnibus test using DSC.
		Posthoc comparison	Use the results of the omnibus test to perform the post-hoc comparison.
	Ranking	*Glicko2-based ranking*	For each pair of algorithms, a running time value at a given target is randomly chosen from all sample points in each round of the comparison. The glicko2-rating is used to determine the overall ranking from all comparisons.
	Portfolio	*Contribution to portfolio (Shapley-values)*	Calculate the approximated Shapley values indicating the contribution of each algorithm to the overall portfolios ECDF.

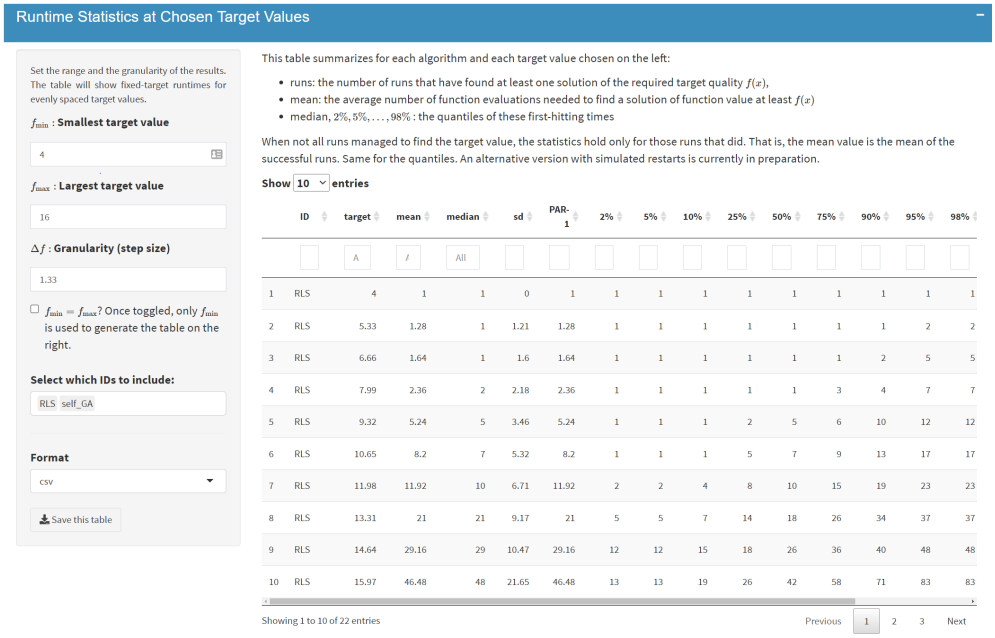


Figure 3.3: Screenshot of the data summary table of some descriptive statistics on the running time.

- Table `Original Runtime Samples:` This table uses the same principle as the `Runtime Statistics:`, but instead displays the values for each individual run. For this table, the user can choose between a "long" (all sample points are arranged in a column) and a "wide" format (all sample points are arranged in a row).

Fixed-Target Results ▶ Single Function ▶ Expected Runtime: An interactive plot illustrates the fixed-target running times. An example of this plot is shown in Figure 3.4. The interactive plot can be adjusted in the menu on the left as shown in the figure. These options include showing/hiding mean and/or median values along with standard deviations and scaling the axes logarithmically. The user selects the algorithms to be displayed as well as the range of target values within which the curves are drawn. By default, this range is set as $[Q_{25\%}, Q_{75\%}]$ of all function values measured in the data set. The displayed curves can be switched on and off by clicking on the legend on the bottom of the plot.

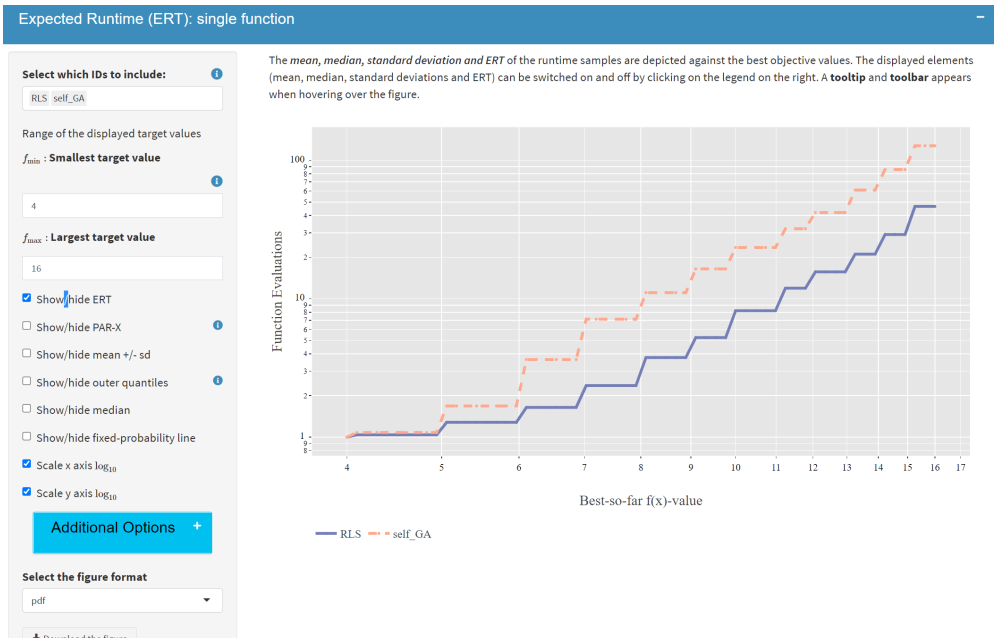
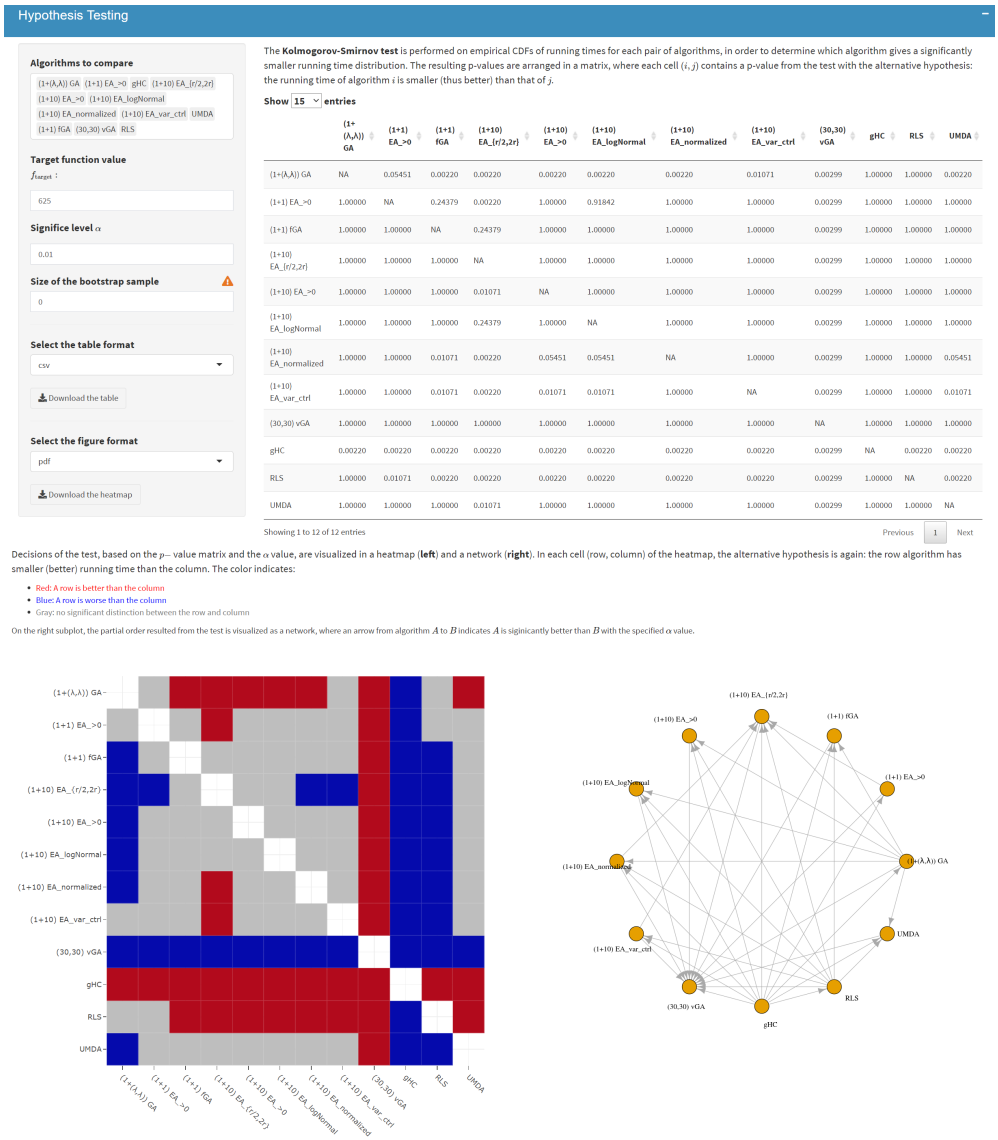


Figure 3.4: Screenshot of the expected running time plot.

Fixed-Target Results ▶ Single Function ▶ Algorithm Parameters: One of the key motivations to build IOHprofiler was the ability to analyze, in detail, the evolution of control parameters which are adjusted during the search. Such dynamic parameters can be found in most state-of-the-art heuristics. While in numerical optimization a non-static choice of the search radius, for example, is needed to eventually converge to a local optimum, dynamic parameters are also more and more common in discrete and mixed-integer optimization heuristics [119, 60]. In the fifth group of fixed-target results for a single function, the evolution of the parameters is linked to the quality of the best-so-far solutions that have been evaluated. In the experimentation (i.e., data generation) phase, the user selects which parameters are logged along with the evaluated function values. These values are then automatically detected by IOHanalyzer and can be chosen in this group for analysis.

As with the interactive plots on expected running time, the user can choose the range of targets, which parameters and algorithms to plot, and the scale (either logarithmic or linear) of x- and y-axis. We omit the example for parameters as the GUI is similar to the one in Figure 3.4. As with "Fixed-Target Results ▶ Single Function ▶ Data Summary", this subsection also provides for each parameter tables of descriptive

statistics (sample mean, median, standard deviation, and some quantiles) as well as the original parameter values.



	(1+(λ,λ)) GA	(1+1) EA_>0	(1+1) fGA	(1+10) EA_{r/2,2r}	(1+10) EA_>0	(1+10) EA_logNormal	(1+10) EA_normalized	(1+10) EA_var_ctrl	(30,30) vGA	gHC	RLS	UMDA
(1+(λ,λ)) GA	NA	0.05451	0.00220	0.00220	0.00220	0.00220	0.00220	0.01071	0.00299	1.00000	1.00000	0.00220
(1+1) EA_>0	1.00000	NA	0.24379	0.00220	1.00000	0.91842	1.00000	1.00000	0.00299	1.00000	1.00000	1.00000
(1+1) fGA	1.00000	1.00000	NA	0.24379	1.00000	1.00000	1.00000	1.00000	0.00299	1.00000	1.00000	1.00000
(1+10) EA_{r/2,2r}	1.00000	1.00000	1.00000	NA	1.00000	1.00000	1.00000	1.00000	0.00299	1.00000	1.00000	1.00000
(1+10) EA_>0	1.00000	1.00000	1.00000	0.01071	NA	1.00000	1.00000	1.00000	0.00299	1.00000	1.00000	1.00000
(1+10) EA_logNormal	1.00000	1.00000	1.00000	0.24379	1.00000	NA	1.00000	1.00000	0.00299	1.00000	1.00000	1.00000
(1+10) EA_normalized	1.00000	1.00000	0.01071	0.00220	0.05451	0.05451	NA	1.00000	0.00299	1.00000	1.00000	0.05451
(1+10) EA_var_ctrl	1.00000	1.00000	0.01071	0.00220	0.01071	0.01071	1.00000	NA	0.00299	1.00000	1.00000	0.01071
(30,30) vGA	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	NA	1.00000	1.00000	1.00000
gHC	0.00220	0.00220	0.00220	0.00220	0.00220	0.00220	0.00220	0.00220	0.00299	NA	0.00220	0.00220
RLS	1.00000	0.01071	0.00220	0.00220	0.00220	0.00220	0.00220	0.00220	0.00299	1.00000	NA	0.00220
UMDA	1.00000	1.00000	1.00000	0.01071	1.00000	1.00000	1.00000	1.00000	0.00299	1.00000	1.00000	NA

Figure 3.5: Screenshot of the multiple testing procedure applied on all 12 reference algorithms on function $f1$ and dimensionality 625. The table shows the p-values resulting from the pairwise KS-test between each pair of algorithms. Then, based on the $\alpha = 0.01$, the resulting hypothesis-rejections are shown in both the matrix-plot and the network.

Fixed-Target Results ▶ Single Function ▶ Statistics: To address the robustness of empirical comparisons, the samples from all algorithm must undergo a proper statistical test procedure [103]. In IOHanalyzer, a standard multiple testing procedure is implemented to compare the fixed-target running time for each pair of algorithms on a single function, for which the well-known Kolmogorov-Smirnov [161] test is applied to the ECDFs of running times. Moreover, the Bonferroni procedure [19] is used to correct the p-value in multiple testing. To demonstrate this functionality, we show, in Figure 3.5, the testing outcome of a data set from running 12 reference algorithms.[3] It can be loaded to the web-based GUI by selecting the PBO data set in the "upload data" section. The data set comprises the results of the experimental study described in [63]. on the PBO problem set from [63], instead of the exemplary two-algorithm data set used previously. Here, the test is conducted across all 12 algorithms on function $f1$ and dimensionality 64 with a confidence level of 0.01. The result of this procedure is illustrated by a table of pairwise p-values, a color matrix of the statistical decision, and a graph depicting the partial order induced by the test (i.e., an arrow pointing from Algorithm 1 to Algorithm 2 is to be read as Algorithm 1 dominating Algorithm 2 with statistical significance). As with all tables and figures in IOHanalyzer, these can be downloaded in several formats, including `tex` and `csv` for tables and `pdf` and `eps` for figures.

Fixed-Target Results ▶ Multiple Functions ▶ Cumulative Distribution: In this group, ECDFs of running times are aggregated across multiple functions, as defined in Eq. (2.19). This functionality is illustrated in Figure 3.6: a table of precalculated target values are provided for each function (all test functions are included by default). This table of targets can easily be edited directly in the GUI, or by a downloading-editing-uploading procedure (which should, of course, not change the format of the tables, just the values). Note that for these ECDF-figures, the corresponding Area Under the Curve (AUC) can also be calculated to get a single value for each algorithm. These AUC-tables are available in the same tab as the ECDF plot.

3.1.3 IOHproblems: Benchmark Suites

Within the IOHprofiler environment, we have the ability to access a wide variety of benchmark problems. Because of the emphasis on extensibility, several benchmark suites have been integrated and used in works related to this thesis. In this section,

[3] This data set is available at https://github.com/IOHprofiler/IOHdata/blob/master/iohprofiler/2019gecco-ins1-11run.rds

3.1. IOHprofiler



Figure 3.6: Screenshot of aggregated ECDF curve across multiple functions and targets.

we provide an overview of the available suites and highlight some of the most interesting results obtained when performing several benchmarking studies. Throughout this thesis, we place a heavy emphasis on the Black-Box Optimization Benchmarking (BBOB) suite [96]. As such, we will discuss this suite in detail in Section 3.2.

The PBO-suite and W-model The Pseudo-Boolean Optimization (PBO) suite was the first set of benchmarking problems integrated in IOHprofiler. This suite consists of 25 problems, ranging from the well-known OneMax to N-Queens. Included in this suite is a set of problems based on the W-model problem generator proposed in [258] (the W-model generator is also available in IOHprofiler). The full description of the problems, as well as results from several well-known optimizers, are available in [63]. This paper also highlights the benefits of flexible parameter tracking, allowing for a detailed analysis of self-adaptive parameters of several GA-variants. Later, another well-known generator of pseudo-Boolean problems (MK-landscapes) was integrated [225].

Submodular Optimization As part of the 2023 Competition on Submodular Optimization at the Genetic and Evolutionary Computation Conference (GECCO), we integrated four different types of (constrained) submodular optimization problems into IOHprofiler: Maximum Coverage, Maximum Influence, Maximum Cut, Packing While Traveling. For each of these problems, we provide several underlying graph instances, leading to a total of 66 functions. Since these functions can either be treated as unconstrained single-objective (by integrating the constraint into the objective function), constrained single-objective or multi-objective, they allow for an interesting comparison of different types of solvers [177, 80].

Star Discrepancy Discrepancy measures are designed to quantify how regularly a point set is distributed in a given space. Among the many discrepancy measures, the most common one is the L_∞ star discrepancy (referred to as simply *star discrepancy* from here on), which is especially important in numerical integration [129, 102], but also in e.g. the design of experiments [207] or in the context of one-shot optimization [21, 20].

The computation of star discrepancy of a set P of n points in a d-dimensional unit-hypercube can naturally be formulated as a real-valued optimization problem on $[0, 1)^d$. However, there is an equivalent discrete formulation [179] on the space $[0, n)^d$. These two formulations of the same underlying problems again allow for potential

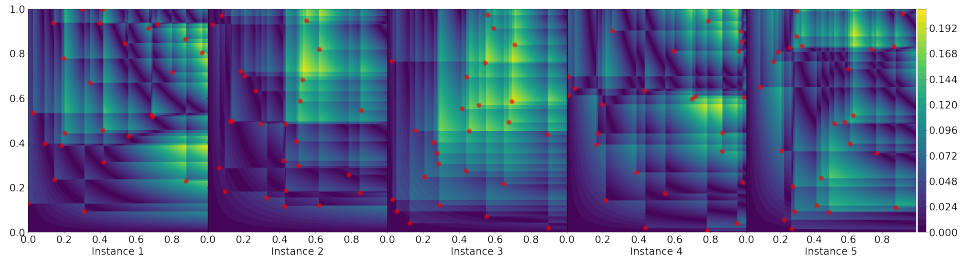


Figure 3.7: Discrepancy values of the first 5 instances of the 2-dimensional version of F31: Uniform sampler with 20 points. The red points indicate the originally sampled points.

comparisons of very different optimization algorithms on the same underlying problem.

To create specific problems for benchmarking, we vary the number of samples in our point sets, as well as the sampler used to create those points. By using Uniform, Halton and Sobol samplers for $n \in \{10, 25, 50, 100, 150, 200, 250, 500, 750, 1000\}$, resulting in 30 problems in IOHexperimenter, with function IDs 30–59. Each problem can additionally be scaled to arbitrary dimensionality, and different instance are obtained by using different seeds for the samplers.

To illustrate the structure of the continuous version of the star discrepancy problems, we show several landscapes in Figures 3.7 and 3.8. Figure 3.7 illustrates the local star discrepancy values for an instance in $d = 2$. We can already observe that the problem of maximizing the local L_∞ star discrepancy bears two important challenges: (1) it is a *multimodal* problem, i.e., there can be several local optima in which the solvers can get trapped (this problem becomes worse with increasing dimensionality); (2) there are *sharp discontinuities* in the local discrepancy values. Slightly increasing one parameter can result in a point falling inside the considered box, causing a $1/|P|$ difference in the local star discrepancy value. Figure 3.8 shows that the problem structure also depends strongly on the point set considered.

To test the performance of our black-box optimization algorithms, we look at the final solutions found by each of the 8 selected optimizers from the Nevergrad library [200]. In order to create a fair comparison, we transform the original discrepancy values to a relative measure, based on the bounds found by two algorithms specifically designed for star discrepancy computation: TA [84] and DEM [59]. Specifically, we

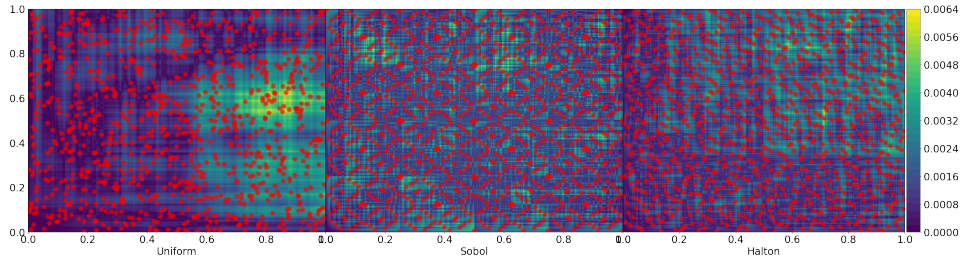


Figure 3.8: Comparison of the 3 samplers for 1000 points in 2D, and the corresponding discrepancy landscapes (instance 1 for F39, F49, and F59 in IOHexperimenter respectively).

consider the following measure:

$$R(x) = \frac{OPT(x) - f(x)}{OPT(x)} \to \min,$$

where $f(x)$ is the final value found after the optimization run, and $OPT(x)$ is the bound calculated by the parallel DEM or TA, depending on the instance size.

Using this relative measure, we can compare the final solutions found by each optimization algorithm across a set of different values of n and d. This is visualized in Figure 3.9. In this figure, we see that the SPSA algorithm is clearly performing poorly, while Random Search seems to be competitive with, if not superior to, all other algorithms for every scenario. In addition to the ranking between algorithms, we also note a clear increase in problem difficulty as the dimensionality increases. Conversely, the number of samples seems to have a rather limited impact on the relative difficulty. This suggests that the structure of the point set has little influence on the performance of the optimizers.

Overall, these results suggest that the star discrepancy benchmarks are challenging for current black-box optimization approaches since random search outperforms every other algorithm considered.

Strict Box Constrained BBOB The final set of benchmark problems we consider is the suite of strict box-constrained BBOB problems (SBOX), which was introduced as part of the workshop at GECCO 2023 under the same name. This suite does not contain any new problems, but instead modifies the BBOB suite by adding strict box constraints ($[-5, 5]^d$, evaluating outside the domain yields a value of ∞). In addition, it modifies some of the instance generation procedures to allow the optimum to be

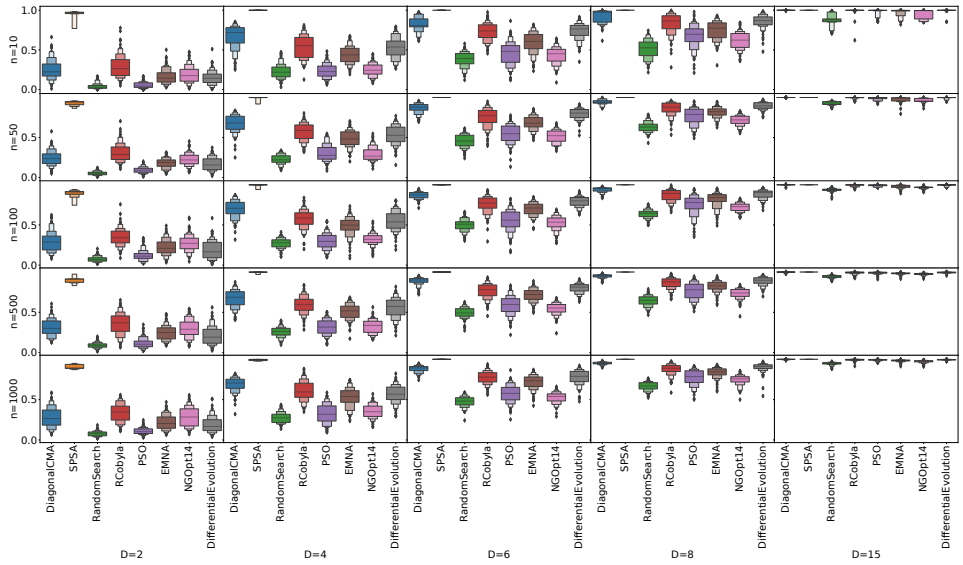


Figure 3.9: Relative final discrepancy value found by each of the used optimizers. Values of 0 correspond to finding the optimal solution, while 1 corresponds to the worst achievable value (0 discrepancy). Box-plots are aggregations of 10 runs on 10 instances, all for the uniform sampler.

located closer to the bounds of the domain. Submissions to this workshop opened up some insightful discussions about the need for box-constraint handling and its relations to reproducibility [242, 167, 57, 27, 114].

3.1.4 Benchmark Data: OPTION

As evident from the large number of benchmark suites and optimization problems, rigorous benchmarking has the potential to generate vast amounts of data. Within the current structure of IOHprofiler, we integrate data for hundreds of algorithms, achieved by integrating with several other platforms. We incorporate COCOs historical data collected from over a decade of workshops, Nevergrads ever-expanding set of benchmarks and optimizers, our own competitions, and much more. By making this data easily accessible in IOHanalyzer, comparisons to existing data are made easier. However, this technique does not fully exploit all information present in this benchmark data. To gain more insight from the existing data, a structured way of storing and querying is required.

One well-established solution to the problem of managing such complex data pools

is the use of ontologies. Ontologies are specifications of a shared conceptualization of data from distributed and heterogeneous systems and databases, and as such, they enable data interoperability, efficient data management and integration, and cross-database search. For this purpose, we collaborated on the creation of the **OPTimization Algorithm Benchmarking ONtology (OPTION)**. This ontology was designed with the main goal of standardizing and formalizing knowledge from the domain of benchmarking optimization algorithms, where an emphasis was put on the representation of data from the benchmark performance space. OPTION offers a comprehensive description of the domain, covering the benchmarking process, as well as the core entities involved in the process, such as optimization algorithms, benchmark problems, evaluation measures, etc. We summarize some of the key design choices for the design of OPTION.

Performance Data There exist many different benchmarking platforms for optimization, each with its own way of storing performance and algorithm data. Three main approaches to the storage of performance data are described below:

- `csv`**-based**: The data is stored as a single file per experiment in a `csv`-based format, where each column represents a performance measure or other meta-information. An example is the format used in Nevergrad [200]. This allows for storing data on many different functions/problems into a single file, with the drawback that the granularity of the data is often limited.

- **Textfile-based**: The data is separated into a single file per function/problem, where the meta-information is delimited in some way, followed by the performance information. This format is easily extendable and human-readable, but it can be hard to work with when files become large. An example of this format is used by the SOS platform [38].

- **COCO/IOH-like**: The data is separated into multiple files and folders: generally, folder structure splits along algorithms and functions/problems. Each folder then contains a file with meta-information about the runs, with links to the files where the raw performance data is stored. This structure makes it easy to find the data sought, but the different links to the files can be an obstacle for practitioners who are not used to this format. Variants of this data format are used by COCO [95] and IOHprofiler [62].

As mentioned, each of these data formats has its advantages and disadvantages. While

there are some commonalities between different methods, the particularities in handling meta-data make interoperability of the data from different sources challenging. Furthermore, these differences lead to more limited post-processing functionalities available to the users of these platforms since they are only compatible with those tools that support their particular data format. While these tools are slowly becoming more interoperable, this process requires significant effort from the developers of the individual tools to make sure all data formats are fully supported. A common data structure would be useful to the benchmarking community to avoid each developer having to do this individually.

In order to create such a common structure, it is crucial to identify the core components of performance data that would be needed in the analysis. Then, an explicit conversion needs to be made for each data format, which extracts these properties from the original data format. Some previous efforts show that it is possible to make such conversions and then jointly analyze data which has been originally stored in different data formats. In particular, performance data from Nevergrad, COCO, IOHexperimenter, and from the SOS platform can be conveniently analyzed through the IOHanalyzer [255].

Problem Landscape Data In addition to the performance data discussed above, we also integrate information about the problems themselves, in the form of ELA features (see Section 2.5). In particular, we use the dataset [202] containing precalculated ELA features for the instances of the BBOB problems we consider.

Domain challenges for data integration A source of complexity in recording performance data from black-box optimization is that we typically do not use a single performance measure. Instead, we are interested in analyzing algorithm performance from different perspectives: small vs. large budgets, the time needed to identify solutions that meet specific quality criteria, the robustness of the algorithm in search and performance space, etc. [8].

To enable such detailed analyses, researchers often record performance data in a multi-dimensional fashion, spanning at least the time elapsed (measured in terms of CPU time and/or function evaluations), solution quality, and robustness. We may also be interested in how dynamic parameters evolve during the optimization process, in which case we record their values along with the performance data. Both requirements add another level of complexity to the data formats and may explain why they differ so much in practice.

There are several other factors that further complicate the interoperability and re-usability of publicly available performance data from different benchmarking experiments:

- Most black-box optimization algorithms are, in fact, families of various algorithm instances. They can be selected by specifying the (hyper-)parameters of the algorithm and/or the operators (e.g., one may speak of Bayesian optimization regardless of the internal optimization algorithm that is used to search the surrogate model, or one may use different acquisition functions, different techniques to build the surrogate, etc.). Different configurations can lead to drastically different search behavior (and hence performance), and it is crucial to associate the recorded data to the appropriate algorithm instance and not only to an algorithm family. However, this is not an easy task, as it can happen that essentially the same algorithm is published under different names (see [35, 218] for recent examples and a discussion, respectively).

- A similar issue appears on the problem side. Different instances of the same problem can be of different complexity, and it is not always clear which problem instances were used within a given benchmark study. In addition, some benchmarking suites automatically rotate, shift, permute, or translate the problem instances, to test specific unbiasedness characteristics and the generalizability of the algorithms. Other suites do not do this (e.g., because the variable order or absolute values carry some meaning) but still refer to problems of different complexity under the same name. As for the algorithms, we can also have the same problem appear under different, possibly multiple, names. The OneMax problem, for example, is sometimes called CountingOnes, OnesMax, the Hamming distance problem, or Mastermind with 2 colors. All these names refer to the same problem.

Identifying such issues cannot (as of yet) be done automatically but require human expertise to annotate the data correctly. While this requires a significant amount of effort for the large amounts of currently available benchmarking data, we aim for the procedure to convert from different data formats to be automated where possible (e.g., by involving the authors of the different benchmarking platforms) and clearly structured where not. In the future, this would then become second nature when introducing a new algorithm / problem / experimental setup, allowing the data ontology to grow organically. The creation of reproducible and readily available data will eventually benefit the optimization community as a whole, so the efforts invested to

achieve this goal would be very much worthwhile.

The Knowledge Base The OPTION ontology contains the semantic model, represented in a formal and standardized way. The OPTION Knowledge Base (KB), on the other hand, leverages the power of the ontology and holds the actual data that has been semantically annotated. This KB is created by annotating data from COCO and Nevergrad for performance data, and ELA features for the BBOB functions for the landscape data.

OPTION in IOHprofiler To query the OPTION KB, SPARQL queries can be used. However, SPARQL queries can become very complex and sometimes are seen as a bottleneck to the broader acceptance of Semantic Web technologies. We recognize that SPARQL query construction is an error-prone and time-consuming task that requires expert knowledge of the whole stack of semantic technologies. Even experts find it sometimes challenging to query semantic data since they first must get familiar with the data annotation schemes or the structure of the knowledge base.

To facilitate the use of the OPTION ontology, we provide a simple GUI that can be used to gain access to performance data without needing to write SPARQL queries. This interface is connected directly to IOHanalyzer, which enables the loaded data to be used directly in performance analysis and visualization, and even be compared to data that might not yet be included in OPTION or to user-submitter performance data. Furthermore, the GUI provides access to a parameterized search process, which can be used without any underlying knowledge about the used semantic data model. Users can express their query by selecting from several drop-down options, which specify the required information, such as suite, function, algorithm, etc., and load the corresponding performance data to analyze. This interface is shown in Figure 3.10. While this interface is static, it illustrates the power of integrating the ontology into IOHanalyzer: users without any background knowledge can use it to gain insight into the performance of the selected algorithms/functions.

Additionally, this interface can be easily expanded based on the community's wishes. To illustrate this potential, we created another entry point into OPTION, which can be used to load all performance data which originated in a specified paper. To this end, the user selects a paper by its title, which then populates the relevant information about the used algorithms and functions in that study. By loading this pre-selected data, the user has full access to the performance data of the selected study, which they can then investigate in more detail by making use of the visualiza-

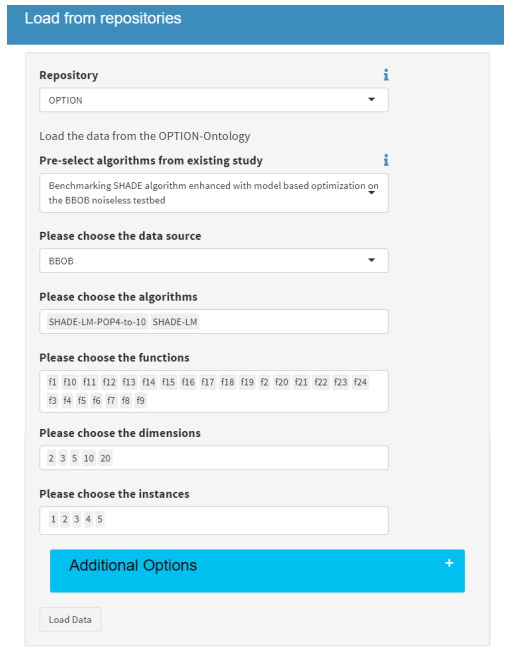


Figure 3.10: The interface of the OPTION-ontology queries within IOHanalyzer (version 1.6.3), available at https://iohanalyzer.liacs.nl/.)

tions within IOHanalyzer. This type of interactive analysis then allows the user to look at the data from different perspectives and to compare it to other algorithms.

3.2 Benchmark Suites: BBOB

Originally developed as part of the COCO platform [95], this set of 24 continuous, single-objective, noiseless problems has become the de facto standard when comparing numerical optimization algorithms for continuous optimization problems. These 24 functions can be separated into five core classes based on their global properties, as seen in Table 2.1. In fact, the suite has been designed to ensure that each problem poses a different 'challenge' to the optimization algorithm. For example, the ellipsoidal function adds ill-conditioning the the sphere problem, so when an algorithm performs well on the latter but not the former the designer gains insight into the deficiencies of their algorithm.

For each BBOB function, arbitrarily many problem instances can be generated by applying transformations to both the search space and the objective values [96] – such

mechanism is implemented internally in BBOB and controlled via a unique identifier (also known as IID) which defines the applied transformations (e.g. rotation matrices). For most functions, the search space transformations are made up of rotations and translations (moving the optimum, usually uniformly in $[-4, 4]^d$).[4] Since the objective values can also be transformed, the performance measures used generally are relative to the global optimum value to allow for comparison of performance between instances, typically in logarithmic scale.

This instance generation method is certainly useful for many applications. For instance, different instances have been considered to enable comparisons between stochastic and deterministic optimization algorithms [189], since using a different instance can in some way be considered as changing the initialization of the deterministic algorithm. It also enables an algorithm designer to test for some invariance properties, particularly with regard to scaling of the objective values, and rotation of the search space [96]. Recently, instances have also been used in a more machine learning (ML) based context, e.g., methods for algorithm selection are trained and tested based on different sets of instances [135].

While creation of different instances of the same function has been very useful to many benchmarking setups, *the underlying assumption that the function properties are preserved is a rather strong one.* For a simple sphere function, the impact of moving the optimum throughout the space can be reasoned about relatively easily, but the impact of the more involved transformation methods on more complex functions is challenging to quantify directly. In addition, the fact that black box optimization problems are in practice often considered to be box-constrained [8], while BBOB was originally designed based on unconstrained function definitions [95], introduces the possibility that some transformations might change key aspects of the function. In fact, it has been shown that the properties of box-constrained functions captured using landscape analysis are not necessarily consistent across instances [171].

In order to analyze the resulting low-level properties of optimization problems, various features of the landscape can be computed. This falls under the field of exploratory landscape analysis (ELA) [165]. While some analysis into the ELA features across instances of BBOB problems has been previously performed [175], we extend the scope of our analysis to include a much wider range of instances. In addition, we consider several other low-level features, such as the location of the global optima, to develop an extensive understanding of the way in which instances might differ. Since

[4] While the suite is originally intended to be used for unconstrained optimization, in practice however, black box optimization functions like this are often considered to be box-constrained [8], in the case of BBOB with domain $[-5, 5]^d$.

we deal with the box-constrained version of the BBOB problems, we also investigate the performance of a set of algorithms, in order to verify that these algorithms perform similarly on different instances of a function – this extends the approach taken in [242].

In this section, we investigate whether the problem characteristics (in the form of ELA features) are preserved across different problem instances and whether the first few (most commonly used) instances are representative of the wider set. In addition, we study whether statistically significant differences in algorithm performance can be found between different instances of the same BBOB problem.

Instance Similarity using ELA

We first focus on analyzing the problem characteristics of different BBOB problem instances based on the ELA approach. For each BBOB instance, we generate 100 sets of DoE data with 1 000 samples each using the Latin Hypercube sampling (LHS) method (so the DoEs are identical for all instances), in order to obtain the ELA feature distribution. We consider a total of 68 ELA features that can be computed without additional sampling, using the package `flacco` [127, 125] and the pipeline developed in [148]. Three of the ELA features, which resulted in the same value across all instances, are deemed not informative and hence dropped out; this means that a final set of 65 ELA features is being considered here.

Comparing Distributions. To investigate how comparable the characteristics of different problem instances are, we carry out the (pairwise) two-sample Kolmogorov-Smirnov (KS) test [161], with the null hypothesis that the ELA distribution is similar in both (compared) problem instances. This results in $\frac{500 \cdot 499}{2} = 124\,750$ comparison pairs per ELA feature. To account for multiple comparisons, we apply the Benjamini-Hochberg (BH) method [12]. To get an overview of differences for a particular ELA feature of each BBOB function, we compute the average rejection rate of the aforementioned null hypothesis of the KS test by aggregating all problem instances (i.e. number of rejections divided by total number of tests). In other words, it shows the fraction of tests which rejected each combination of ELA feature and BBOB function, as shown in Figure 3.11.

On the $5d$ problems, we notice that some features clearly differ between instances, in particular the `ela_meta.lin_simple.intercept`. However, this does not necessarily indicate that all instances should indeed be considered to be different since, as illustrated in [253], some features including this linear model intercept are not invariant to scaling of the objective function. For some other features, such as those

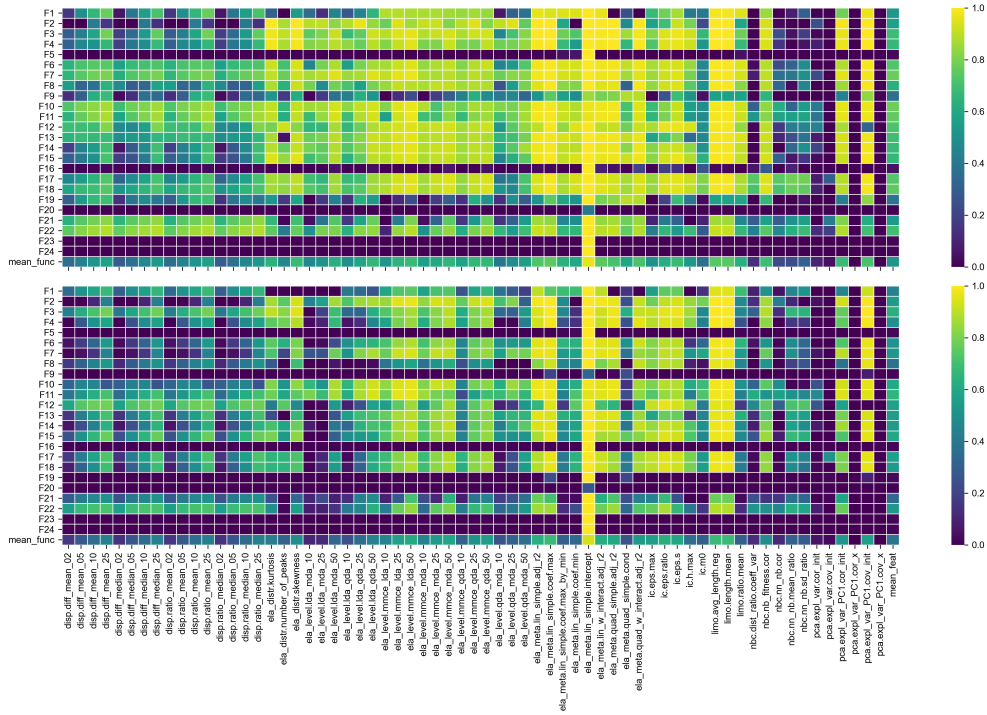


Figure 3.11: Average rejection rate of null hypothesis *distribution of ELA feature between instances is similar*, aggregated over 500 BBOB problem instances in $5d$ (top) and $20d$ (bottom). A lighter color represents higher rejection rate. An extra row (bottom) for the mean over all BBOB functions and an extra column (right) for the mean over all ELA features is added in each heatmap.

related to the principal component analysis (PCA), we notice that barely any test rejections are found. This is largely explained by considering that this feature set is built primarily on the the samples in the search-space, which are identical between instances (the same 100 seeds are used in the calculations for each instance). While the objective values still have an influence on some of the PCA-features, their impact is relatively minor. For the remaining sets of features, we see some commonalities on a per-function basis. Functions F5 (linear slope), F16 (Weierstrass), F23 (Katsuura), and F24 (Lunacek bi-Rastrigin) show no difference between instances.

It is worthwhile to point out that even for a simple function such as F1 (sphere), many features differ between instances. Since translation is the only transformation applied in F1 [96], which (uniformly at random) moves the optimum to a point within $[-4, 4]^d$, it is clear that the high-level function properties are preserved. If the problem

is considered unconstrained, this transformation would indeed be a trivial change to the problem. However, since for ELA analysis, we are required to draw samples in a bounded domain, we have to consider the problems as box-constrained, and thus moving the function can have a significant impact on the low-level landscape features. This might explain why many ELA-features differ greatly across instances on the sphere function.

On the other hand, the same overall patterns can be seen in $d = 20$ as on $d = 5$, albeit with a reduced magnitude. Moreover, functions F9 (Rosenbrock), F19 (Composite Griewank-Rosenbrock), and F20 (Schwefel) now barely show any statistical difference between instances.

Dimensionality Reduction. In addition to the statistical comparison approach, we visualize the ELA features in a $2d$ space using the t-distributed stochastic neighbor embedding (t-SNE) approach [230], as shown in Figure 3.12 for features standardized beforehand by removing mean and scaling to unit variance. It is clear that most instances of each problem are tightly clustered together. Nonetheless, there are outliers, where several instances of a function are spread throughout the projected space, indicating that these instances might be less similar. This is particularly noticeable in $5d$, where several functions are somewhat spread throughout the reduced space. In $20d$, function clusters appear much more stable, matching the conclusion from the differences with regard to dimensionality in Figure 3.11. It is worthwhile to note that differences between BBOB functions are indeed *easier to be detected* in higher dimensions using ELA features, as shown in previous work [204], which matches the more well-defined problem clusters we see in Figure 3.12.

Algorithm Performance across Instances

We now analyze the optimization algorithm performances across different BBOB problem instances. Here, we consider single-objective unconstrained continuous optimization with the following eight derivative-free optimization algorithms available in Nevergrad [200] (all with default settings as set by Nevergrad): DiagonalCMA (a variant of covariance matrix adaptation evolution strategy (CMA-ES)), differential evolution (DE), estimation of multivariate normal algorithm (EMNA), NGOpt14, particle swarm optimization (PSO), random search (RS), constrained optimization by linear approximation with random restart (RCobyla) and simultaneous perturbation stochastic approximation (SPSA). We run each algorithm on each of the 500 instances of the $5d$ BBOB problems, 50 independent runs each, resulting in a total of 4.8 million

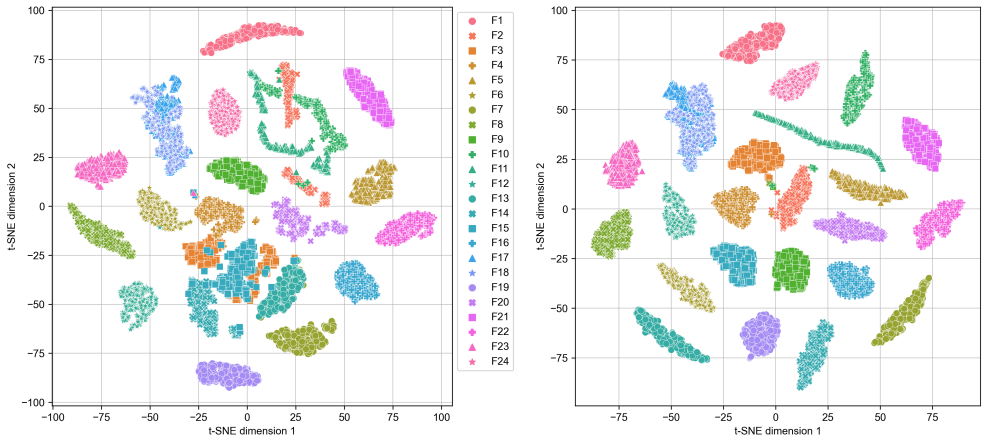


Figure 3.12: Projection of high-dimensional ELA feature space (altogether 64 features, without `ela_meta.lin_simple.intercept`) onto a $2d$ visualization for the BBOB problems in $5d$ (left) and $20d$ (right) using t-SNE approach with default settings.

($= 8 \times 24 \times 500 \times 50$) algorithm runs, each run having a budget of 10 000 function evaluations.

We consider the best function values reached after a fixed budget of 1 000 and 10 000 evaluations. Since we have 50 runs of each algorithm on each instance, we use a statistical testing procedure to determine whether there are significant differences in performance between instances – here, we use the Mann-Whitney U (MWU) test [158] with the BH correction method [12]. In addition to the pairwise testing, we consider the same procedure in a one-vs-all setting. In other words, we repeatedly compare the algorithm performances between the selected instance and the remaining (499) instances. The results are visualized in Figure 3.13, as fractions of times the test rejects the stated null-hypothesis.

We note that RS indeed seems to be invariant across instances, which is to be expected since we make use of relative performance measure (precision from the optimum) rather than the absolute function values. Furthermore, with exception of SPSA, all algorithms have stable performance on F5, F19, F20, F23 and F24, which mostly matches the results from Figure 3.11. The fact that SPSA shows differences in performance between these instances, even on F1, shows that this algorithm is not invariant to the transformations used for instance generation. This matches with previous observations that SPSA displays clear structural bias [244].

We would expect several other algorithms, specifically DiagonalCMA and DE, to

be invariant to the types of transformation used for the BBOB instance generation. However, for some problems, e.g. F12 (bent cigar), such assumption does not seem to hold. This indicates that for these problems, the instances lead to statistically different performance of these invariant algorithms. This might be explainable considering the fact that these algorithms treat the optimization problem as being box-constrained, while the BBOB function transformations make the assumption that the domain is unconstrained [95]. In addition, while the algorithms might in principle be invariant to rotation and transformation, applying these mechanisms does impact the initialization step, which can have significant impact on algorithm performance [247]. This is an intended feature of the BBOB suite, since it is claimed that *"If a solver is translation invariant (and hence ignores domain boundaries), this [running on different instances] is equivalent to varying the initial solution"* [95]. While this is true for unconstrained optimization, it is *not as straightforward* when box-constraints are assumed, as is commonly done when benchmarking on BBOB, since here changing the initialization method might significantly impact algorithm behavior.

Properties across Instances

For most functions, the general transformation mechanism consists of rotations and translations. However, in order to preserve the high-level properties, these transformations are not applied in the same manner for each problem. While translation and

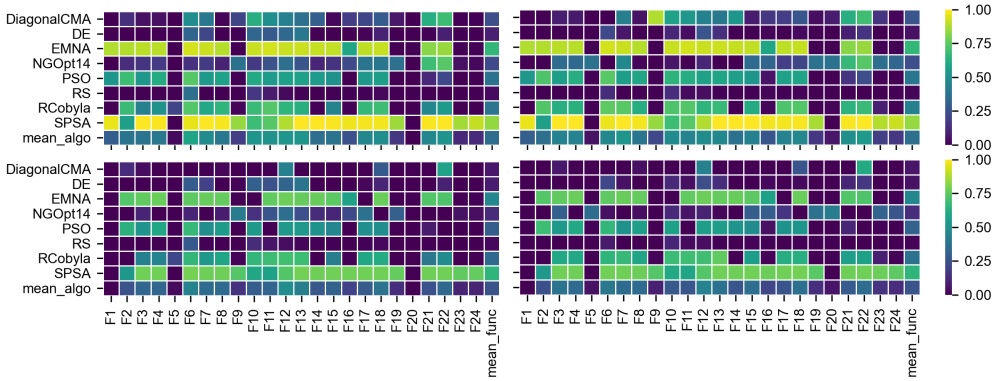


Figure 3.13: Average rejection rate of null hypothesis *algorithm performances are similar across instances*, aggregated over problem instances per function. Left and right column show results for 1 000 and 10 000 function evaluations, respectively. Top and bottom rows show pairwise and one-vs-all comparisons, respectively. Average values are shown in the last column and row of each figure.

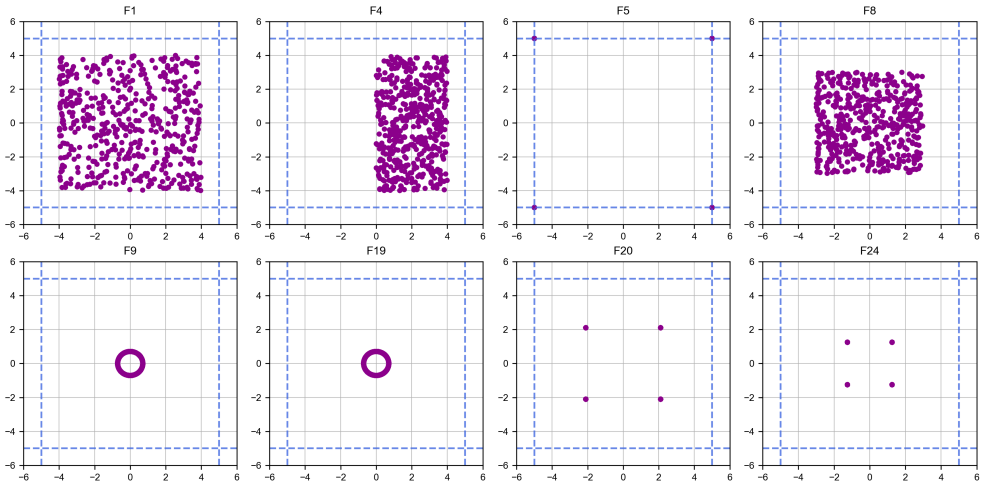


Figure 3.14: Locations of global optima of 500 instances for selected BBOB functions in $2d$. Each dot represents the optimum of a BBOB-problem instance. The remaining 16 BBOB functions (not shown here) have a similar distribution pattern as F1. Dashed lines mark the commonly used boundary of search domain, i.e., $[-5, 5]^2$.

rotation are indeed the core search space transformations, the order in which they are applied in the chain of transformation which creates the final problem can change. For simple functions such as the sphere, the transformation is straightforward (a translation only, since rotating a sphere has no impact). For other functions, such as the Schaffers10 function (F17), one rotation is applied, followed by an asymmetric function and another rotation, after which the final translation is applied. The precise transformations and their ordering is shown in [96]. *While these different transformation processes are necessary to preserve the global properties of the problems, their impact on the low-level features of the problem can not always be as easily interpreted.* As a result, the amount of difference between instances on each function is impacted by its associated transformation procedure, which can make some functions much more stable than others.

One aspect of the instances which is treated differently across problems is the location of the global optimum. By construction, for most BBOB problems, location of this optimum is uniformly sampled in $[-4, 4]^d$. This is achieved by using a translation to this location, since for the default function the optimum is located in the origin. However, for some other problems, such as the linear slope (F5), a different procedure is used. Here, we visualize *true locations of optima across the first 500 instances* of the BBOB functions in $2d$ in Figure 3.14. We note that on most functions the situation

is equivalent to that of F1, with some exceptions: (i) the asymmetric pattern for F4 (Büche-Rastrigin) stems from the even coordinates by construction being used in a different way from the odd ones; (ii) on F8 (Rosenbrock), a scaling transformation is applied before the final translation, resulting in the optimum being confined to a smaller space around the origin; (iii) for the remaining functions (F9, F19, F20, F24), construction of the problem requires a different setup (to ensure the relevant challenges of the problem remain fully inside the domain), and as such the optima will be distributed differently.

In addition to considering the location of the optimum, we aggregate the instances together, resulting in an *overview of regions of the space which are on average better performing, across multiple instances*. This highlights potential bias in the function definition, see Figure 3.15. We observe, e.g., that for sphere function (F1), the domain center has a much lower function value on average than the boundaries, which matches our intuition. This also indicates that initializing a (reasonably designed) algorithm close to the center might more likely result in good algorithm performance, as we on average directly start with better function values. For the BBOB suite overall, we see a *clear skew towards the center of the space*. While this is reasonable given the construction of problems (and the underlying implicit assumption that optimization is unconstrained), it potentially hints towards a set of *functions which are not represented in the suite*, namely those which have optima located near the boundaries, or in general give lower fitness to points close to the bounds. It is also worth mentioning that, unexpectedly, instance generation on some functions, such as the linear slope (F5) does not lead to equal treatment of dimensions, which results in consistently better regions along the boundary of one dimension only. Such a skew is clearly an artefact of a particular choice of slopes for F5.

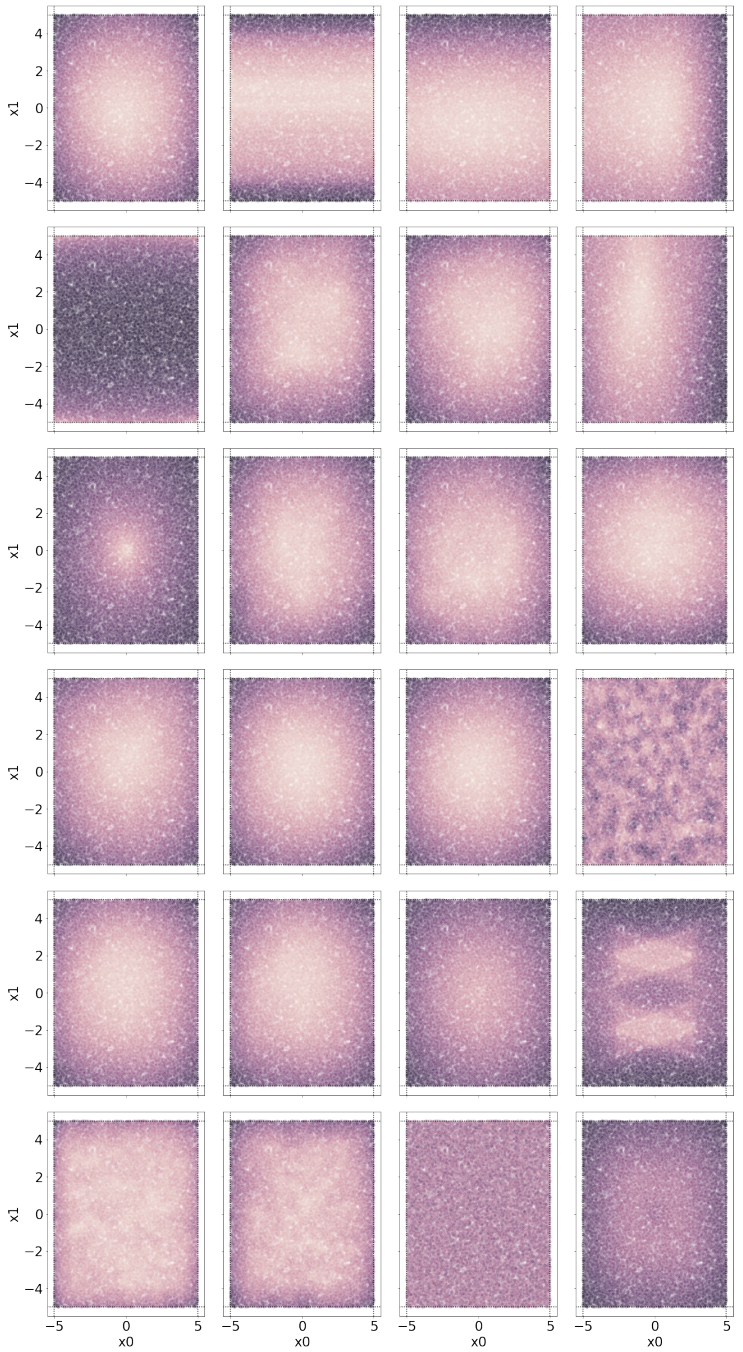


Figure 3.15: Geometric average of relative function value (precision) across the first 500 instances of each BBOB function in 2d.

3.3 Benchmarking Algorithm Behavior: Structural Bias

In addition to the performance or internal state of an algorithm, analyzing the behaviour in terms of points sampled in the domain can lead to important insights into the nature of the optimization procedure. One aspect which we can investigate is whether the algorithm is inherently pulled to some region of the domain, even when there is no selective pressure present. This bias towards a specific part of the search space is called *Structural Bias* (SB) [133]. In this section, we will discuss how SB can be detected, and show some analysis of the SB of several types of optimization algorithms.

The notion of structural bias is built on the assumption that we cannot practically make any assumptions on the location of the optimum within our domain – having an algorithm that consistently finds an optimum only located in the origin is of no use. Therefore, good algorithms should not be biased towards specific locations of the search space, e.g., towards solutions at the origin, centre, or in the borders of the search space. Extrapolating such reasoning, a good optimisation algorithm should be able to find the optima regardless where exactly they are located within the domain. Or, even stronger, a good algorithm should ideally locate solutions anywhere in its domain with equal 'effort', assuming no prior knowledge about the problem is available.

In iterative optimisation heuristics, we can view points sampled during the initialisation as being 'moved' within the domain over time. This movement occurs under the influence of algorithm's operators, which are aiming to find improvements in fitness value. In effect, such movement of the algorithm towards the optima gets steered by the differences in the values of the objective function in the sampled points or their derivatives of some kind. Any feedback that is external to the objective function or domain knowledge might hinder such progression to the optima. Such external feedback stemming from the iterative nature of the algorithm is referred to here as *Structural Bias (SB)*.

Because of the high interdependence between the fitness landscape and the information on the fitness obtained from this cyclical application of the algorithm's operators, the structural bias contribution during the search for optima cannot be easily unveiled if not by means of a specific objective function capable of nullifying such interaction over multiple optimisation runs. The f_0 function serves this purpose and can be used to decouple these effects, thus separating the bias component, arising from algorithmic design choices, from the main driving force represented by the sampled differences in

the fitness landscape [133]. Function f_0 is defined as follows:

$$f_0 : [0,1]^d \to [0,1], \text{ where } \forall x, f_0(x) \sim \mathcal{U}(0,1). \tag{3.1}$$

To identify the structural bias of an algorithm we can thus run it on f_0 and investigate the resulting points found at the end of the optimization run.[5] Up until now, methods to check the uniformity of the distribution of best solutions over multiple runs included visual or statistical inspections.

Displaying locations of the best solutions collected from multiple runs in the so-called 'parallel coordinates' [105] appears to be the most effective way for visualising SB on a multidimensional problem [133]. This approach is easily reproducible, graphically valid and hence convenient. However, when a large number of images is generated [39, 235], visual inspection can become too laborious. Such an approach is also clearly subjective and therefore not reliable for cases where graphical artefacts or unclear patterns cannot be judged by a naked eye.

Instead, we can consider the problem of testing uniformity from the statistical perspective. Assume that a heuristic optimisation algorithm was run N times to minimise f_0. At the end of each run i, the best solution $\mathbf{x}^{(i)}$ found by the algorithm by the end of the run is recorded, where naturally $\mathbf{x}^{(i)} \in [0,1]^d$. The random sample $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(N)}\}$ represents the set of best solutions retrieved by the N runs of the algorithm. Assume that $\{x_j^{(1)}, x_j^{(2)}, \cdots, x_j^{(N)}\}$, $j \in \{1, 2, \cdots, d\}$ was drawn from a probability distribution with a continuous cumulative distribution function $\mathcal{F}_d$. A goodness-of-fit test can be used to test the null hypothesis $H_0 : \mathcal{F}_d \sim \mathcal{U}(0,1)$.

The Kolmogorov–Smirnov test [130] was employed with a sample of size $N = 50$ and significance level $\alpha = 0.01$ in [133]. Subsequently, following the 'power analysis' performed in [131] across three common tests, namely Kolmogorov-Smirnov, Cramér-Von Mises [47], and Anderson-Darling [1] tests, the latter test was chosen and used in combination with the Benjamini–Hochberg [11] correction method for multiple comparisons to achieve higher statistical power. However, it was noted that the original sample size was not adequate for testing all algorithms under investigation. Hence, a higher number $N = 100$ of runs had to be used for some algorithms in order to achieve a satisfactory level of statistical power. Similar problems were encountered in a further study on SB in a subclass of Estimation of Distribution Algorithms [132], even when using an aggregated measure of SB defined as the sum of the statistically significant (across all dimensions) test statistics of the Anderson-Darling test.

[5] Alternatively, the best-so-far point can be used, to track the emergence of SB over time [235]

It was concluded that the described statistical approaches can effectively detect most cases of 'strong' SB but are deficient on other scenarios, including clear 'mild' SB that can be identified with the visual approach. Reasons for these discrepancies between the two methodologies might be a conservative nature of the employed tests combined with the relatively low sample size. Indeed, more accurate SB detection results could be obtained with $N = 600$, as used in [235], instead of $N = 100$.

Using a large sample size ($N = 600$) indeed seems to catch bias more often, but still gives no guarantee to detect *all* different kinds of SB, at any significance level [235]. Even larger sample sizes are necessary for smaller levels of significance, higher desired power and smaller sizes of the deviations to be identified [118]. However, given the limited computational resources to run heuristic optimisation algorithms, it is not always possible to obtain (very) large sample sizes. Therefore, tests better able to detect significant deviations from uniformity given limited sample sizes are desirable for detecting SB.

The BIAS Toolbox As can be concluded from the above, there is a clear need for a better automated statistical testing procedure to detect SB readily available to the community as a software package. We thus propose a toolbox, called BIAS (Bias in Algorithms, Structural), to improve the detection of SB based on algorithm runs on f_0. Based on a large set of statistical tests, BIAS provides an indication of whether or not structural bias is present in the sample of final positions and, in case some SB is found, an assessment of the possible type of SB observed using a random forest model. In addition to the structural bias detection, the toolbox also contains the needed functionality to benchmark other statistical tests for detecting structural bias. The BIAS toolbox is available as a Python package.

Tests

BIAS makes use of statistical tests on the final solutions found by many runs of the algorithm on f_0 (denoted as $\mathcal{F}_d$). These tests all have the **null hypothesis** $H_0 : \mathcal{F}_d \sim \mathcal{U}(0, 1)$. Since the algorithm can work on search spaces of arbitrary dimensionality, we have two types of tests: per-dimension tests and across-dimension tests.

The following tests are designed to work on an individual dimension. However, by aggregating all data we can run these tests on a sample size which is effectively d times larger. If these tests are run on a per-dimension basis, correction strategies need to be applied to deal with the multiple-comparison problem. For this purpose, the Benjamini-Holberg (BH) method was proposed originally, since it is less stringent

Table 3.3: Tests present in the BIAS toolbox for per-dimension (top) and across-dimension (bottom) detection of uniformity.

Test Name	Shorthand	Reference
1-spacing-based test	1-spacing	[194]
2-spacing-based test	2-spacing	[194]
3-spacing-based test	3-spacing	[194]
Sample Range-based test	range	[244]
Sample Minimum-based test	min	[244]
Sample Maximum-based test	max	[244]
Anderson-Darling	AD	[1, 75]
Transformed Anderson-Darling test	tAD	[241]
Shapiro test	Shapiro	[214]
Jarque–Bera test	JB	[111, 220]
Minimum Linear Distance-based test	LD-min	[244]
Maximum Linear Distance-based test	LD-max	[244]
Kurtosis-based test	Kurt	[185]
Minimal Minimum Pairwise Distance-based test	MPD-min	[244]
Maximal Mininimum Pairwise Distance-based test	MPD-max	[244]
Wasserstein distance-based test	Wasserstein	[117]
Neyman-Smooth test	NS	[178, 14]
Kolmogorov-Smirnov test	KS	[130, 75]
Cramer-Von Mises test	CvM	[44, 50]
Durbin test	Durbin	[67, 50]
Kuiper test	Kuiper	[30, 50]
1st Hegazy-Green test	HG1	[101, 50]
2nd Hegazy-Green test	HG2	[101, 50]
Greenwood test	Greenwood	[86, 50]
Quesenberry-Miller test	QM	[196, 50]
Read-Cressie test	RC	[46, 50]
Moran test	Moran	[169, 50]
1st Cressie test	Cressie1	[45, 50]
2nd Cressie test	Cressie2	[45, 50]
Vasicek test	Vasicek	[239, 50]
Swartz test	Swartz	[222, 50]
Morales test	Morales	[168, 50]
Pardo test	Pardo	[184, 50]
Marhuenda test	Marhuenda	[159, 50]
1st Zhang test	Zhang1	[265, 50]
2nd Zhang test	Zhang2	[265, 50]
The mutual information-based test	MI	[213, 186]
Maximum Minimum Pair-wise Distance-based test	MMPD	[244]
Maximum Difference per Dimension between a linear uniform distribution-based test	MDDLUD	[244]

than the standard Bonferroni method. However, in this chapter, we also investigate the effects of other multiple comparison correction methods.

We consider a total of 36 per-dimension tests, listed in Table 3.3. In addition to the tests which work on a per-dimension basis, we can also perform tests on the full set of 30-dimensional data at once. This can be done by grouping together the samples or distances and performing the same test as the per-dimension testing on the aggregated data. For this purpose, we use all per-dimension tests, with the exception of the sample limits-based tests, *LD*, *MPD*, and *Wasserstein* tests. Alternatively, we can use tests which are explicitly designed to handle multiple dimensions at once. In this toolbox, we consider three of these tests, listed at the bottom of Table 3.3.

Table 3.4: Overview of parameterised data sampling scenarios in $[0, 1]$, the total of 194 / 249 scenario settings (per dimension / across dimensions), per considered sample size.

scenario name	how sampled	parameter 1	parameter 2	number of settings	diagnosis[6]
`Uniform`[7]	sample full sample size via $\mathcal{U}(0, 1)$	–	–	1	no bias
`Cut Uniform`[8]	subscenario 1: sample full sample size via $\mathcal{U}(z_c, 1)$, subscenario 2: sample full sample size via $\mathcal{U}(\frac{z_c}{2}, 1 - \frac{z_c}{2})$	fraction cut $z_c \in \{0.01, 0.025, 0.05, 0.1, 0.2\}$		10/10	centre-bias
`Cut Normal`[9]	sample full sample size via $\mathcal{N}(\mu, \sigma)$, remove all points outside $[0, 1]$ and repeat until full sample size	$\sigma \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$	$\mu \in \{0.5, 0.6, 0.7\}$	15/15	centre-bias
`Inverse Cut Normal`	same as above, but transform to have most mass at bounds	same as above	same as above	15/15	bound-bias
`Cut Cauchy`	similar to `Cut Normal` but for Cauchy distribution	same as above	same as above	15/15	centre-bias
`Inverse Cut Cauchy`	same as above, but transform to have most mass at bounds	same as above	same as above	15/15	bound-bias
`Clusters`	sample n_c cluster centre points c_i via $\mathcal{U}(0, 1)$, sample remaining points around them via $\mathcal{N}(c_i, \sigma)$[10]	number of clusters $n_c \in \{1, 2, 3, 4, 5\}$	$\sigma \in \{0.01, 0.025, 0.05, 0.1, 0.2, 0.3\}$	30/30	cluster-bias
`Consistent Clusters`[11]		same as above	$\sigma \in \{0.01, 0.025, 0.05, 0.1, 0.2, 0.3\}$	0/30	cluster-bias
`Loose Clusters`	sample z_u portion of sample size via $\mathcal{U}(0, 1)$. For each remaining point, select an existing point x_i and sample $\mathcal{N}(x_i, \sigma)$	fraction of uniform samples $z_u \in \{0.1, 0.25, 0.5\}$	$\sigma \in \{0.01, 0.02, 0.05, 0.1\}$	12/12	cluster-bias
`Gaps`	sample full sample size via $\mathcal{U}(0, 1)$, select n_c sampled points x_i, remove all sampled points in $[x_i - r_g, x_i + r_g]$, resample missing points outside gaps via $\mathcal{U}(0, 1)$ until full sample size[12]	number of centres $n_c \in \{1, 2, 3, 4, 5\}$	gap radius $r_g \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$	25/25	gap-bias
`Consistent Gaps`[13]		same as above	same as above	0/25	gap-bias
`Spikes`	randomly sample integers in $[1, n_s]$, rescale as uniformly placed spikes in $[0, 1]$	number of spikes $n_s \in \{25, 50, 100, 150, 200, 250, 500, 1000\}$	–	8/8	discretization-bias
`Noisy Spikes`	same as above, but spike locations are independently shifted according to $\mathcal{N}(0, \sigma)$	same as above	$\sigma \in \{0.005, 0.01, 0.02, 0.03, 0.04, 0.05\}$	48/48	discretization-bias

[6] See Section 3.3
[7] Sanity check, excluded from the tests
[8] Two subscenarios: 1. modify only min (equivalent to varying only max, so only one is used to save time); 2. modify both min and max at the same time, with the same parameter setting (half cut on both sides)
[9] Vary μ only to one side since it is equivalent to the other side
[10] For across-dimension tests, need to differentiate between same cluster-centres in each dimension (`Consistent Clusters`) vs. different gaps (`Clusters`)
[11] Only used in across-dimension tests, as it is equivalent to `Clusters` per-dimension
[12] For across-dimension tests, need to differentiate between same gaps in each dimension (`Consistent Gaps`) vs. different gaps (`Gaps`)
[13] Only in across-dimension tests, as it is equivalent to `Gaps` per-dimension

To determine rejection based on the test statistic, we need to either calculate the corresponding p-values, or check if the test statistic exceeds the corresponding critical value. Several of the test we include calculate the p-value by default, but for the others we will use the critical values. To get accurate estimates of these values, we use a 100 000 samples Monte Carlo simulation of the test statistic under the uniform distribution, from which we determine the α-quantiles for $\alpha \in \{0.01, 0.05\}$. The Monte Carlo test is a well known procedure for implementing hypothesis tests [181]. It enables calculating the critical values when the true (sampling) distribution of the test statistic is unknown. The resulting critical values calculated using this procedure are made available.

Methodology

To effectively judge the performance of the proposed tests for different types of SB, we have defined a large portfolio of bias scenarios according to which we can generate an arbitrary number of samples. This set of scenarios is chosen in such a way that most common types of SB are represented. Additionally, these scenarios are parameterised to control the level of bias, which enables us to better judge the robustness of tests. The specification of these scenarios is shown in Table 3.4.

Adding up all parameterizations gives us 194 scenarios to consider in the per-dimension case, and 249 scenarios in the across-dimension case. For each of these scenarios, we generate data with sample sizes $\{30, 50, 100, 600\}$. In the per-dimension case, we collect 1 500 independent sets of samples for each use-case, while the across-dimension cases all use 100 sets of 30-dimensional samples.

For each of the generated sets of samples, we apply the corresponding test-battery with $\alpha \in \{0.05, 0.01\}$: 36 tests for per-dimension case and 32 for the across-dimension case.[14] Using this setup, we thus collect $194 \cdot 1\,500 \cdot 4 \cdot 36 = 4.19 \times 10^7$ test statistics / p-values for the per-dimension tests, and $249 \cdot 100 \cdot 4 \cdot 32 = 3.19 \times 10^6$ for the across-dimension tests.

We show an example of the set of statistical tests applied to an instance of the `Cut Normal` scenario in Figure 3.16. This figure shows the rejections for each dimension individually, as well as the corresponding sample on which this decision is based. This visualisation is available as part of the BIAS toolbox, and provides a visual way to inspect the structural bias present in the scenario.

[14] Several per-dimension tests can not be used for the across-dimension case, full details in [244].

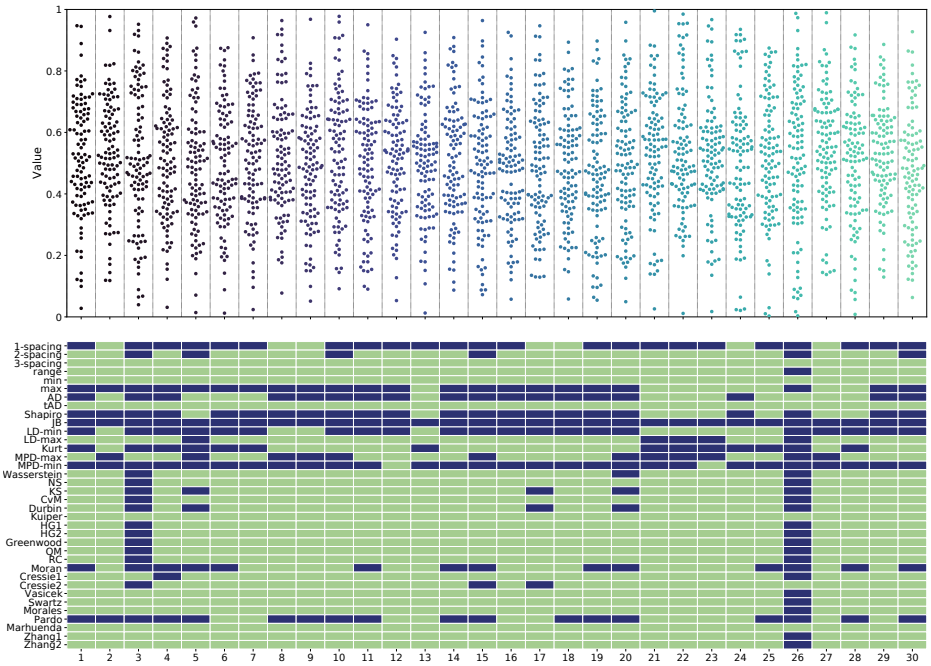


Figure 3.16: Example of an instantiation of the `Normal Cut` scenario with $\mu = 0.5$ and $\sigma = 0.2$, with 100 samples in each of 30 dimensions. The top figure shows the assumed distribution of the final positions potentially returned by an optimisation algorithm in each dimension. Jitter is applied here to reveal vertically overlapping points. The colour scheme is used to highlight different dimensions. The binary heatmap in the bottom figure shows in green which tests reject the null-hypothesis of uniformity per dimension with $\alpha = 0.01$ (no multiple comparison correction applied).

Sample size

To study the impact of the available sample size on the overall performance of different statistical tests, we can aggregate the number of rejections over all parameterizations of each scenario. This allows us to show the fraction of cases of a scenario which are rejected by each test given a certain sample size. Figure 3.17 shows this for the `Shifted Spikes` scenario. From this figure, we can see that the effect of sample size is not the same across all tests. As an example, the *`AD`* test has a relatively high number of rejections at 30 samples, but doesn't reach the same precision as other tests when increasing sample size to 600. This indicates that analysis of the performance of the tests should take the number of available samples into account, as this will influence which tests are more distinguishing.

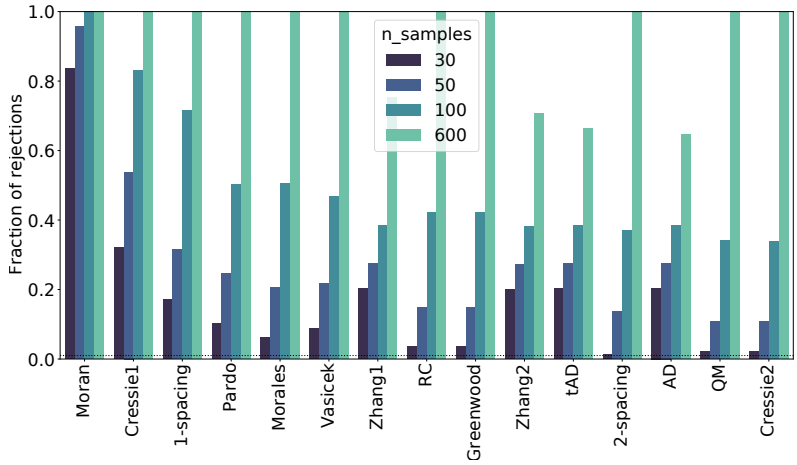


Figure 3.17: Fraction of rejections for each test on the `Shifted Spikes` scenario, with $\alpha = 0.01$ and no multiple comparison correction method applied. Data is aggregated over all parameterizations of the scenario, as described in Table 3.4. This figure shows 15 tests with the most rejections (when aggregated over the different sample sizes). Note that the negative space over each bar $(1 - x)$ is equivalent to the false negative rate of the test.

From Figure 3.17, we can also see that the ***`Moran`*** test significantly outperforms all others on this scenario, but even this test does not reject all cases when the sample size is small. This reinforces the notion that if possible, increasing the sample size is beneficial to the ability to detect less clear cases of structural bias. However, we also note that for most scenarios, a sample size of 50 seems to be sufficient to detect the presence of structural bias. While increasing the sample size would increase the ability to detect less obvious cases of SB, $N = 50$ should be able to correctly identify the most blatant ones.

Overall analysis

With the rejection data, we can investigate the interplay between statistical tests and the scenarios, in order to find what set of tests is more suitable to each kind of structural bias. For this analysis, we make use of the concept of Shapley values [215] to assess the contribution of each test to a portfolio of tests for finding bias in each type of scenario. In particular, we define the marginal contribution of a test t to a

portfolio of tests $T' \subset T$ on scenario S as follows:

$$c(t, T', S, n, \alpha) = \sum_{s \in S} \sum_{i=0}^{n-1} \max_{t' \in (T' \bigcup \{t\})} \mathbb{1}_{t'(s_i) < \alpha} - \sum_{s \in S} \sum_{i=0}^{n-1} \max_{t' \in T'} \mathbb{1}_{t'(s_i) < \alpha} \tag{3.2}$$

where n is the number of realizations s_i of scenario s. The indicator function $\mathbb{1}$ corresponds to the test t rejecting the null hypothesis with significance α on the data from realization s_i.

Based on this definition of marginal contribution, we can compute approximate Shapley values by sampling random permutations calculating the marginal contribution for each test at each position within this permutation [229, 41]. This can be formulated as follows:

$$S(t, S, n, \alpha) = \sum_{r} \sum_{i=0}^{m} c(t, T', S, n, \alpha) : T' \subset T, |T'| = i \tag{3.3}$$

where r is the number of repetitions used, and m is the maximum size of these permutations, which is introduced to ease with computations and because the impact of larger permutations on the total sum is relatively minor – in this paper, we set $m = 10$.

From our experiments, we have found that no single test is clearly preferable over all others. Moreover, an analysis of the Kendall-Tau [122] correlations between the rejections of tests across all scenarios shows that very few tests are highly correlated. Figure 3.18 shows the correlation heatmaps for sample size 600 and $\alpha = 0.01$. We can observe relatively higher correlations among some of the tests listed from `NS` to `Greenwood`, and among some of the tests listed from `QM` to `Pardo`. However, these higher correlations involve very few of the tests and the correlations among the other tests are very low. Therefore, it is likely that different tests are best suited to recognise different types of deviations from uniformity. For this reason, we include *all considered tests* in the BIAS toolbox.

For the per-dimension tests, we should take into account the fact that multiple tests are being done, and thus the p-values should be changed using a correction procedure. For this purpose, we use the Benjamini-Yekutieli (BY) [13] correction method, which we found to obtain the best tradeoff between false positives and false negatives [244].

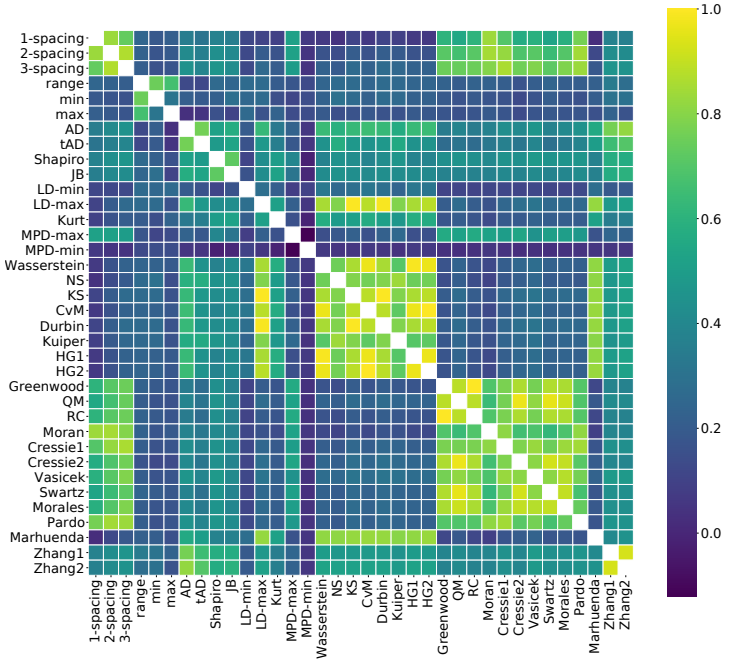


Figure 3.18: Cluster plot showing the Kendall-Tau correlations [122] between test rejections on all scenarios, with sample size 600, $\alpha = 0.01$.

Estimation of the SB type

Since we use the results of many statistical tests to find bias in artificially generated samples and different tests may be better at capturing different deviations from uniformity, we can use these tests to not only check if structural bias is present, but also to identify what the most likely form of bias is. This provides an answer to RQ2. To achieve this, we build a random forest (RF) model, which takes as input the *test-rejections from all per-dimension tests*. This is done to allow scaling to arbitrary dimensions while having one model for all sample sizes. Specifically, if we use statistical test values directly, we would need one model per sample size, and a way to aggregate the resulting predictions. Instead, a RF based on rejections only needs to deal with the aggregation problem.

The data used to train and evaluate the random forest model consists of the full set of scenario results (per-dimension version) on all tests, with the output being the scenario-type it comes from. However, if for a specific sample no test rejects the null-hypothesis, these samples are discarded, since we have no evidence of structural bias.

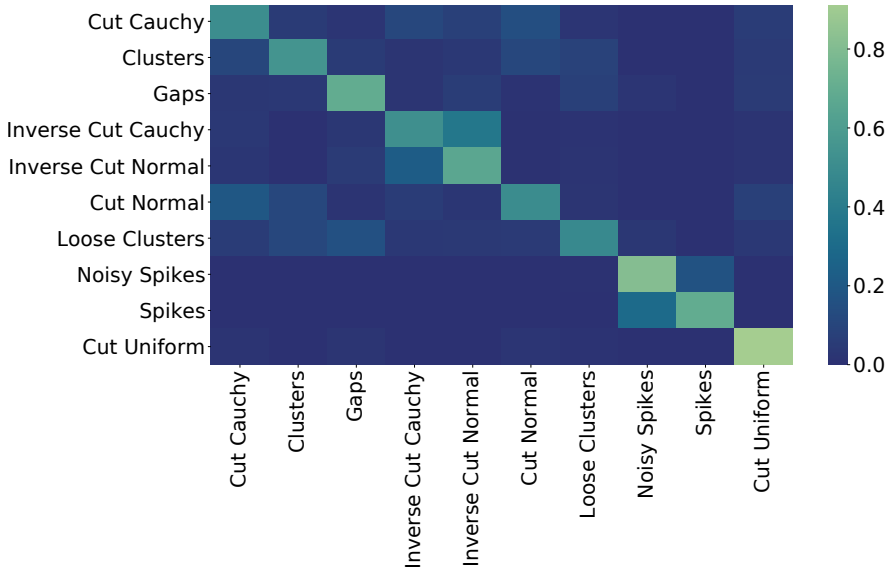


Figure 3.19: Confusion matrix for the random forest model trained on test rejections, aggregated over all sample sizes. The true scenario is shown on the y-axis, while the predicted scenario is on the x-axis.

This two-stage approach leaves us with 1 158 000 biased samples, on which we train the RF model with 100 trees and balanced class weights. A confusion matrix created from an 80-20 test split is shown in Figure 3.19 (F1-score of 0.56). We see that the distinction between the `Cut Uniform` and the other scenarios can be challenging to accurately detect. However, this doesn't have to be an issue for practical detection of SB, since the scenarios misidentified as `Cut Uniform` might show similar types of bias, even though their initial creation mechanism is different.

To provide a more practical estimation of SB in our toolbox, we create an additional model to predict the type of bias, as shown in the final column of Table 3.4. These 5 categories are more distinct from each other, removing overlap between some similar classes, i.e. between `Spikes` and `Noisy Spikes`. Overall, this model gives us an improved F1-score of 0.79 on a similar 80-20 split.

To use these models in the BIAS toolbox to predict bias of the multi-dimensional test, we need to perform some aggregation across dimensions to transform it into a binary vector. We do this by checking the number of false positive tests in 30D uniform samples. We run 10 000 simulations, where we record the maximum number of test rejections by each test. This gives us a total of 92 cases where a test gives 2

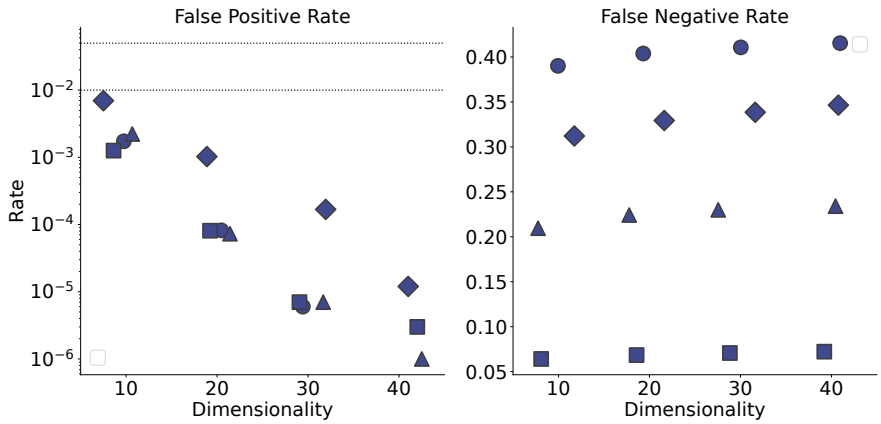


Figure 3.20: Evaluation of toolbox in different dimensions at $\alpha = 0.01$: fraction of false positives (on the left) and aggregated fraction of False Negatives across all scenarios (on the right). On both figures, markers identify the used sample size: ◯, ♢, △ and □ are 30, 50, 100 and 600, respectively.

rejections, and 2 cases where a test gives 3 rejections. As such, we set the threshold for the aggregation of multi-dimensional data to $0.1 \cdot d$. If no test is rejected in this aggregation, we consider the samples to be non-biased. This threshold value is then used to create the binary input vector for the RF model.

To verify that this works for other dimensionalities as well, and to gauge the overall performance of the toolbox, we simulate the false positive and false negative rates. This is achieved by sampling (with replacement) from the set of test-statistics on each of our used scenarios and applying this aggregation rule. For false positives, this is done 1 000 00 times on the (true) uniform data, while for false negatives it is done 10 000 times on every non-uniform scenario. The results, shown in Figure 3.20, indicate that while the $0.1 \cdot d$ threshold is rather conservative on higher dimensionalities, the FPR is well below the selected $\alpha = 0.01$, while the FNR is not needlessly increased.

Benchmarking SB of real algorithmic data

We use data from a heterogeneous pool of heuristics executed over f_0 at dimensionality $d = 30$ for a maximum of $10000 \cdot d$ fitness function calls. In total, we consider 432 optimisation heuristics, which fall into the following categories (all except the latter use $N = 100$, while the latter uses $N = 50$ runs each):

- Variants of Differential Evolution (195 configurations),

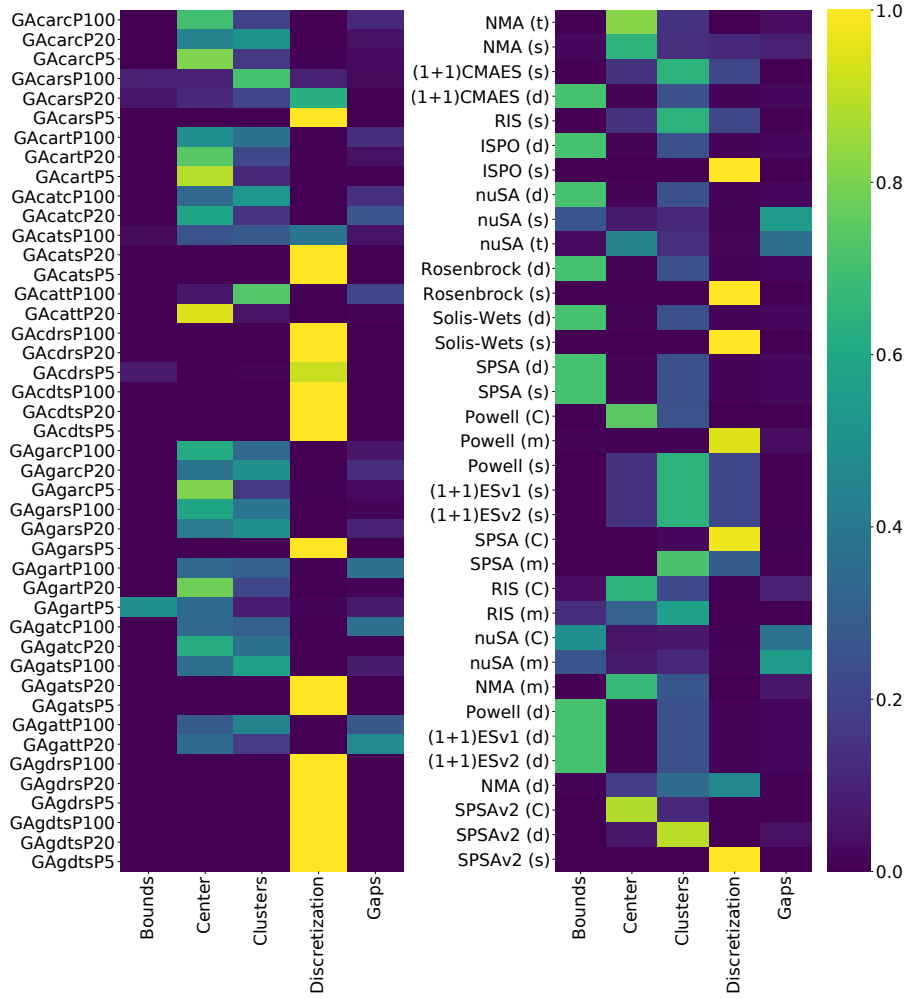


Figure 3.21: Predicted SB class probabilities of the biased GA configurations (left, sorted alphabetically) and the biased single-solution algorithms (right), using the random forest model. Names for the GA are structured as mutation–crossover–selection–SDIS–population size. For the single-solution algorithms, the character in brackets refers to the used SDIS.

- Compact optimisation algorithms (81 configurations),
- Single-solution algorithms (60 configurations),
- Variants of Genetic Algorithms (96 configurations).

For each of the considered algorithm configurations, we collect their final positions and feed these into the BIAS toolbox. In Figure 3.21 (left side), we show the outcome from the RF predicting the type of structural bias present in the different GA configurations (only the biased ones are shown). This shows that there are quite some differences in the detected bias, even within this limited algorithm design space. It is also interesting to note that the population size seems to have a relatively small impact on the type of predicted bias, which seems to be mostly impacted by the operator configuration.

For the single-solution algorithms, we see in the right part of Figure 3.21 that the strategy of dealing with infeasible solutions (SDIS) seems to drastically change the type of detected bias. For example, the Powell algorithm is classified as 'discretization' bias when using mirror strategy, while the classification changes completely with a COTN strategy. Such differences can give us useful insight into the effect of these SDIS methods on the optimisation behaviour of these algorithms.

DEEP BIAS In addition to the statistical approach used in the BIAS toolbox, we can make use of deep learning techniques to identify deviations from uniformity. To this end, we extended the BIAS toolbox with DEEP-BIAS: a convolutional neural network which detects both the presence and type of structural bias. This network structure is visualized in Figure 3.22.

By not going through the intermediary step of getting test statistics from the per-dimension test, this setup allows for a better classification of the type of bias detected, while remaining competitive with the statistical test in terms of bias detection, as is highlighted in Figure 3.23.

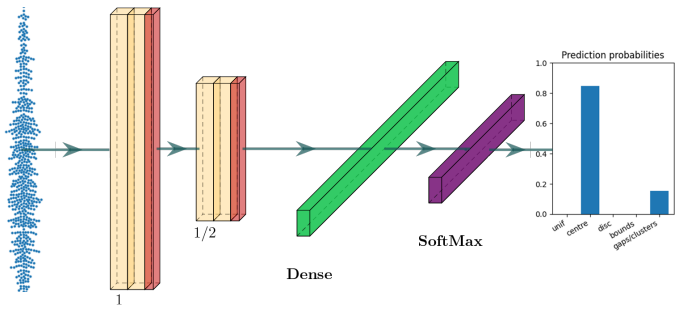


Figure 3.22: General one-dimensional CNN architecture with optimal hyper-parameters per sample size. The network takes as input a sorted distribution fixed sample size. Yellow layers are 1d-CNN layers, red layers are max-pooling layers, green layer is a dense layer and finally a classification head with SoftMax activation function resulting in five class probabilities per sample.

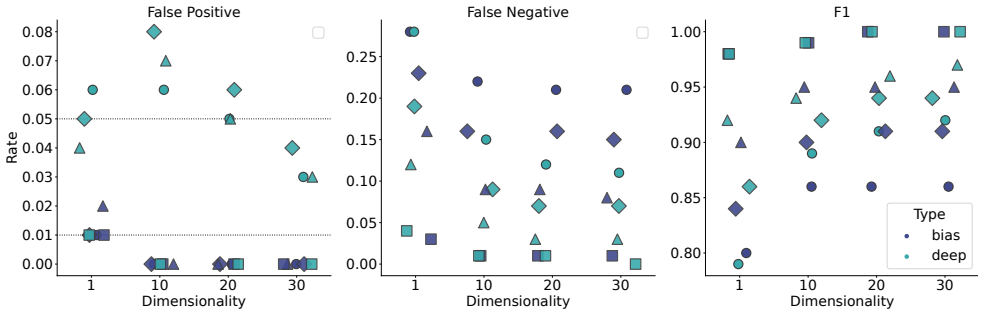


Figure 3.23: Comparison (with $\alpha = 0.01$) of the original BIAS toolbox (blue) and the Deep-BIAS (teal) in terms of false positives (left), false negatives (middle) and F1-score (right). On all figures, markers identify the used sample size: $\circ$, $\diamondsuit$, $\triangle$ and $\square$ are 30, 50, 100 and 600, respectively.

Chapter 4

Algorithm Configuration and Selection

In the previous chapter, we have seen that benchmarking plays a critical role in the understanding of optimization algorithms. By observing the performance of an algorithm on a variety of different problems, we can gain insights into its strengths and weaknesses. Critically, different algorithms naturally take advantage of different kinds of problem structures. A greedy hill-climbing algorithm for example is very efficient on a sphere-model, while it would perform poorly on highly multi-modal functions. Compare this to a purely random search, and it is clear that we want to choose a different algorithm to solve our sphere problem than our multi-modal example.

It should be noted that the notion of exploiting algorithm complementarity is not limited to the optimization context [217]. For example, in machine learning, complementarity between predictors is a large source of improvement, and top-performing frameworks for automated machine learning rely on the ensembling of complementary models to achieve state-of-the-art performance [76, 74]. Similarly, SAT-solving is a context in which the exploitation of algorithm complementarity has led to big improvements in the state-of-the-art [261, 104].

In general, *algorithm selection* aims to exploit these kinds of complementary strengths of algorithms by selecting a different optimization algorithm for each problem [205]. In the optimization context, algorithm selection attempts to find the best algorithm A from a portfolio $\mathcal{A}$ to solve a specific function f from a set of functions $\mathcal{F}$. Specifically, this static version of algorithm selection can be defined as follows:

Definition 4.1 (Static Algorithm Selection)**.** Given an algorithm portfolio $\mathcal{A}$ and a function $f \in \mathcal{F}$, we aim to find:

$$\arg\min_{A \in \mathcal{A}} \text{PERF}(A, f) \ ,$$

where $PERF : \mathcal{A} \times \mathcal{F} \to \mathbb{R}$ is a performance measure (which assigns lower values to better-performing algorithms).

One key aspect of algorithm selection in the context of black-box optimization is the representation of the problem. The black-box nature of the functions implies that we have very limited information available to base our decisions on. While algorithm selection approaches based only on the available information (problem dimensionality, variable types, evaluation budget) are being developed [166], the amount of complementarity these approaches can exploit is naturally limited. As such, the most common setup is to spend part of the total evaluation budget to collect samples from the function and use information from these samples as the problem representation [174]. In this setup, an algorithm selector is a machine learning model trained on a dataset containing the performance of each algorithm from $\mathcal{A}$ on each function from $\mathcal{F}$, where the function is characterized by a set of features [124]. These features are usually extracted via Exploratory Landscape Analysis (ELA, see Section 2.5), which needs relatively low sample sizes to calculate large sets of problem characteristics [164].

In addition to algorithm selection, we consider the *algorithm configuration* scenario. In this setup, we have a parameterized algorithm or algorithm family, and we aim to find the parameter setting which performs best on the selected (set of) problem(s). Where algorithm selection exploits complementarity between the algorithms in the portfolio, algorithm configuration benefits from the sensitivity of an algorithm's performance to its parameter settings.

Definition 4.2 (Static Algorithm Configuration)**.** Given an algorithm A with parameterization Θ_A and a function $f \in \mathcal{F}$, we aim to find:

$$\arg\min_{\theta \in \Theta_A} \text{PERF}(A_\theta, f)$$

Where many algorithm selection techniques rely on the availability of a complete enumeration of all algorithms and functions, this is generally not feasible in an algorithm configuration context. Algorithm configuration is thus treated as an optimization problem in its own right, where noisy evaluations, mixed-variable and conditional

search spaces, and expensive evaluations are common. Nevertheless, applying algorithm configuration to optimization heuristics has been very successful, and a wide variety of specific tools have been developed for this purpose.

In this chapter, we illustrate how we apply modular design principles to create large parameterized search spaces for both differential evolution and evolution strategies. We then apply a state-of-the-art algorithm configurator to these spaces and show what insights we gain into the impact of different algorithmic components on the final performance in certain landscapes. We end the chapter by illustrating some of the limitations we still face, with a particular focus on the large variability in performance of algorithm configuration when applied to the modular CMA-ES.

This chapter is based on the following publications: [51, 240, 247].

4.1 Modular Algorithm Design

Many popular optimization algorithms have been well-studied over the last decades. This has led to significant improvement and allowed for a great deal of specialization to different types of problems. While these modifications are all interesting in isolation, the true impact they have on the state-of-the-art is often hard to assess. One particular reason for this arises from the fact that algorithms can be inherently challenging to implement. Inconsistencies in the description, ignored edge cases, and even potential bugs can have a significant impact on the behaviour of an algorithm and the interpretation of results [26, 16]. Issues such as these have raised questions regarding the reproducibility of research in computer science as a whole, and evolutionary computation is no exception [151]. Because of this, comparing different variants of algorithms can be difficult to do fairly. Since researchers often implement the underlying algorithm from scratch, to then add their proposed modification (and in most cases a selected set of other algorithm variants for comparison), clear comparisons are often hard to find.

In an ideal setting, the community would maintain standardised implementations of core algorithms and the proposed modifications would be compared against the same set of state-of-the-art algorithm variants. Unfortunately, this might still be an impossible goal. However, algorithm modifications can still be fairly compared, as long as they are implemented in one common framework. This can be achieved through modular algorithms. From one common core algorithm, the variants are implemented as modules that can easily be swapped out.

The ideas behind modular algorithms have been around for decades [32, 153, 156],

but although they have been shown to be extremely useful [65], their adoption in evolutionary computation has been relatively slow. In recent years, several new modular implementations of popular algorithms have been released, including the modular CMA-ES [51, 234] and the Particle Swarm Optimisation framework [34]. These works highlight the benefits of modular algorithms not only for fair comparisons, but also hint at the potential to study interactions between modules.

4.1.1 Modular DE

In this section, we propose a first step towards a modular version of Differential Evolution (DE), a heuristic originally introduced in [219] to optimise a single-objective real-valued fitting problem, and whose design took into consideration elements from evolutionary algorithms and swarm intelligence optimisation (see [40, 242] for some insights on these aspects) and a simple core mechanism based on computing difference vectors through linear combinations of candidate solutions. DE has been around for almost 30 years and its popularity means that a wide variety of modifications have been proposed over the years [49]. However, when comparing the benchmark data, the relative benefits of many of these modifications seem to vary widely. Our objective is to provide an initial analysis of the performance of a set of 14 independent modules. This does not cover the full space of DE variants, but nonetheless highlights the potential of modular algorithms to aid in understanding the contributions made by these algorithmic variations.

Included Modules

Similar to other heuristic optimisers, DE naturally lends itself to a reformulation as a modular algorithm made up of a number of connected modules/operators where every independently made choice for a module is fully compatible with all choices for other modules. In fact, previous work has shown the usefulness of considering these operators as independent modules, e.g. to rigorously analyse the impact of the crossover operator [36]. In this section, we use this modularity to create a framework which we call *Modular DE* where a full combinatorial range of modules is available for each algorithm component, see Table 4.1.

Initialisation

To create the initial population, we implemented several sampling strategies (`Sampler`, see Table 4.1). The most common is to create a uniform distribution across the

entire domain. Alternatives are to use other distributions or low-discrepancy sampling methods. We choose to include the Halton and Sobol sequences to represent low-discrepancy sampling and a Gaussian distribution (centred around the origin, with $\sigma = (U - L)/6$, where U and L are the upper and lower bounds, respectively) to represent other kinds of distribution. Furthermore, a previous study has proposed using an oppositional initialisation strategy [197] (`Opposition`), where each time we generate an individual for the initial population, we also generate its mirror image around the origin.

Mutation

The mutation operator has been the focus of many modifications of DE, see, e.g. [266, 73, 106, 49, 37]. To capture the most established mutation variations of the kind `x/y` (where x is the base vector and y the number of differences), and to give flexibility in adding new variants, we implement the mutation operator through the combination of 3 modules. The first two modules, namely `Base` and `Ref`, help define the strategy `x`. Note that the reference solution `Ref` can be set to `none`, while the `Base` solution is not optional. In this scenario `x = Base`. Conversely, when `Ref` is one of the admissible reference solutions displayed in Table 4.1, a scaled version of the vector directed from `target` to the reference point is generated and added to `Base`, i.e. `Base + F(Ref-target)`. Therefore, when `Ref` is not `none`, one obtains any of the classic strategies of the kind `x = target-Refs`, plus new ones by varying the base vector. The third module, namely `Diffs`, is used to set the number `y` of difference vectors.

In addition to this restructuring of the definition of the mutation operator, we implement the option of using `WeightedF`, which reduces F at the beginning of the search and then increases it towards the end [25].

One more modification makes use of an archive of external solutions, as done, e.g., in [266], where one of the solutions in the archive is chosen to be part of one of the difference vectors - a scheme that has been shown to lead to improvements in the past and is activated via the module `Archive`.

Table 4.1: Available modules and parameters for the modular DE, their type ('c' for categorical, 'i' for integer or 'r' for real) and their domain. The choices shown in bold correspond to the default settings. For the numerical parameters, the default values are added after their domain. The 'Shorthand' column indicates the names used for these modules in the figures throughout this paper.

Operation	**Module Name**	**Shorthand**	**Type**	**Domain**
Initialization	Base sampler	`Sampler`	c	{'gaussian', 'sobol', 'halton', **'uniform'**}
Initialization	Oppositional initialisation	`Opposition`	c	{true, **false**}
Mutation	Base vector	`Base`	c	{**'rand'**, 'best', 'target'}
Mutation	Reference vector	`Ref`	c	{**none**, 'pbest', 'best', 'rand'}
Mutation	Number of differences	`Diffs`	c	{**1**, 2}
Mutation	Use weighted F	`WeightedF`	c	{true, **false**}
Mutation	Use archive	`Archive`	c	{true, **false**}
Crossover	Crossover method	`Crossover`	c	{**'bin'**, 'exp'}
Crossover	Eigenvalue transformation	`EigenX`	c	{true, **false**}
Bound correction	Bound correction	`SDIS`	c	{none, **'saturate'**, 'unif-resample', 'COTN', 'toroidal', 'mirror', 'hvb', 'expc-target', 'expc-center', 'exps'}
Adaptation	F adaptation method	`AdaptF`	c	{**none**, 'shade', 'shade-modified', 'jDE'}
Adaptation	CR adaptation method	`AdaptCR`	c	{**none**, 'shade', 'jDE'}
Adaptation	Population size reduction	`LPSR`	c	{true, **false**}
Adaptation	Use JSO caps for F and CR	`Caps`	c	{true, **false**}
Parameter	Population size	λ	i	$\{4, \ldots, 200\}$ ($\mathbf{4+\lfloor(3\log(d))\rfloor}$)
Parameter	Scale factor	F	r	$[0, 2]$ (**0.5**)
Parameter	Crossover rate	CR	r	$[0, 1]$ (**0.5**)

Crossover

The classical studies in DE generally consider two types of crossover: binomial (`z=bin`) and exponential (`z=exp`) [193], where the names refer to the distributions used for the probability of exchanging design variables between `target` and `mutant`. Both these types of crossover are included in this work.

Furthermore, we also include the option of performing the procedure from [87], by activating the eigenvalues transformation module `EigenX`, which allows using the `bin` or the `exp` operator and still maintaining rotational invariant behaviour. This is obtained by producing a covariance matrix from the individuals that make up the current population and diagonalising it with the Jacobi method [54] to calculate the eigenvalues and eigenvectors. These are real-valued and form an orthogonal basis (since the covariance matrix is symmetric and surely diagonalisable) and are arranged in a matrix R used to rotate `target` and `mutant` before performing the crossover. Note that the obtained `trial` has to be transformed back to the original coordinate system. This is an easy task, as the conjugate matrix R^* is equivalent to R^T in this scenario. Therefore, the multiplication between the transposed transformation matrix R^T and the newly generated point returns the desired `trial`.

Boundary Correction

There exist several mechanisms for boundary correction in the literature that allow us to deal with infeasible solutions. The most used within the DE community can be found in [17, 134]. For the proposed modular DE framework, we selected a varied set of 10 strategies for box-constrained problems.

Parameter Adaptation

Most state-of-the-art DE variants make use of adaptive parameters. So, in the proposed modular framework we implement adaptation methods for the DE core parameters, namely F, CR, and λ. The simplest is `LPSR`, which linearly reduces the population size over time [24]. For F and CR, we implement the adaptation mechanisms of SHADE and jDE [23, 223]. For F, we add an additional mechanism which uses the mean of the memory, instead of generating a different distribution for each individual, in the SHADE's adaptation strategy.

One final option to change the adaptation process is to use JSO [25] caps for F and CR (`Caps`), which, once activated, caps the values of these two parameters with different thresholds depending on conditions on the used computational budget.

4.1.2 Modular CMA-ES

Similar to the modular Differential Evolution, we also consider a modular variant of CMA-ES. This framework is in large part a redesign of the Modular Evolutionary Algorithms (ModEA) framework introduced in [234]. The modifications focus on the CMA-ES family of algorithms, to such an extent that the design of other evolutionary algorithms is no longer possible, thus requiring the change of names. The new framework was dubbed the Modular CMA-ES (modCMA) and is available as an open-source Python package within the IOHprofiler [63] environment.[1]

To design the Modular CMA-ES, we use the implementation from the popular CMA-ES tutorial [90] as a starting point. This work provides a detailed description of the CMA-ES algorithm, including a practical guide to its implementation. From this basic design, we separate the CMA-ES in a number of functionally related blocks, in order to allow a customization of a specific part of the algorithm. This allows us to implement algorithmic variants of the CMA-ES as functional modules. From a user perspective, any of these modules could then be combined in order create a custom instantiation of the CMA-ES, by selecting an option for each available module.

In ModEA, eleven of such modules were already implemented. These were all reimplemented in the Modular CMA-ES, with a few changes to the structure of the options. Specifically, we removed the *Pairwise Selection* as a module. Instead, we incorporated this option in the *Mirrored Sampling* module as the option *Mirrored Sampling with Pairwise Selection*, converting this module from binary to ternary. This is done because the pairwise selection method is not suited for use without mirrored sampling [3].

We implemented a new module for performing boundary correction, and added five alternative options for performing step-size adaptation. These two extensions to the framework will be the focus of our analysis through out this work. This set of changes give us the following list of modules for the redesigned Modular CMA-ES:

1. **Active Update**: Bad candidate solutions are penalized in the covariance matrix update using negative weights [112]. Note that in [90], this is given as the default version, here we consider it to be optional.

2. **Elitism**: $(\mu + \lambda)$ - selection instead of (μ, λ) - selection.

3. **Orthogonal Sampling**: All the newly sampled points in the population are orthonormalized using a Gram-Schmidt procedure [254].

[1] https://github.com/IOHprofiler/ModularCMAES

4. **Sequential Selection**: Candidate solutions are immediately ranked and compared with the current best solution. If an improvement is found, no additional objective function evaluations are performed [28].

5. **Threshold Convergence**: A method for balancing exploration with exploitation, scaling the mutation vectors to a required length threshold, which decays over time [187].

6. **Step-Size Adaptation**: Supplementary to the default Cumulative Step-size Adaptation (CSA), Two Point step-size Adaption (TPA) [88] is implemented. TPA requires two additional objective function evaluations, used for evaluating both a shorter and a longer version of the population's center of mass. The version which shows the higher objective function value determines whether the step-size should be increased or decreased. Five newly added mechanisms for performing step-size adaptation are implemented.

7. **Mirrored Sampling**: For every newly sampled point, its mirror image is added to the population, by reversing its sign [3]. This can be turned on or off, or as a third option this module can be set to *Mirrored Pairwise Selection*, where only the best point of each mirrored pair is used in recombination.

8. **Quasi-Gaussian Sampling**: Instead of performing the simple random sampling from the multivariate Gaussian, new solutions can alternatively be drawn from quasi-random sequences (a.k.a. low-discrepancy sequences) [6]. We implemented two options for this module, the Halton and Sobol sequences.

9. **Recombination Weights**: Three options are implemented; 1) default weights (see [90]), 2) equal weights: $w_i = 1/\mu$, and 3) $w_i = 1/2^i + 1/(\lambda 2^\lambda)$ for $i = 1, 2, \ldots, \lambda$.

10. **Restart Strategy**: When the optimization process stagnates, the CMA-ES can be restarted using a restart strategy. Two strategies are implemented in addition to the default 'off' setting. IPOP [5] increases the size population after every restart by a constant factor. BIPOP [155] also changes the size of the population, but alternates between larger and smaller population sizes.

11. **Boundary Correction**: If candidate solutions are sampled outside the search domain, they can be transformed back into the search domain by applying a boundary correction operation. In Section 4.1.2, we describe six options for performing boundary correction which have been implemented.

Table 4.2: The modules available for the Modular CMA-ES. The numeric index for each module corresponds to the index used in the text of Section 4.1.2. Newly added modules/options are given in bold.

#	0 (default)	1	2	3	4	5	6
1	off	on	-	-	-	-	-
2	off	on	-	-	-	-	-
3	off	on	-	-	-	-	-
4	off	on	-	-	-	-	-
5	off	on	-	-	-	-	-
6	CSA	TPA	**MSR**	**PSR**	**xNES**	**m-xNES**	**p-xNES**
7	off	on	on w. PS	-	-	-	-
8	off	Sobol	Halton	-	-	-	-
9	default	$\frac{1}{\lambda}$	$\frac{1}{2^i} + \frac{1}{\lambda^2\lambda}$	-	-	-	-
10	off	IPOP	BIPOP	-	-	-	-
11	off	**UR**	**MCS**	**COTN**	**SCS**	**TCS**	-

In Table 4.2, an overview is given of all currently implemented modules and their options in the Modular CMA-ES framework.

Boundary Correction

In the original modEA framework [233], a boundary correction function taken from [141] was implemented, and always applied after each mutation. In some cases, however, this operator can degrade the performance of the algorithm quite drastically. We therefore decided to make the boundary correction optional, and to implement it as a module, for it to only be used when beneficial. A number of different boundary correction strategies were implemented, taken from [40]:

1. **None**: No correction is applied to infeasible coordinates of solutions.
2. **Uniform Resample (UR)**: Replaces all infeasible coordinates of a solution with new coordinates sampled uniformly at random within the search space.
3. **Mirror Correction Strategy (MCS)**: Mirrors all infeasible coordinates of a solution with respect to its closest boundary.
4. **Complete One-tailed Normal Correction Strategy (COTN)**: All infeasible coordinates are replaced with new coordinates inside the search space according to a rescaled one-sided normal distribution centered on the boundary.
5. **Saturation Correction Strategy (SCS)**: All infeasible coordinates is set to the closest corresponding bound.

6. **Toroidal Correction Strategy (TCS)**: All infeasible coordinates get reflected off the opposite boundary.

Step-Size Adaptation

We consider a number of alternative step-size adaptation mechanisms for the Modular CMA-ES. We take inspiration from [137], which provides a qualitative evaluation of multiple step-size adaptation mechanisms used in ES. In addition to the CSA and TPA step-size adaptation methods, which were already available, we added the following procedures:

1. **Median success rule (MSR)** [72]: The MSR mechanism adapts the step-size σ as follows: it firstly computes a success rate by checking the number of current individuals that are better than some user-defined quantile of the function values in the previous population, then accumulates such success rates in every iteration, and finally decides to increase the step-size if the cumulated value is bigger than $1/2$ and decrease it otherwise.

2. **Population success rule (PSR)** [154]: PSR determines the success rate of the current population using a rank-based approach. It firstly sorts all individuals in the current and previous population together, then retrieves the set of ranks of individuals belonging to the current iteration and the one for the previous iteration, and finally calculates the average rank difference between those two sets as the population success rate, which controls the step-size updates.

3. **xNES step-size adaptation (xNES)** [83, 260, 137]: This method calculates the length of each standardized mutation vector and subtracts from it the expected length of the standard Gaussian vector. The resulting difference is then scalarized using the same weights used in the recombination, which is finally fed into an exponential function to generate a multiplicative coefficient to modify the step-size.

4. **mean-xNES step-size adaptation (m-XNES)** [137]: This mechanism functions similarly to xNES, with the exception that it takes the standardized differential vector between current center of mass and the one in the previous iteration and compares it to the expected length of the standard Gaussian vector.

5. **xNES with log normal prior step-size adaptation (p-xNES)** [137]: This approach resembles the principle of self-adaptation for step-sizes, where λ trial

Table 4.3: Set of 11 commonly used DE variants and the way they are implemented in modular DE. Empty cells indicate default values are used.

Name/Author		Mutation Settings	F	CR	λ	Other Settings
L-SHADE	[223]	Base : target, Ref : pbest	adaptive		$18 \cdot d$	LPSR, Archive, AdaptF_CR : shade
SHADE		Base : target, Ref : pbest	adaptive		$10 \cdot d$	Archive, AdaptF_CR : shade
DAS1	[48]		0.8	0.9	$10 \cdot d$	
DAS2		Base : target, Ref : best	0.8	0.9	$10 \cdot d$	
Qin1	[195]		0.9	0.9	50	
Qin2			0.5	0.3	50	
Qin3		Ref : best	0.5	0.3	50	
Qin4		Ref : best, Diffs : 2	0.5	0.3	50	
Gamperle1	[82]	Ref : best, Diffs : 2	0.45	0.4	$2 \cdot d$	
Gamperle2		Ref : best, Diffs : 2	0.6	0.9	$2 \cdot d$	
jDE	[23]		adaptive		100	AdaptF_CR : jDE

step-sizes are generated from a log-normal distribution which takes the current step-size as its mean and each trial step-size is used to sample a candidate point. To determine the new step-size, this method calculates the weighted sum of the log-transformed trial step-sizes, where those assigned to their corresponding candidate points in the recombination.

4.2 Algorithm Configuration for Modular Algorithms

4.2.1 Results of Configuring ModDE

Experimental Setup

Experiment 1 In order to analyse the potential of modular implementation of DE, we recreate a set of 11 *known* versions of DE within our framework (referred to as *common* variants). These algorithms are shown in Table 4.3, where all non-default parameters are mentioned. In addition to this, we can create a set of 30 *single-module variations*: DE versions where all modules are set to their default value, except for one. As such, each non-default module option is enabled in exactly one *single-module* variant. For these *single-module* variants, we set $F = CR = 0.7$, and $\lambda = 10 \cdot d$, based on the recommendations of [146].

For each DE variant, we collect performance data on all 24 BBOB problems (Section 3.2), using IOHexperimenter [53] for data collection. We perform 50 runs per function, spread over 10 instances (5 independent runs per instance). We repeat this for dimensionalities $d \in \{5, 10, 20\}$, where we give each run a budget of 50 000 function evaluations.

To evaluate the performance of each algorithm, we opt to use the Empirical Cumulative Distribution Function (ECDF). In particular, we use a normalized *Area Over*

the ECDF Curve (AOC) as an *anytime performance* measure [93] (more details in Section 2.4).

Experiment 2 For our second set of experiments, we use the algorithm configuration tool irace [152] to tune the performance of the modular DE on the same set of BBOB problems. Each irace run uses a budget of 10 000 evaluations, where each evaluation corresponds to running a DE variant with the selected parameter setting. We use irace, with its first-test parameter set to 5, and the remaining parameters kept at their default values.

We perform 10 independent runs of irace on each function from the BBOB suite, for dimensionalities $d \in \{5, 10, 20\}$, where irace has access to the first 5 instances of the function. We set the targets for ECDF to 81 logarithmically spaced values between 10^8 and 10^{-8}. We use AOC as the target since it has been shown that the increased signal it captures relative to measures such as Expected Running Time (ERT) can lead to overall performance improvements, even when evaluating the result with a different measure [264].

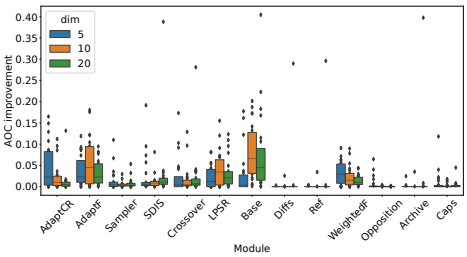
In addition to these per-function tuning runs, we also perform 10 tuning runs where we tune for aggregated performance over all the functions by setting the irace instance set to the 24 BBOB problems.

The resulting *elite configurations* for the across-function tuning are validated using the same settings as the DE variants from the first experiment: 5 independent runs on 10 instances of each BBOB problem. For the per-function tuning, we instead perform 5 independent runs on 50 instances of the problem on which the tuning was performed.

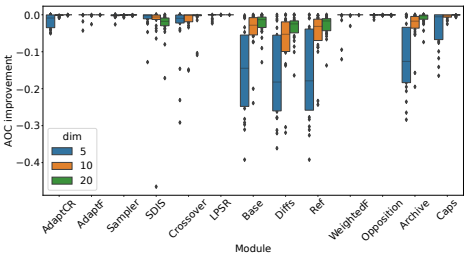
Single-Module and Common DE Variants

First, we investigate the *single-module* DE variants, which can be used to illustrate the impact of each module in isolation. We achieve this by comparing the performance of the default DE (all modules at their default value as seen in Table 4.1) to the variant with the identified best options enabled for each module. The resulting distribution of improvements is shown in Figure 4.1a.

From Figure 4.1a, we can see that some modules have relatively minor impact when the optimal option is selected independently from any other modules. This is the case for e.g. the number of difference components (`Diffs`) and the use of an archive population (`Archive`). In fact, if we instead consider the performance deterioration when making the worst choice for each module, these ones show a significant change over the default setting, as can be seen in Figure 4.1b. The combination of these two figures gives an overall importance of each module, in the sense that if only



(a) Improvement in AOC over the default setting when selecting the best-performing option for each module.



(b) Reduction in AOC over the default setting when selecting the worst-performing option for each module.

Figure 4.1: Impact of selecting the best (a) and worst (b) option for each individual module, measured as the difference in AOC relative to the default configuration.

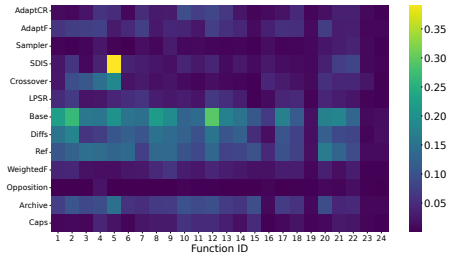


Figure 4.2: Importance of each module to the AOC on each of the 24 BBOB functions, aggregated over the used dimensions. Importance is calculated as the sum of absolute values from Figures 4.1a and 4.1b.

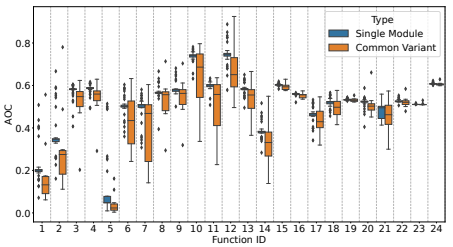


Figure 4.3: Performance distribution (AOC) of the 30 *single-module* DE variants and the 11 *common* DE variants from Table 4.3, for the 10-dimensional BBOB problems.

one module can be modified, some modules will likely have a much larger impact on the overall performance of the algorithm than others. The aggregation of maximum improvements and deteriorations for the selection of different module options is visualized in Figure 4.2. This figure shows the way in which these module importances are distributed across functions. For some functions, all *single-module* configurations perform similarly poorly, e.g. for F24, so no differences are detected. For most others, differences are present, with a clear impact on the choice of the base vector used for mutation (`Base`). In general, the mutation modules have relatively more impact than most others. Somewhat surprisingly, the impact of the adaptation methods for F, CR and population size λ is rather small. This might indicate that these settings work best when combined with other modules or more specific parameter settings. Also

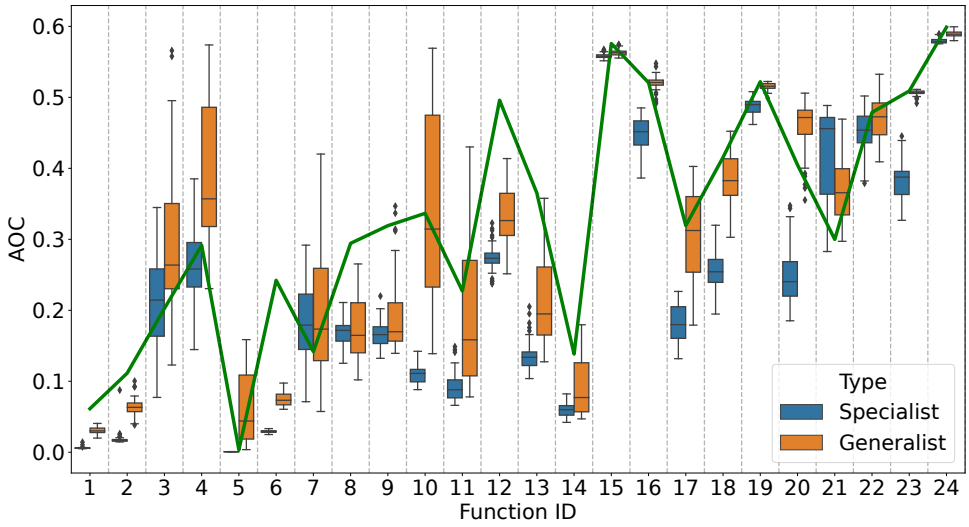


Figure 4.4: Performance distribution (AOC) of the configurations tuned for an individual function (*specialist*) and the configurations tuned for the full BBOB suite (*generalist*), for the 10-dimensional BBOB problems. The green line shows the AOC of the best DE version from the union of *single-module* DE and *common* DE variants from Table 4.3.

worth noting is that boundary correction is usually not impactful, with the exception of F5 (linear slope). For this function, the optimum lies directly on the boundary, so the boundary correction will be triggered often when close to the optimum, and thus have a large impact on the algorithm's performance [134]. All other BBOB functions are known not to have optima in the relative vicinity of domain boundaries [150].

To get insight into how hand-crafted DE versions, such as L-SHADE, compare to the *single-module* ones, we look at the performance distributions on the 24 BBOB problems. This is visualised in Figure 4.3. From this figure, we see that there is a fairly wide distribution of performance in both groups. Overall, the *common* DE variants seem to contain better configurations, although the set of configurations is relatively much smaller.

Performance of Tuned DE

Next, we compare the hand-crafted and *single-module* DE versions to those resulting from tuning the modular DE using irace. The resulting performance on the 10D BBOB problems is visualized in Figure 4.4. From this figure, we can see that generally, both

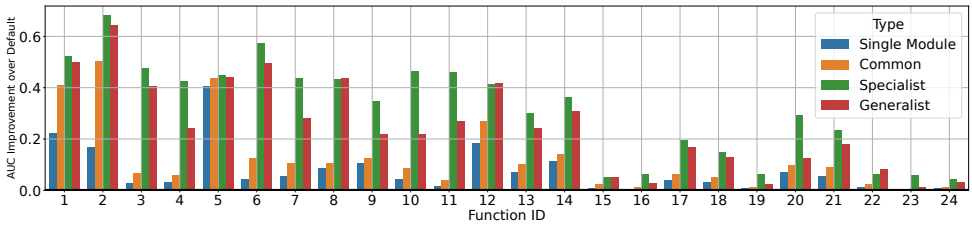


Figure 4.5: Relative improvement in AOC value between the best configuration of each type and the default setting, in 20D.

of the tuned DE settings outperform the hand-crafted ones. As expected, tuning for a particular function improves the performance on that function rather significantly.

Next, we aim to understand the impact of tuning relative to picking the best configuration from the set of *common* variants. To investigate this, we look at the relative gain in AOC over the default, for each set of configurations (*common* variants, *single-module* variants, *specialists* and *generalists*). For each type, we look at the performance of the best configuration of that type on each function and take the improvement it makes over the default setting. These improvements, for the 20D BBOB functions, are visualized in Figure 4.5. From this figure, we can see that the default setting performs particularly poorly on most of the unimodal problems, as even the best *single-module* configuration can outperform it significantly. However, this also shows the additional benefit which can be gained from tuning, which is particularly noticeable e.g. F3 and F4. We should also note that the performance gains shown here are slightly larger than those seen in Figure 4.4, which in turn are slightly larger than those achieved on the 5D version of these problems.

One more important note from Figure 4.4 is the wide distribution of AOC values. For the *generalist* configurations, this is natural, as configurations with different strengths can achieve similar performance when aggregated over the whole BBOB suite, resulting in a large per-function variance when grouped together. However, for the configurations tuned on a single function, the variance on some functions is still clearly visible. This might be caused by the inherent stochasticity of DE which potentially misleads the algorithm configurator when limited samples are available [247].

This variance might also explain why for F21 one of the hand-crafted DE variants outperforms almost all configurations which were tuned on that function. When considering Figure 4.3, we see that the performance might be considered an outlier, which performs much better than the remaining *common* variants. This observation might indicate that using the *common* DE variants to initialize irace could provide

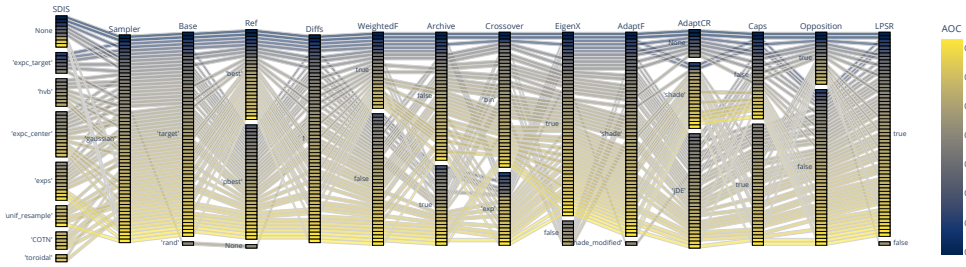


Figure 4.6: Parallel coordinate plot showing the modules activated in the elite configuration found across 10 runs of irace, on F19 in 5D. Configurations are colored based on normalized AOC.

some additional benefits over the current random sampling.

Analysis of Elite Configurations

Since multiple repetitions of irace are performed for each problem, we have a set of between 10 and 50 elite configurations for each setting. By analysing the commonalities between these elites, we can get an overview of the benefit of different parameter settings. This can be done on a global level by aggregating the activations of certain module options across runs and dimensionalities.

To understand which modules are selected often, we consider the module activations of a single function and visualise them as a parallel coordinate plot. Figure 4.6 shows this for Function 19 in 5D. In this figure, we see that all elite configurations make use of a Gaussian sampler for initialisation (`Sampler`). This makes sense when we consider the properties of F19 in more detail. In particular, we should note that for this function, the location of the optimum is not uniformly distributed in $[-4, 4]^D$ as for most BBOB problems, but it is instead limited to the shell of the hypersphere of radius 1, centred at the origin [150]. Because of this, a Gaussian initialisation will significantly outperform any uniform or low-discrepancy initialisation strategy.

In Figure 4.6 we also observe that all configurations, except one, make use of the SHADE-based adaptation for F (`AdaptF`), with 'target' based mutation mechanism (`Base`). This suggests that, unlike the common belief of adding many components in the mutation operator to deal with such problems, adaptation systems based on the history of successful control parameter values are beneficial for multimodal problems similar to F19, especially when combined with 'target'-based mutations and `Diffs`$= 1$.

4.2.2 Incremental Assessment of Module Performance: ModCMA

While the expansion of modular algorithms leads to an exponential increase in the number of possible configurations, we can still use algorithm configuration techniques to gain insight into the combinations of modules which perform well. When new modules are added we can retain the performance data from earlier experiments, and incrementally build upon this. Rather than looking at the new module in isolation, we use our algorithm configuration setup with the expanded search space and compare the resulting high-performing configurations to find interactions resulting from the modules inclusion.

We propose the following roadmap to formalize this procedure, which is designed to be generic, so that it can function with any modular algorithm, hyperparameter tuner, and performance metric:

1. Select a modular implementation of the base algorithm to which the new module has been added, a hyperparameter optimizer and a performance metric.

2. Collect a list of the existing modules and relevant hyperparameters (without the new module to assess). This will be the search space for the hyperparameter optimization.

3. Run the selected hyperparameter optimizer on this search space, ideally for a wide set of relevant benchmark functions. This data will then serve as the baseline performance.

4. Extend the original search space by including the new module to assess, and run the hyperparameter optimization on this extended search space (using the exact same setup as the baseline).

5. Compare the data from the baseline to the experiment with the extended search space. This should not only be done from a performance perspective, but also from the resulting configurations themselves. This allows for the analysis of potential interactions between modules.

Experimental Overview

To illustrate our proposed approach, we make use of the modular CMA-ES framework introduced in Section 4.1.2. Specifically, we consider the stepsize adaptation and

boundary correction, which were added on top of the previously implemented modules as shown in Table 4.1.

For our experiments, we stick with irace as our algorithm configurator. Four runs of irace are performed for each of the 24 objective functions in the BBOB single objective noiseless problem suite [96, 95], of which the first 5-dimensional function instance is used. Each run of irace is given a budget of 1 000 algorithm evaluations, which themselves have a budget of $10\,000 \cdot d$ function evaluations. We use the AOC attained by a run of a given configuration as the objective function value. Irace will designate one or more configurations as elites, which are the best configurations found. We validate the performance of these elite configurations by performing 25 validation runs, with the same random seeds for all configurations. We use the results of these runs to assess the final performance.

Following our roadmap, we define a baseline by tuning the existing modules from modCMA, which are shown in Table 4.2. In addition, we tune four continuous hyperparameters c_1, c_μ, c_c, and c_σ, which control the dynamics of the adaption of the covariance matrix (c_1, c_μ, and c_c) and of the step-size (c_σ).

We compare two experiments to our baseline where in addition to the existing modules, 1) several new step-size adaptation methods (see Section 4.1.2) are included, and 2) a new boundary correction module (see Section 4.1.2) is added to the tuned parameters. Both of these experiments use the same experimental setup as the baseline experiment (excluding the tuned parameters). Note that in the boundary correction experiment, the new step-size adaptation methods cannot be selected and vice versa.

Single Module Performance

Before considering our proposed method, we run a basic benchmarking experiment on each of the individual module options (including the new options). This is similar to the common approach of benchmarking a new module against a set of other algorithm variants. We show the resulting best single-module configurations (a.k.a. the virtual best solver, VBS for short) relative to the default CMA-ES in Table 4.4. In this table, we see that among the new modules, only two have been selected: MSR for F23 and m-XNES for F5. We can further look at the overall contributions of the newly introduced step-size settings by plotting the ECDF-curves over all functions, as done in Figure 4.7. In this figure, we can clearly see that most methods are quite competitive, with the only exception being xNES, which has an overall worse performance than the others. Overall, the MSR method seems to be quite effective, but there is no strict domination over the other settings.

Table 4.4: Table showing the AOC of the best single-module configuration for each function (VBS), compared to that of the default CMA-ES. The name of the solver corresponds to the module which is active, e.g. <module_name>_<option_value>. Note that these values does not include benefits from tuning the continuous hyperparameters, which are set to the default values for all configurations in this table.

Fid	VBS	AOC of VBS	AOC of Default	Improvement
1	elitist_True	247	326	24%
2	active_True	1 272	1 659	23%
3	local_restart_BIPOP	38 374	44 518	14%
4	local_restart_IPOP	41 746	44 613	6%
5	step_size_adaptation_m-xnes	43	63	31%
6	elitist_True	655	904	28%
7	step_size_adaptation_tpa	1 312	39 199	97%
8	base_sampler_halton	1 186	4 544	74%
9	base_sampler_sobol	959	2 470	61%
10	active_True	1 309	1 729	24%
11	active_True	1 162	1 749	34%
12	base_sampler_sobol	2 186	2 980	27%
13	active_True	1 627	2 191	26%
14	active_True	601	831	28%
15	local_restart_BIPOP	30 380	43 313	30%
16	local_restart_BIPOP	8 172	34 132	76%
17	threshold_convergence_True	12 464	26884	54%
18	threshold_convergence_True	15 764	33724	53%
19	mirrored_mirrored	33567	36 688	9%
20	threshold_convergence_True	36 482	40691	10%
21	local_restart_IPOP	38 028	40 371	6%
22	mirrored_mirrored	566	8 632	93%
23	step_size_adaptation_msr	11 060	34 433	68%
24	local_restart_IPOP	42 099	44 351	5%

Analysis and Results

In this section, we present the results of our hyper-parameter tuning experiment. We consider two paths to analyze the contributions of the newly introduced modules: the performance-perspective and the perspective of the selected modules. We start by examining our baseline. This is followed by an analysis of the performance-perspective and a deeper analysis of the selected modules.

Baseline

As mentioned in Section 4.2.2, we conduct a baseline tuning experiment.

Since we run four runs of irace for each function, this results in 4 sets of elites (each set has up to five configurations), for which we then perform the verification runs. We plot the distribution of the AOC for each of these configurations in Figure 4.8, in addition to this, the AOC of the default CMA-ES and the VBS is shown. From this figure, it is clear that the tuning of all parameters at once is much better than simply selecting a single-module variant, as is to be expected. This plot also highlights the

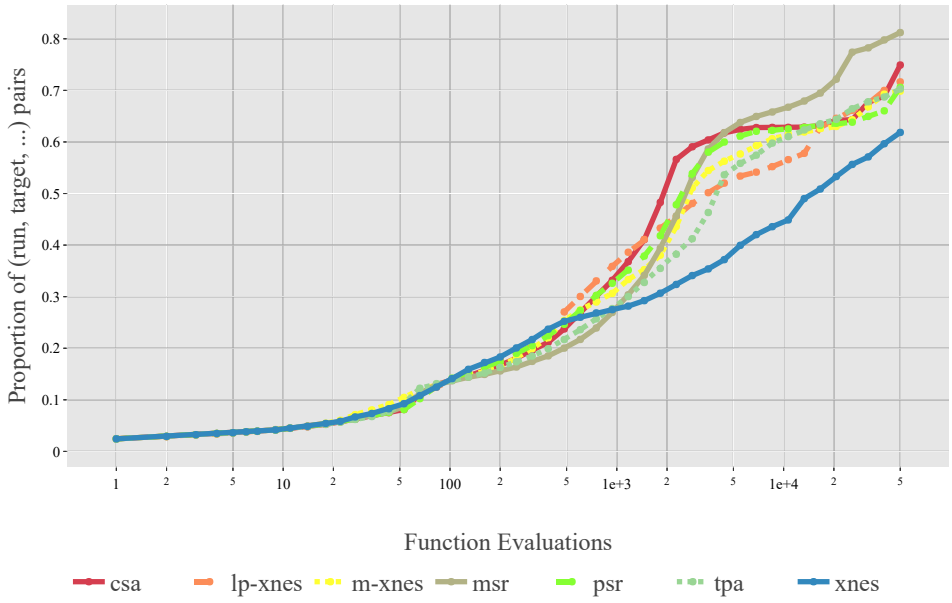


Figure 4.7: ECDF-curve of all single-module stepsize options. Figure generated using IOHanalyzer [255].

variance in performance of the final found configurations. There are two main reasons for this fact: the inherent stochasticity of the CMA-ES itself, and the large impact of the initially generated configurations of irace. We discuss these challenges in detail in Section 4.3.

From this baseline data, we can also study the resulting configurations themselves. This can be done by aggregating the modules which have been selected in the final elite configurations in the separate irace runs, as is visualized in Figure 4.9. In this figure, we can see that there is a large variability in the selected module options, which seems to indicate that they are all usable for at least some functions. One notable exception is the weights option "equal", which is chosen in less than 1% of configurations.

Performance analysis

First, we visualize the distributions of the AOC of the single best configuration found in each run of irace (based on the verification runs) in Figure 4.10. In this plot, we can see that the effect of introducing the new modules is quite mixed. For some functions, performance decreases (e.g., on F8) after introducing new modules, while for others

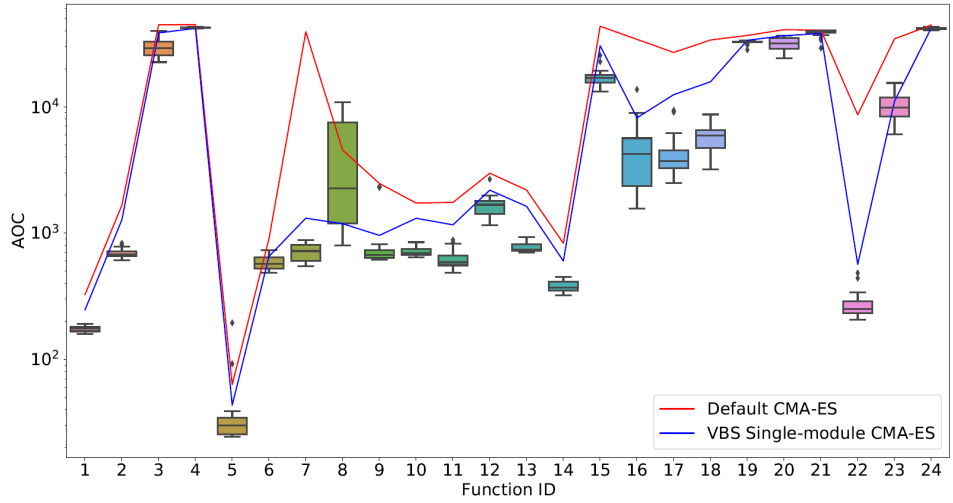


Figure 4.8: Distribution of the area over the ECDF curve for the final elite configuration of the baseline irace runs. All AOC's are averages of 25 verification runs. The VBS single-module configurations can be seen in Table 4.4.

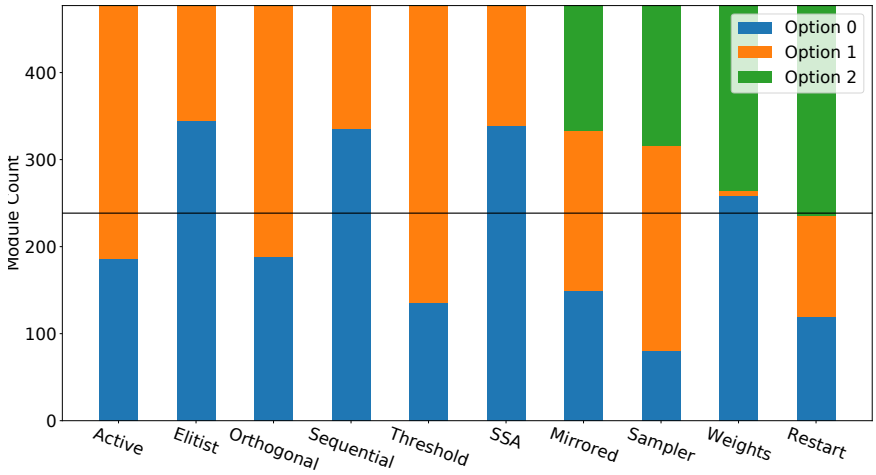


Figure 4.9: Module counts of all elites found in the baseline-experiment, over all 24 BBOB-functions. The option numbers correspond to those in Table 4.2

we see the desired improvement (e.g. on F23).

In order to better show these differences, we show in Figure 4.12 the AOC of the single best configurations found in both the SSA and bound-experiments relative to

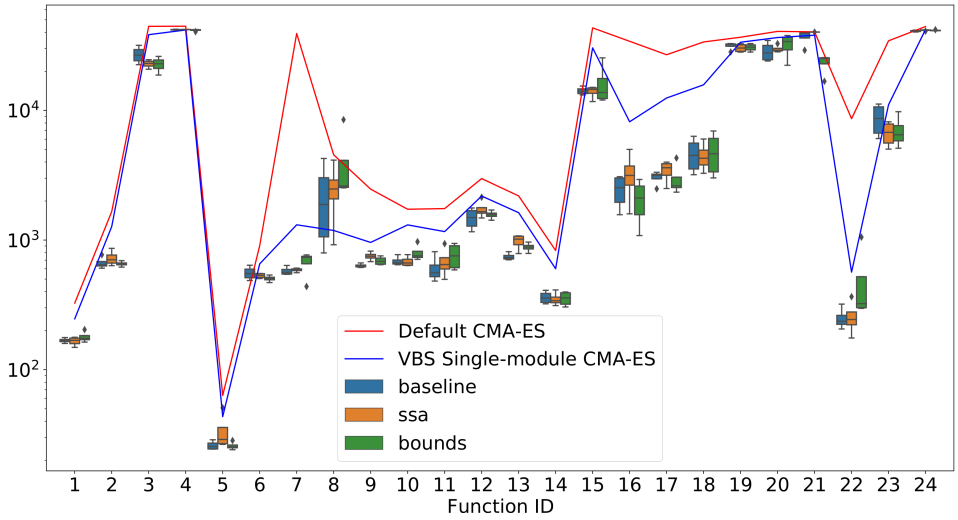


Figure 4.10: Distribution of the single best elites from the baseline and the tuning with the additional modules. AOC values are the result of averaging over 25 verification runs.

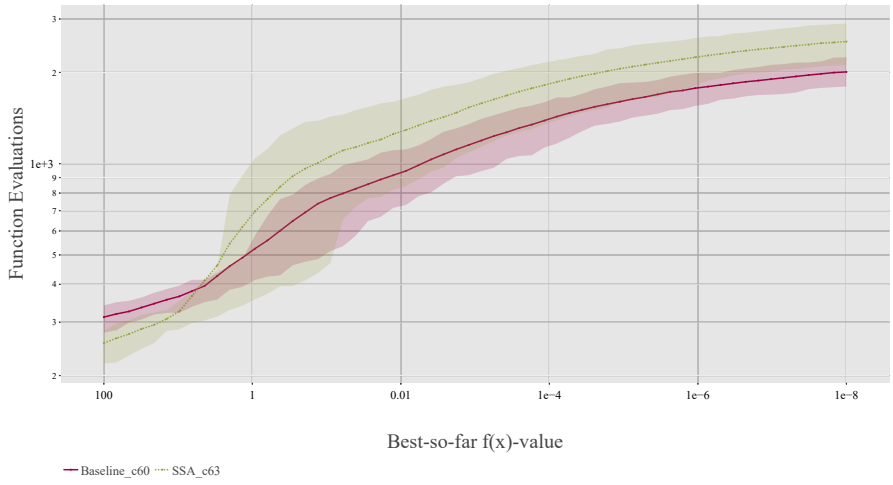


Figure 4.11: Comparison of the Expected Running Time of the best configurations found on F12 by both the baseline and the SSA experiments. Shaded areas indicate the outer quantiles (20-80).

the best configuration from the baseline. Here we observe a generally negative trend, with outliers in both directions. This seems to indicate that these new modules are not always beneficial to the final performance. For example, we can consider F12, where the configuration found by the baseline has an average AOC of 1 159, while the best configuration found when including the new SSA-methods in the search space reaches an average AOC of 1 480. We show the expected running time of these two configurations in Figure 4.11, where we can clearly observe this difference. However, we can observe a large variance between runs, which can partly explain poor performance. Indeed, if we look at the average AOC as found during the irace run (instead of the later verification runs), the difference between these two configurations is only 7%, even though the distance between them in the verification runs is much larger. This leads to an important observation about the assessment of the new algorithmic modules: when judging results purely from the average performance measures, it is necessary to also consider the overall variability of the experiment, as well as the inherent stochasticity of the base algorithm.

We perform the same procedure for the boundary correction methods. The impact of this module is expected to be smaller, since for most of the "easier" functions, the boundary condition is rarely violated. For some of the more challenging functions however, the penalty value given by BBOB function itself might not be sufficient to "guide" the algorithm back in bounds, but an explicit boundary correction could be beneficial in these cases. We can see that this seems to indeed be the case in Figure 4.12, where on the more complex functions, e.g., F21, the performance is improved when the boundary correction module is tuned.

In Figure 4.12, we also see that the inclusion of the new SSA methods manages to improve the overall performance for some functions. As an example, on F23 we saw an improvement of 17.1% over the best baseline configuration. If we consider all four elite configurations and compare the average performance differences, the average improvement is even higher, at 22.3%. The stability of this improvement is promising, but in order to fully grasp how the inclusion of the new SSA mechanisms leads to this improvement, we need to analyze the selected modules across these different experiments.

Module Analysis

We have seen that the performance of the elite configuration found on F23 improves when we include the new SSA modules in the search space. In order to identify what this performance can tell us about the new modules themselves, we should study the

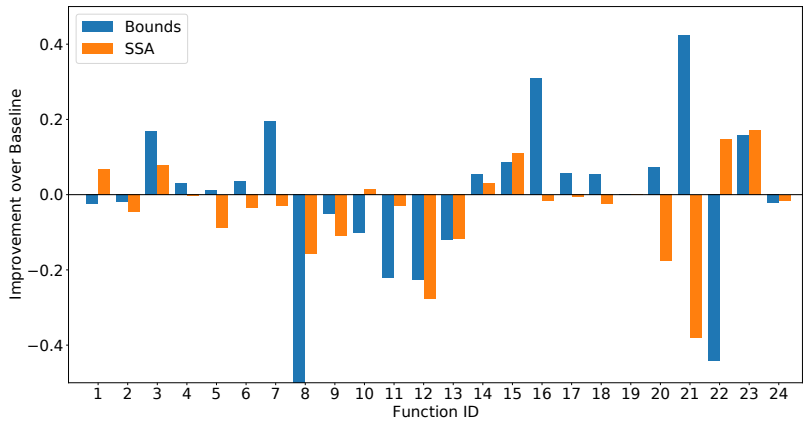


Figure 4.12: The relative improvements per function of the best configuration found by irace relative to the baseline experiment's best configuration.

configurations in more detail. The obvious way to see the difference is by looking at how often the new module options have been selected in the final elite configurations. Over 20 elites, the PSR update was selected 14 times, MSR once, and CSA five times. This shows that these new modules are indeed used in successful configurations. To see how the inclusion of these module options changes the interactions with the other modules, we look at the combined module activation plot, which is shown in Figure 4.13. From this figure, we can see that there are some interesting differences between the two sets of configurations: the options for the restart and mirrored module are not as uniform when using the new SSA methods, and the weights option is changed completely. These observations show that there is a clear interplay between these modules.

We can extend this module analysis to all functions by aggregating the most important differences found between the baseline and SSA-experiments. First, we can plot how often each new module option is selected in the elites for each function, as is done in Figure 4.14. We can use the same principle to study the interaction with the other modules. For the binary modules, we can directly capture the module difference by looking at which modules occur more or less often in the final set of elites, as is visualized in Figure 4.15. From this figure, it becomes clear that the elites on some functions are barely affected by the inclusion of the new modules, while others require completely different module settings to properly exploit the changed search space.

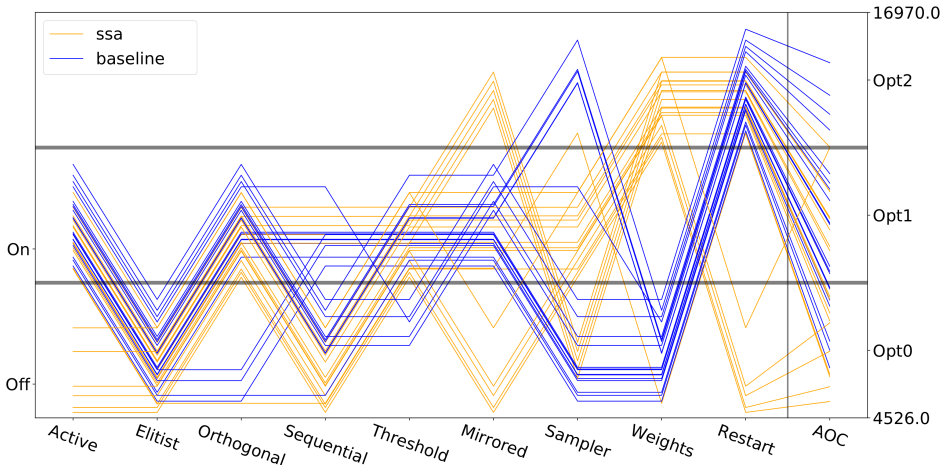


Figure 4.13: Combined module activation plot for the elites found in the baseline and SSA experiments, for function 23. The lower the line, the better its performance, scaled within each band according to the AOC. The option numbers correspond to those in Table 4.2.

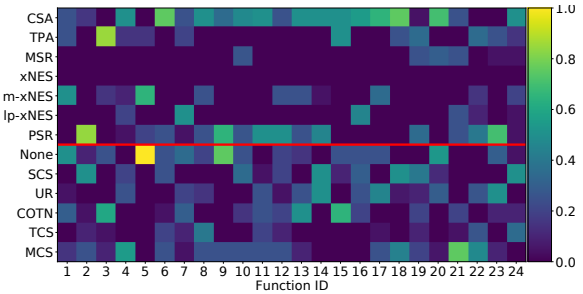


Figure 4.14: Heatmap showing the fraction of the elite configuration in which each of the options for either SSA (top) or boundary correction (bottom) are active.

4.3 Selected Challenges in Algorithm Configuration

While Section 4.2 highlights some of the benefits of algorithm configuration in the context of modular algorithms, our approach still faces some inherent limitations. In this section, we discuss some selected challenges, and illustrate specifically how the inherent stochasticity of the iterative optimization heuristics impacts the results of algorithm configuration.

The cost of tuning The first thing we should note about the incremental tuning approach is that only considering the final elite configurations does not tell the full story of a module's contribution. As noted previously, introducing a new module

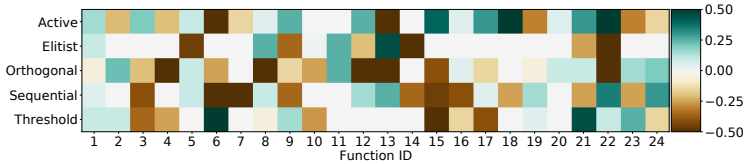


Figure 4.15: Heatmap showing difference in the fraction of the elite configuration in which each the of the binary modules are active, between the baseline and the SSA experiment. Positive values indicate a module is turned on more often in the SSA experiments.

increases the size and complexity of the search space, which has a large impact on the hyperparameter tuning task. If a module is very dependent on the settings of other hyperparameters, this can lead to deterioration of the final results, since the initially sampled configurations are likely to have worse performance than those in the baseline. This is visualized in Figure 4.16, where this is clearly seen on function F5. This is a linear slope function, but the BBOB-specification does not include a sufficient penalty for leaving the search space. As a result, an algorithm which quickly leaves the search space will reach the required objective value very quickly. Thus, when adding boundary correction methods, five out of six random configurations are not able to abuse this loophole, leading to a worse initial performance. While for F5, the function is simple enough that the good configurations can still be found (and the inclusion of the default CMA-ES settings in the initial population means that there is always at least one good configuration present), the same issue exists to a lesser extent in other functions. Figure 4.16 also shows that the "tunability" of modules on different functions varies widely. For instance, on functions F16 - F18, the spread of AOC values is significantly larger than those on functions F19 - F21, suggesting that it is relatively more difficult to tune the modules in the latter since the tuner will very likely take a considerably larger budget to identify optimal configurations. Also, while on some functions it is trivial to get improvement (e.g., F7) over the default CMA-ES, it is a lot more challenging on others, for example on functions F16 - F18.

Limits of the per-instance analysis: As is commonly done, our performance assessment is done on a per-instance basis. While this can be preferred over tuning for large sets of functions/instances [9], it does have some drawbacks. Specifically, if a module is designed to have a good performance over a wide set of functions, but other settings exist for each individual function which outperform it, this new module would not be seen as beneficial. Because of this, we argue that module assessment

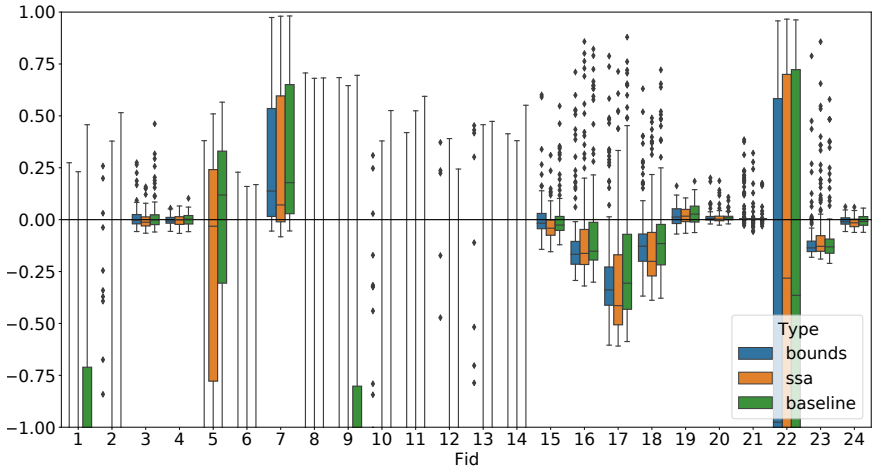


Figure 4.16: Distribution of the relative AOC values found in the initial race of irace (relative to the default CMA-ES configuration; positive values equate to lower AOC.)

by hyperparameter tuning should not replace the traditional assessments, but rather complement it for more in-depth, per-instance analysis.

Influence and stochasticity of the hyperparameter tuning: While we showed that assessing the impact of an algorithmic component by using a hyperparameter tuning approach provides useful insights, there are several factors which can complicate this approach. Since hyperparameter tuning is a very challenging problem, with many different approaches to solving it, the kind of tuner used will have a large impact on the resulting assessment [221]. In this chapter, we used irace, which tends to focus on converging to a single configuration, instead of covering a large set of different solutions. This necessitates running multiple repetitions of the irace procedure itself, as the initialization might otherwise have too much impact on the final configurations. This can quickly become computationally expensive.

4.3.1 Noise in Algorithm Configuration

The final, and perhaps most relevant, limitation we discuss is the inherent stochasticity in the algorithm which we are using. The amount of variance of the algorithm configurations on a certain function has a large impact on the search procedure of irace. Since we generally end up selecting elites based on average performance, we are inherently underestimating the AOC of the final configuration. Even though irace largely mitigates this by using statistical testing in the races to decide when to discard

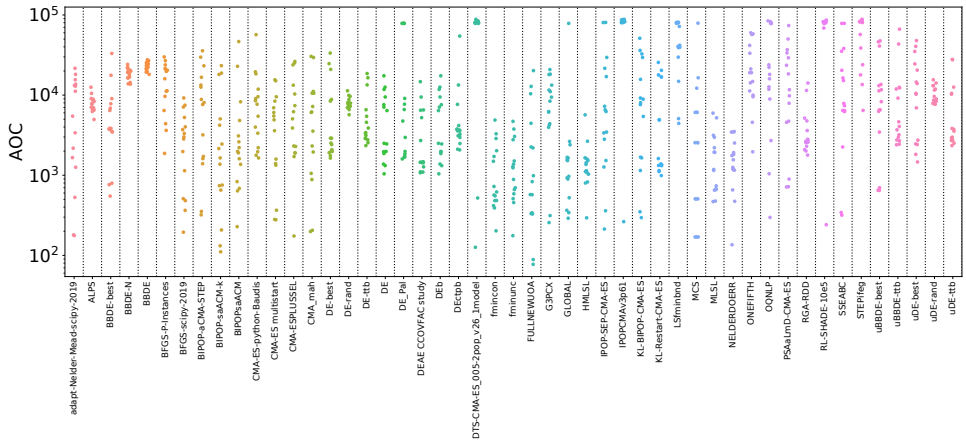


Figure 4.17: Distribution of the AOC values of 15 independent runs of algorithms from the BBOB archive [4] on F21 in 5D.

configurations, there will always be some degree of underestimation of the performance (for example, the median performance in the verification runs from our Section on modCMA is 3.4% worse than predicted from the irace runs).

In this section, we highlight this challenge inherent in comparing the performance of stochastic optimization algorithms. While our focus is on algorithm configuration methods, we show that, for several cases including standard benchmarking-based comparisons, the currently recommended values for the number of samples and testing procedure can lead to mistakes. We show that the distribution of performance values has a large impact on algorithm configuration methods, indicating that there is not one method of performance comparison that dominates all others. Importantly, our results demonstrate that we must identify better ways to handle the stochasticity of iterative optimization heuristics when applying algorithm configuration methods.

Why 15 runs are not enough

While it is clear that any aggregated performance measure used to compare randomized algorithms is an empirical estimation of their true performance, the variance of this estimation is not necessarily equal for all algorithms on all functions. However, for a practical benchmarking setup, this nuance is often ignored in favour of simpler guidelines, such as aggregating a fixed number of *samples* (i.e., individual performance values from independent runs) for each algorithm on each function. The usual recommendation of 15 samples [95] is often enough to make clear decisions on simple

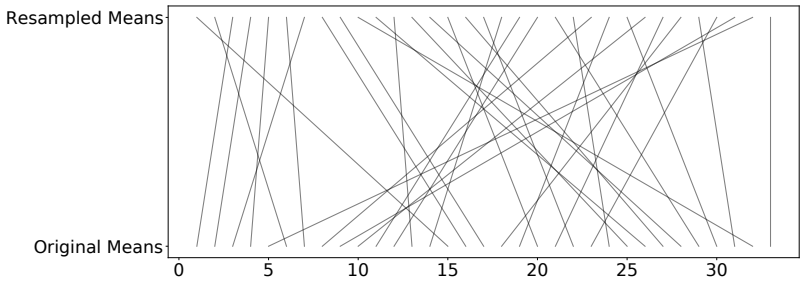


Figure 4.18: Changes in ranking of 33 **random modCMA configurations** based on calculating the mean over a sample of 15 AOC values (*Resampled Means*) versus the 200 verification run samples (*Original Means*), on F21.

uni-modal functions, but the situation is much less clear on more challenging optimization problems.

We illustrate the significant variation in performance between runs by showing in Figure 4.17 the distribution of 15 independent AOC-values for a wide variety of algorithms from the BBOB-repository (https://numbbo.github.io/data-archive/bbob/) on F21 in 5D. This figure also shows that the normality assumption, commonly taken for granted in benchmarking studies, is not well supported by the apparent distribution of the 15 performance values shown for each algorithm.

For some algorithms, the performance distributions even appear to show signs of bi-modality. As such, any analyses made based on this set of samples should be treated with care. While this large amount of variance is very pronounced in F21, it is not limited to this function, as other functions display similar effects but to a slightly lesser extent.

The impact of performance variability can potentially be even larger when considering the task of algorithm configuration. It has previously been observed that the performance of an algorithm configuration on verification runs can differ significantly from the runs performed during the configuration task [51].

We illustrate this effect by showing in Figure 4.18 the changes in the ranking of the 33 **high-quality modCMA configurations** described in Section 4.1.2 when calculating mean performance using a small sample size (15) and a larger number of verification runs (200) on F21.

While this might be considered a rather extreme case, it is by no means the only scenario in which behaviours like this can occur. Since algorithm configuration often generates similarly performing configurations near the end of a configuration run (while exploiting promising regions), making decisions about which configuration to select

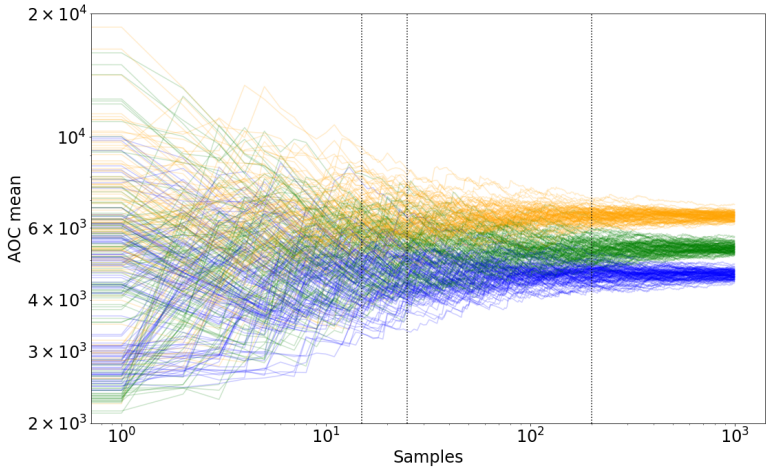


Figure 4.19: Evolution of the cumulative mean over sample sizes of 3 selected **high-quality modCMA configurations** on F18. The vertical lines indicate sample sizes 15, 25 and 200 respectively. Means are based on sampling with replacement from the original 200 samples of each configuration.

might become very noisy when using relatively low sample sizes. This phenomenon is exemplified in Figure 4.19, where we show the evolution of the mean AOC of 3 selected **high-quality modCMA configurations** relative to an incremental number of AOC values. Each horizontal line of the same color corresponds to the cumulative mean of a sequence of values sampled with replacement from the same 200 AUC values. Despite sampling from the same 200 AUC values, the variance of the means of 15 and 25 samples is quite large and those means often poorly estimate the true mean performance.

In practice, making an incorrect decision between two configurations matters less when their true performance is very similar. However, when the set of configurations which are being compared increases in size, the risk of making incorrect decisions between more distinct configurations could potentially grow as well. In some situations in algorithm configuration tasks, we have observed significant differences between the performance of the selected elite configurations, and the best one from all configurations sampled according to the verification runs.

In Figure 4.20, we show the distribution of AOC values for each configuration sampled during a run of irace on each of the 24 BBOB functions. The performance of each configuration is based on the mean of 200 verification runs, and the plot

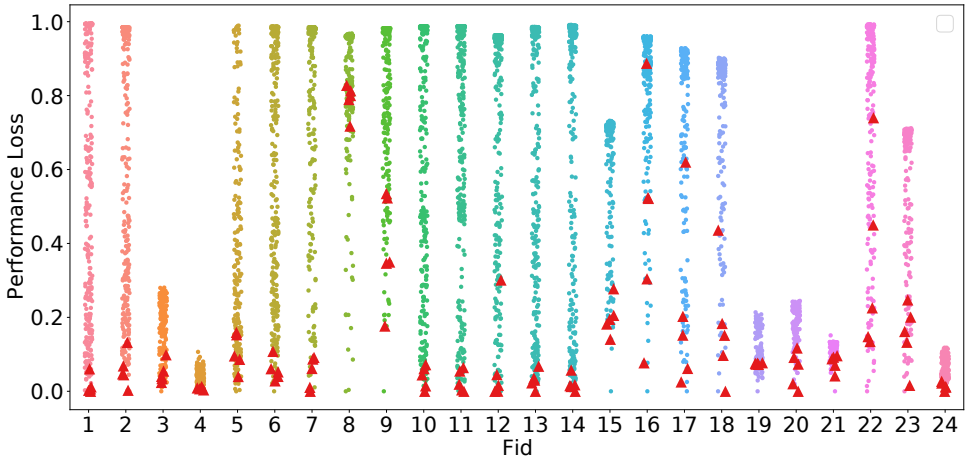


Figure 4.20: Performance losses between all modCMA configurations explored during the execution of one irace run on each function, and the best one from this set of configurations. The final elites of each irace run are marked in larger red triangles. All data points shown are based on 200 independent samples.

shows the relative performance loss to the best of these means. The (up to five) elite configurations returned by irace are marked with a red triangle. The lowest of these elites corresponds to the level of performance loss achieved by irace compared to the best-performing configuration sampled during the configuration process. From this figure, it can be seen that for some of the more complex functions, a 10% performance loss or more can occur, clearly demonstrating that the variability of performance can severely hinder the outcome of the configuration efforts.

Impact on Benchmarking

To simulate a common algorithm comparison scenario, we make use of the set of 33 **high-quality modCMA configurations** from Section 4.1.2 and simulate the benchmarking procedure by randomly re-sampling with replacement AOC values, for sample sizes 2, 5, 10, 15, 25 and 50 from the set of 200. Then, we select the configuration with the best mean for each particular sample size as the winner, and compare its true performance (i.e., over 200 runs) to that of the actual best configuration to get an estimate for the performance loss. This process is repeated 5000 times for each sample size and each function, and the resulting performance loss per function is shown in Figure 4.21. We conclude that using means to determine the best-performing algo-

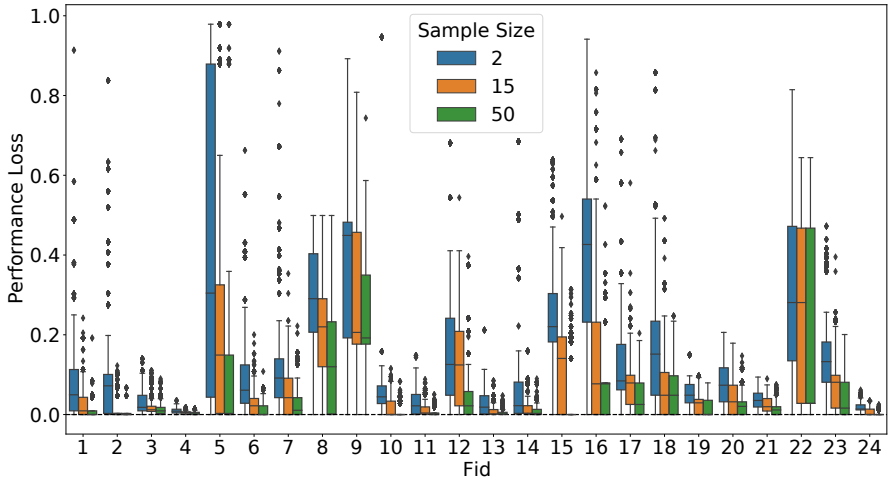


Figure 4.21: Performance loss, relative to the configuration with the best mean calculated over 200 samples, when comparing 33 **high-quality modCMA configurations** based on mean calculated from different number of samples. Each bar represents 5000 repetitions of the experiment.

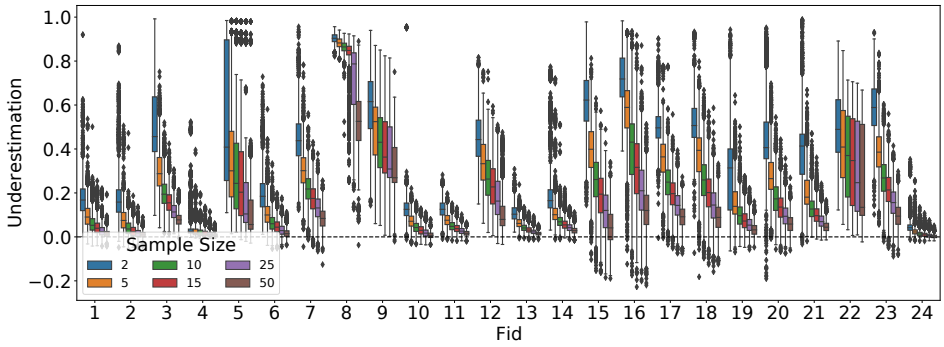


Figure 4.22: Underestimation when comparing 33 **high-quality modCMA configurations** based on mean calculated from different numbers of samples. Each bar represents 5000 repetitions of the experiment.

rithm is not always reliable, and can lead to selecting configurations that are clearly sub-optimal. Many benchmarking studies use non-parametric statistical tests to assess significant differences without assuming normality, yet they still rely on the comparison of means to rank the algorithms. While we can see that increasing the used sample size is always beneficial, even using as many as 50 samples can still see performance losses of 10% and more on some functions.

One possible explanation for these results is that, when we are determining the best algorithm from a large set of algorithms of wide performance variability, our decision is prone to underestimate the true mean due to the small sample size, i.e., we might "luckily" sample many good values for an algorithm with sub-optimal performance.

We quantify this impact by calculating, for each selected configuration and a given sample size, the *underestimation error*, that is, the relative error of the mean estimated from the selected samples relative to its true mean performance (based on the 200 verification runs). Positive values indicate that the sample mean is lower, i.e., better, than the true mean. We plot in Figure 4.22 the underestimation error for **high-quality modCMA configurations**.

We observe large underestimation errors in almost all functions. In some functions, such as F8, the underestimation error is large even for a sample size of 50. We also notice that large underestimation errors in Figure 4.22 often coincide with a large performance loss seen in Figure 4.21. This observation can be explained by looking in more detail at the performance distribution of the used configurations on a particular function, as is done in Figure 4.23 for F8.

We see in this figure that all configurations have a fraction of runs where the AOC value is very large, indicating that these were very poorly performing runs. When calculating the mean value of a configuration from a limited number of samples, if none of these poor runs appears in the samples, then the mean of the configuration will be lower than its true mean, leading to the large underestimation seen in Figure 4.22.

Additionally, since the difference in a configuration's performance seen during configuration and its true mean is often larger than the difference in the means of configurations as estimated from a small number of samples, a relatively poor-performing configuration can end up being chosen simply because it got 'lucky', which can explain the performance losses we observed previously.

Another common way in which the mean is used in benchmarking is in the basic pairwise comparison scenario, where two algorithms are directly compared to each other. To investigate this scenario, we simulate pairwise comparisons based on a limited sample size, and correlate the decisions made by the pairwise comparison to the

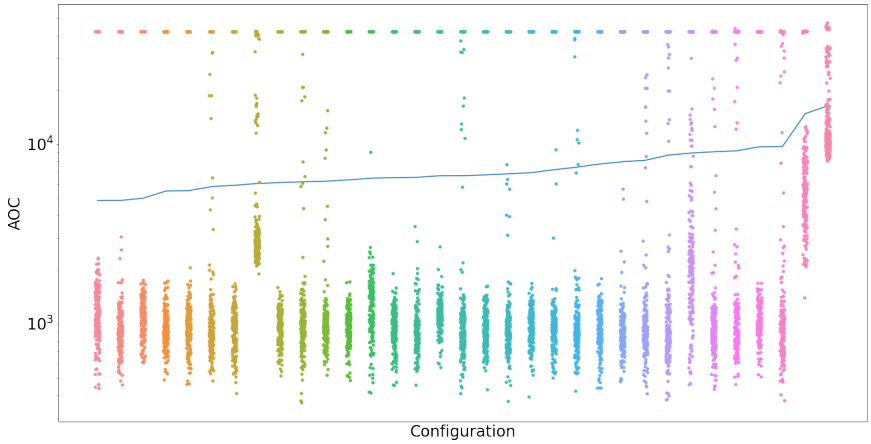
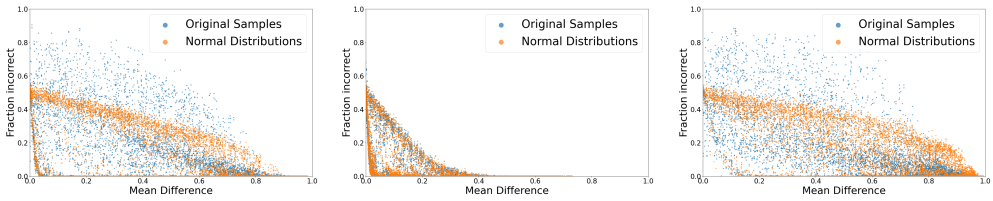
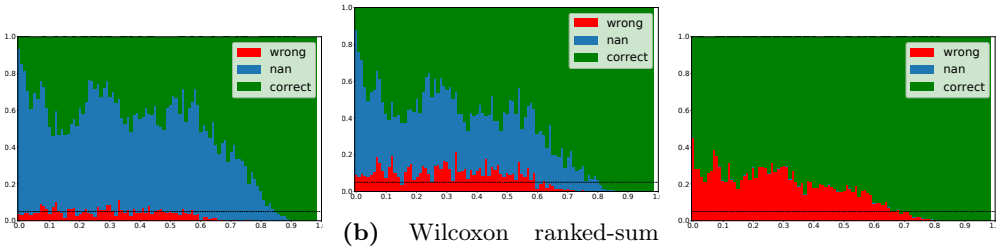


Figure 4.23: Distribution of AOC values of 200 individual runs of **high-quality mod-CMA configurations** on F8. The line indicates the mean AOC value of each configuration, and is the basis for the sorting on the x-axis.



(a) F9 **(b)** F15 **(c)** F22

Figure 4.24: Fraction of incorrect decisions when using the sample mean to compare pairs of modCMA configurations. Each subplot contains 10 000 points. Each point compares two configurations selected uniformly at random from the available configurations. The x-axis indicates the normalized difference between their true means (based on the 200 AOC values per configuration). The y-axis indicates the fraction of incorrect decisions based on 500 independent samplings of 15 AUC values for each of the two selected configurations. *Original samples* refers to sampling with replacement from the 200 AOC values available, while *Normal distributions* refers to sampling values from a normal distribution with the same mean and standard deviation as the 200 values of the corresponding configuration.

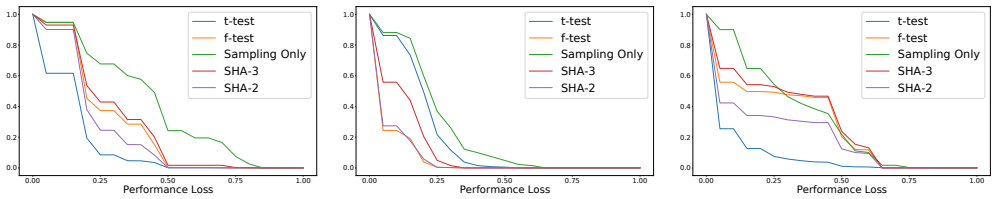


(a) t-test **(b)** Wilcoxon ranked-sum test **(c)** Comparison means

Figure 4.25: Correctness of decisions made in pairwise comparisons between modular CMA-ES configurations on F9, using different procedures. The x-axis shows the relative difference in true mean between the selected configurations. The y-axis shows the fraction of comparisons, out of 500 repetitions, that the decision was correct, incorrect or inconclusive (nan) when comparing configurations with this difference. Each repetition samples 15 values out of the 200 available for each configuration compared.

difference in true means between the selected configurations. To achieve this simulation, we use the full set of modCMA configurations generated during an irace run, which totals over 200 configurations on each function. From this set of configurations, we take 10 000 pairs, drawn uniformly at random, to perform the pairwise comparison. Then, for each pair of configurations, we sample a number of AOC values from the 200 values available, calculate the sample means and compare them to decide which configuration is the best. The comparison is *correct* if it gives the same conclusion as comparing the true means. We repeat the sampling and comparison step 500 times to calculate the fraction of times that the comparison is correct. The results of this experiment on F9, F15, and F22, with sample size 15, are displayed in Figure 4.24.

We observe in this figure that, as expected, the fraction of incorrect decisions decreases when the difference in true means increases. However, the decrease is much faster for F15 than for F9 or F22. There are also notable differences when comparing the fraction of incorrect decisions generated by sampling with replacement from the 200 AOC values available (Original samples) versus sampling values from the normal distribution that has the same mean and standard deviation as those 200 values. These distributions are almost identical for F15 but different for F9 and F22, which suggests that the fraction of incorrect decisions made by comparing means for F9 and F22 is impacted by the non-normality of the sample distribution.



(a) F9 **(b)** F15 **(c)** F22

Figure 4.26: Cumulative performance loss of 5 variants of the racing procedure using *FirstTest* = 2: t-test, Friedman-test, sampling and selecting based on mean, and successive halving with reduction factors 2 and 3.

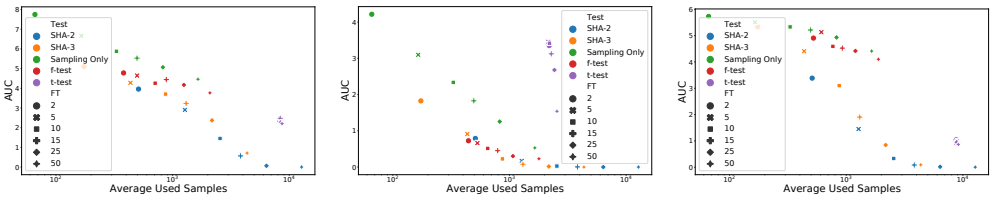
Statistical testing

When considering pairwise comparisons between algorithms, we often use statistical tests to determine if one algorithm outperforms the other. Two of the most common tests are the t-test and the non-parametric Wilcoxon rank-sum test.

To more closely analyze these two testing procedures, we re-sample with replacement, for sample size 15, from the set of 200 AOC values of the 33 **high-quality modCMA configurations**. Then, we apply a one-sided t-test to the samples of size 15 and measure the fraction of pairs in which the test was "correct", "incorrect" or "inconclusive". We consider here that the test is "incorrect" when, for a pair of algorithms A and B, the null hypothesis that A has a lower mean than B is rejected but the mean of A is indeed lower than the mean of B based on the 200 values. When neither of the two one-sided null hypotheses (A has a lower mean than B nor B has a lower mean than A) are rejected, the test is considered "inconclusive".

We zoom in on function F9 in Figure 4.25, and look at the difference between making decisions based on means, t-test and Wilcoxon rank-sum test. We note that both statistical tests show an error rate that is larger than α for pairs of configurations with a difference in means up to 60%. We also note that even though the t-test is less frequently incorrect, it is also more frequently inconclusive compared to the Wilcoxon rank-sum test, even for configurations whose means differ significantly.

Inconclusiveness is not a factor when comparing based on means, but that comes with the cost of making more incorrect decisions as well. While the number of incorrect decisions decreases when adding more samples, the overall observations for the three comparison procedures remain similar.



(a) F9 **(b)** F15 **(c)** F22

Figure 4.27: Comparison of AUC value of Cumulated performance loss (Figure 4.26), relative to the average amount of samples used by each process.

Racing

To investigate the impact of performance variability on algorithm configuration, we focus on the racing procedures used by irace, which we simulate using the **high-quality modCMA configurations** from Section 4.1.2. In particular, we consider two variants of the racing procedure [160] using either the t-test or the Friedman-test. In addition to these racing variants, we also consider two variants of Successive HAlving (SHA) [120] with reduction factors 2 and 3, respectively. For the races using statistical tests, we loosen the total budget restriction, which is usually used as stopping criteria [152], (e.g., in irace) to 10 000 total samples, which means we continue the race until 5 or fewer configurations remain, or until we exceed 10 000 sampled runs ('target runs' in irace terminology). We simulate this race 1000 times for each function and several values of *FirstTest*, and show the resulting performance loss for F9, F15, and F22 in Figure 4.26. In this figure, the performance loss is defined as the difference in the true mean of the best elite (configuration with the best sampled mean during the race) against the best configuration which was present in the race.

The cumulative performance loss is compared for both the Friedman-test and t-test variants of the racing procedure, as well as a naive *sampling-only* approach that selects based on means after *FirstTest* samples have been collected for each configuration.

When comparing the different approaches, we note that there is not a clear winner across all functions and values of *FirstTest*. Interestingly, for some of the functions where Figure 4.20 shows the largest performance losses of irace elites, the races using the Friedman test seem to perform relatively poorly. This might indicate that for these functions, we could regain some of the lost performance, if it can be detected during the algorithm configuration that a different testing strategy would be required.

From Figure 4.26, we can clearly see that any variant of racing or SHA is much more reliable than the *sampling-only* approach. However, racing uses more total samples,

since it adds runs when needed, while the sampling-only approach uses a fixed number of samples. The SHA method uses a fixed number of samples as well, but this number is significantly larger than the sampling-only approach and depends on the reduction factor used.

In order to account for the differences in total budget, we summarize cumulative performance loss curves, such as those in Figure 4.26, using their corresponding AUC values, and plot these AUC values against the total samples used in Figure 4.27.

Here, we see an explanation for the great performance of the t-test: it uses significantly more samples for the same *FirsTest* value than any of the other methods. This can happen when the test can not make any conclusive decision between the configurations and thus fails to reject enough configurations to reach the 5 elites, using up the full budget of 10 000 evaluations in the process. This matches our findings from Figure 4.25a, where we could see that the pairwise t-test often does not give any decision, even when the difference in true means between configurations is relatively large.

Chapter 5

Dynamic Algorithm Configuration and Selection

Exploiting complementarity between algorithms or configurations of algorithms can lead to significant gains in performance, as illustrated in the previous chapter. It is also known that, when solving an optimization problem, different stages of the process require different search behavior. For example, while exploration is needed in the initial phases, the algorithm needs to eventually converge to a solution (exploitation). State-of-the-art optimization algorithms therefore often incorporate mechanisms to adjust their search behavior *while optimizing*, by taking into account the information obtained during the run. These techniques are studied under many different umbrellas, such as *parameter control* [70], *meta-heuristics* [22], adaptive operator selection [162], or *hyper-heuristics* [31]. The probably best-known and most widely used techniques for achieving a dynamic search behavior are the *one-fifth success rule* [201, 56, 211] and the *covariance adaptation technique* that the family of CMA-ES algorithms [97, 98] is built upon. While each of these control mechanisms tackles the problem of balancing performance in different phases of the search in its own way, they are mostly working with a specific algorithm, aiming to tune its performance by changing internal parameters or algorithm modules. This inherently limits the potential of these methods, since different algorithms can have widely varying performances during different phases of the optimization process. By switching between these algorithms during the search, these differences could potentially be exploited to get even better performance. We refer to the problem of choosing which algorithms to switch between, and under which

circumstances, as the *Dynamic Algorithm Selection* (dynAS) problem.

Solving the dynAS problem would be an important milestone towards tackling the more general *dynamic Algorithm Configuration (dynAC)* problem, which also addresses the problem of selecting (and possibly adjusting) suitable algorithm configurations. Specifically, dynAS is limited to switching between algorithms from a discrete portfolio of pre-configured heuristics, whereas for dynAC, the algorithms come with (possibly several) parameters whose settings can have a significant influence on the performance.

While dynamically changing between algorithms or algorithm configurations can be tackled in a variety of ways. For example, in the context of machine learning, there are a variety of works which utilize principles from meta-learning to allow their algorithms to handle data streams which change over time [206] or where choices have to be made at multiple time points [231]. These problems can similarly be tackled using portfolios of algorithms, with bandit algorithms running during the optimization determining how to allocate resources between them [81].

In our work, we focus on a reinforcement-learning-based formulation of the DynAS problem [2]. In particular, we follow the principles outlined in [15] to represent the switching between algorithms as a policy function. This results in the following problem definition:

Definition 5.1 (Dynamic Algorithm Selection (dynAS))**.** Given an algorithm portfolio $\mathcal{A}$, a function $f \in \mathcal{F}$ and a state description $s_t \in \mathbb{S}$ at time step t of an algorithm run. We want to find a policy $\pi : \mathbb{S} \to \mathcal{A}$ which minimizes a performance measure $\text{PERF}(A_\pi, f)$.

Note that this definition can be extended to dynamic algorithm configuration by changing the policy to be $\pi : \mathbb{S} \to (\mathcal{A} \times \Theta_A)$, where Θ_A is the configuration space of algorithm A.

This chapter is based on the following publications: [245, 243, 110, 135, 248]

5.1 Complementarity in Anytime Performance

Before tackling the dynAS problem, we first aim to show the potential of this approach for numerical optimization. We do this by taking a data-driven approach, where we identify the complementarity between algorithms from a large portfolio purely based on their performance profiles, for a simplified version of dynAS where we can switch between algorithms only once during the optimization run. As in previous chapters,

we stick to the BBOB suite from the COCO environment [95], and in particular we make use of its rich collection of algorithm performance data [4].

Our considerations are purely based on a theoretical investigation of the potential, which might be too optimistic for the single-switch dynAS case – most importantly, because of the problem of *warm-starting* the algorithms: since the heuristics are adaptive themselves, their states need to be initialized appropriately at the switch. This may be a difficult problem when changing between algorithms of very different structure. We do not consider, on the other hand, the possibility to switch more than once, so that our bounds may be too pessimistic for the full dynAS setting, in which an arbitrary number of switches is allowed.

Given the above limitations, we therefore also provide a critical assessment of our approach, and highlight ideas for addressing the main challenges in dynAS.

5.1.1 Analysis of Available data

Since the set of available algorithms from the BBOB competitions is quite large, several issues in terms of data consistency arise. When processing the algorithms, we found that a small subset has issues such as incomplete files or missing data. We decided to ignore these algorithms and work only with the ones which were made available within the IOHanalyzer tool [62]. This leaves us with a set of 182 out of 226 possible algorithms to do our analysis.

There are some caveats to this data, mostly related to the lack of a consistent policy for submission to the competitions over the years. For example, the 2009 competition required the submission of 3 runs on 5 instances each, while the 2010 version changed this to 1 run on 15 instances. In theory, the instances should have very little impact on the performance of the algorithms, as they are selected in such a way as to preserve the characteristics of the functions. However, in practice there has been some debate about the impact of instances on algorithm performance, claiming that the landscapes of different instances of the same function can look significantly different to an algorithm [173, 170, 127] (see Chapter 3.2 for more discussion on this topic). In the following, we ignore this discussion and assume that performance is not significantly impacted by the instances.

Another issue with the dataset is the usage of widely inconsistent budgets for the different algorithms. These can be as low as $50D$ and as large as $10^7 D$. However, since we use a fixed-target perspective to study the performance of the algorithms, these differences are not very impactful.

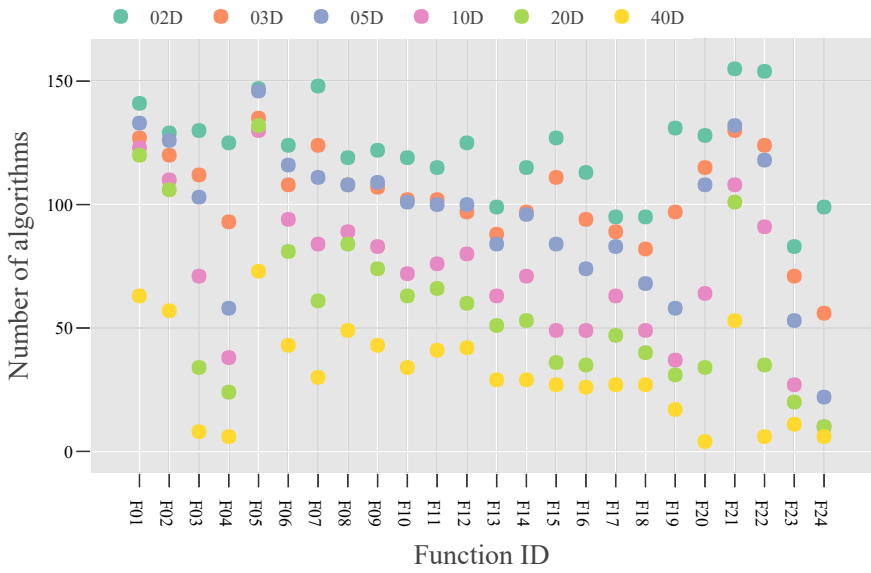


Figure 5.1: Number of algorithms with at least 15 independent runs and at least one of them reaching the target $\phi = 10^{-8}$.

Since the BBOB competitions see an optimizer as having 'solved' an optimization problem when reaching a target precision of 10^{-8}, many of the algorithms will stop their runs after reaching this point to avoid unnecessary computation. Because of this, we will use the same target value in our computations. However, for some of the more difficult functions, this target can be challenging to reach within their budget. To avoid the problem of dealing with algorithms without any finished runs, we only consider an algorithm in our analysis when it has at least 15 runs on the function, of which at least one managed to reach the target 10^{-8}. Figure 5.1 plots the number of algorithms per each function/dimensionality pair that satisfy all the requirements mentioned above. We observe large discrepancies between functions and dimensionalities, with the number of admissible algorithms ranging from 4 to 155, and note that there are no algorithms which are admissible on all functions in all dimensionalities.

5.1.2 DynAS for BBOB-Functions

In this section, we will restrict the dynAS problem on BBOB-functions to using policies which switch algorithms based on the target precisions hit. To get an indication for the amount of improvement which can be gained by dynAC over static algorithm configuration, we use the BBOB-data to theoretically simulate a simple policy which

only implements a single switch of algorithm. We can define this as follows:

Definition 5.2 (Single-Switch dynAS)**.** Let $f^{(d)}$ be a d-dimensional BBOB-function and $\mathcal{A}$ the corresponding portfolio of admissible algorithms. A single-switch policy is defined as the triple $(A_1, A_2, \tau) \in \mathcal{A} \times \mathcal{A} \times \Phi$, where $\Phi = \left\{10^{2-0.2i}) | i \in \{0, \ldots, 50\}\right\}$ is the set of admissible switchpoints. This corresponds to the policy which starts the optimization procedure with algorithm A_1, and run this until target τ is reached, after which the algorithm is changed to A_2.

The performance of this single switch method can then be calculated as follows:

$$\begin{aligned} T(f^{(d)}, A_1, A_2, \tau, \phi) = &\operatorname{ERT}(A_1, f^{(d)}, \tau) \\ &+ \operatorname{ERT}(A_2, f^{(d)}, \phi) - \operatorname{ERT}(A_2, f^{(d)}, \tau) \end{aligned}$$

Here, ϕ is the final target precision we want to reach. For the BBOB-functions, we set $\phi = 10^{-8}$, as noted in Section 5.1.1.

Generally, to assess the performance of an *algorithm selection* method, its performance can be compared to the *Single Best Solver (SBS)*, which can be defined as follows:

Definition 5.3 (Single Best Solver)**.** For each dimensionality $d \in \mathcal{D}$, we have:

$$\operatorname{SBS}_{\text{static}}(\mathcal{F}^{(d)}) = \arg\min_{A \in \mathcal{A}} \sum_{f \in \mathcal{F}} \operatorname{PERF}(A, f^{(d)}, \phi)$$

Often, ERT is used as the performance function, but this value can differ widely between functions, leading to a biased weighting. To avoid this, we can instead use the ranking of ERT per function, to give equal importance to every function. Note that we have final target precision $\phi = 10^{-8}$.

While this SBS has a good average performance, it can easily be beaten by a decent *algorithm selection* technique. As such, a better baseline for performance is needed. This is the theoretically best algorithm selection method, which is called the Virtual Best Solver. This can defined as follows:

Definition 5.4 (Static Virtual Best Solver ($\text{VBS}_{\text{static}}$))**.** For each function $f \in \mathcal{F}$ and dimensionality $d \in \mathcal{D}$, we have:

$$\operatorname{VBS}_{\text{static}}(f^{(d)}) = \arg\min_{A \in \mathcal{A}} \operatorname{PERF}(A, f^{(d)})$$

For the BBOB functions, we use $\operatorname{PERF}(A, f^{(d)}) = \operatorname{ERT}(A, f^{(d)}, \phi)$ with $\phi = 10^{-8}$.

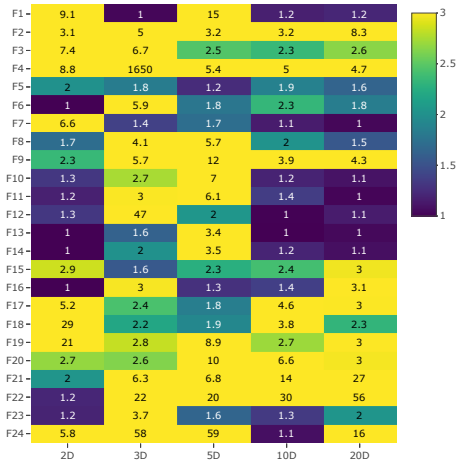


Figure 5.2: Relative ERT of the SBS over the $\text{VBS}_{\text{static}}$. The selected SBS are: Nelder-Doerr (2D), HCMA(3, 10 and 20D) and BIPOP-aCMA-STEP (5D). dimensionality 40 was removed because no algorithm hit the final target on all functions in this dimensionality.

Note that the $\text{VBS}_{\text{static}}$ will always perform at least as good as the SBS, and theoretically gives an upper bound for the performance of any real implementation of algorithm selection techniques. Thus, the difference between SBS and $\text{VBS}_{\text{static}}$ gives an indication of the maximal possible performance gained by algorithm selection. For the BBOB-data, the relative ERT between these two methods is visualized in Figure 5.2. From this, we see that the differences can be extremely large, highlighting the importance of algorithm selection.

Similar to the way we defined $\text{VBS}_{\text{static}}$, we can define a Dynamic Virtual Best Solver, VBS_{dyn}, as follows:

Definition 5.5 (Dynamic Virtual Best Solver)**.** For each BBOB-function $f \in \mathcal{F}$ and dimensionality $d \in \mathcal{D}$, we have:

$$\text{VBS}_{\text{dyn}}(f^{(d)}) = \underset{(A_1, A_2, \tau) \in (\mathcal{A} \times \mathcal{A} \times \Phi)}{\arg\min} T(f^{(d)}, A_1, A_2, \tau, \phi)$$

5.1.3 Results

Since the number of algorithms considered in this paper is relatively large, many of the results are only shown for a subset of functions, dimensionalities or algorithms.

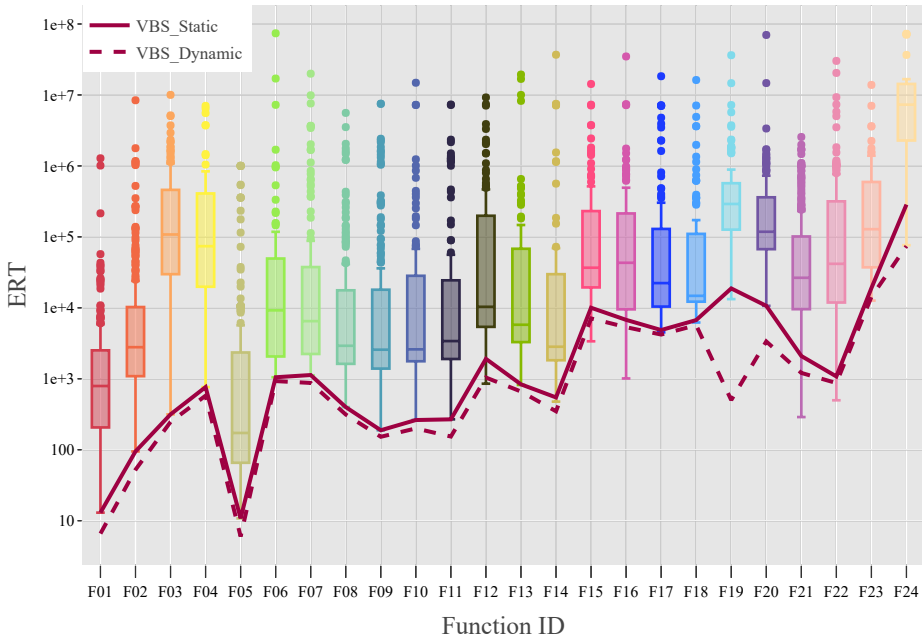


Figure 5.3: Distribution of ERTs among all algorithms for all 24 BBOB-functions in dimensionality 5. Please recall from Figure 5.1 that the number of data points varies between functions. Also shown are the ERTs of the $\text{VBS}_{\text{static}}$ and VBS_{dyn}.

Overall Gain of Single-Switch DynAS

Before investigating the possible improvements to be gained by dynamic algorithm selection, we investigate the performance of the static algorithms from the BBOB-dataset. To achieve this, we look at the distribution of ERTs among the BBOB-functions. For dimensionality 5, this is visualized in Figure 5.3.[1] This figure shows the large differences in performance, both between the algorithms as well as between the different functions. We marked the performance of the $\text{VBS}_{\text{static}}$ and VBS_{dyn}, and see that their differences also vary largely between functions.

To zoom in on the differences between the $\text{VBS}_{\text{static}}$ and VBS_{dyn} we see in Figure 5.3, we can compute for each function, dimensionality and corresponding algorithm portfolio the relative ERT of a the Single-Switch VBS_{dyn} over $\text{VBS}_{\text{static}}$. Specifically, this is calculated as $\frac{\text{ERT}(\text{VBS}_{\text{dynamic}}(f^{(d)}))}{\text{ERT}(\text{VBS}_{\text{static}}(f^{(d)}))}$. This value is shown for each (function, dimensionality)-pair in Figure 5.4. From this figure, we can see that for most func-

[1] Note that for function F05, the linear slope, most algorithms simply move outside the search-space to find an optimal solution, which is accepted by the BBOB-competitions, but leads to a disadvantage to those algorithms which respect the bounds.

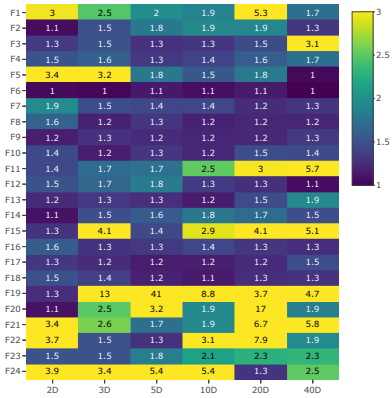


Figure 5.4: Heatmap of the ratio of ERTs between the Virtual Best Static Solver and the Virtual Best Dynamic Solver, for each (function, dimensionality)-pair.

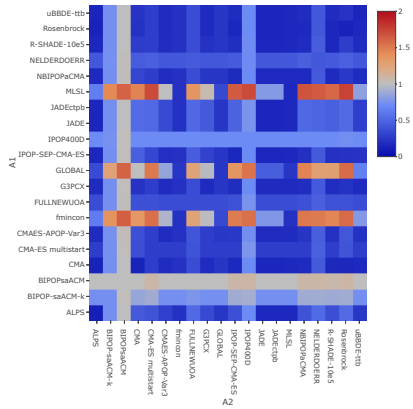


Figure 5.5: Relative ERT of configuration switches relative to $\text{VBS}_{\text{static}}$, for 10-dimensional function 21. The X- and Y-axes indicate algorithms selected as A_2 and A_1 respectively. Larger values (red) indicate better algorithm combinations.

tions, the improvements when using a single configuration change are quite large. Especially for the functions which are traditionally considered more difficult for a black-box optimization algorithm to solve, the possible improvement is massive. In terms of the median over all (function, dimensionality)-pairs, the VBS_{dyn} is 1.49 faster than the $\text{VBS}_{\text{static}}$.

Selected Algorithm Combinations

Since the VBS_{dyn} shows a lot of potential improvement over the classical $\text{VBS}_{\text{static}}$, it makes sense to study its behaviour in more detail. To achieve this, we can zoom in on a single (function, dimensionality)-pair and study the behaviour of the VBS_{dyn} and switching algorithm configurations in general. In Figure 5.5, we show the ERT of the best possible switch between any combination of algorithms in our portfolio $\mathcal{A}$, on function 21 in dimensionality 10. This figure shows some clear patterns in the horizontal and vertical lines. A horizontal line, such as the one for the MLSL-algorithm [147], indicates that an algorithm adds to the performance of most algorithms by being the A_1-algorithm.

This can be interpreted as having a good exploratory search behaviour, but poor exploitation. There are also vertical lines present, which indicate the algorithms which

perform well as A_2-algorithms. These are less pronounced than the horizontal lines, which might indicate that the choice of A_2 algorithms has less impact on the performance than the choice of A_1.

Small Portfolio: Case Study

Since the algorithm space we consider is quite large, it can be challenging to gain insights into the individual algorithms. To show that dynamic algorithm selection is also applicable to smaller portfolio's, we limit ourselves to 5 algorithms. These are representative of some widely used algorithm families: Nelder-Doerr [61], DE-Auto [252], Bipop-aCMA-Step [155], HMLSL [183], and PSO-BFGS [142]. With this reduced algorithm portfolio, we can study the improvements over their respective $\text{VBS}_{\text{static}}$ in more detail, and find interesting algorithms combinations to explore further.

To illustrate the configuration switches which can be considered in this algorithm portfolio, we can zoom in on function 12 in dimensionality 3 and look at the fixed-target curve showing ERT. This is done in Figure 5.6, where we also indicate the best switching points between algorithms. This figure highlights the different behaviors of the algorithms in the portfolio, and thus indicates where switching algorithms would be beneficial. The best possible switch in this function would occur from PSO-BFGS to Nelder-Doerr, at target $10^{-6.4}$, leading to a relative speedup of 1.76 over $\text{VBS}_{\text{static}}$.

To decide which algorithms to use in an algorithm portfolio such as the one used here, two main ways of selecting the algorithms are possible. The first is to use some knowledge about the algorithms to determine which are important. This is useful for initial exploration, but might lead to useful algorithms being ignored. Instead, one can use performance information, such as the I_1 and I_2-values, to provide some initial representation of the usefulness of algorithms to the portfolio. This approach is much more generic, however the choice of measures can be challenging. For example, the I_1 and I_2 measures are hard to extend to more general k-switch dynAS methods. Instead, an extension of marginal contributions [262] and related concepts such as measures building on Shapley values (like those suggested in [79]) would capture algorithm contribution to a portfolio in a much more robust sense, and thus be useful additions to the dynAS setting.

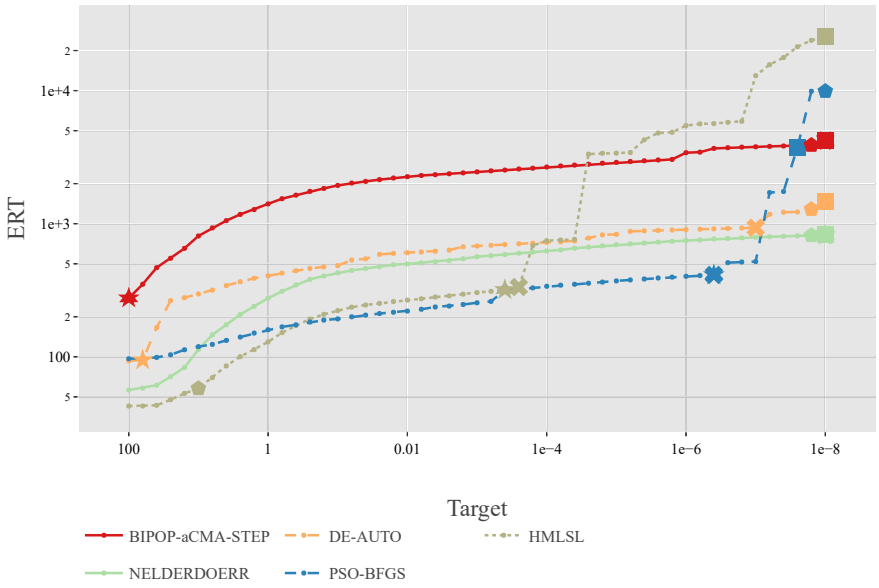


Figure 5.6: ERT-curves for a selected algorithm portfolio of size 5 on F12 in 3D. Markers indicate optimal switch points between algorithms. Their color and symbol indicate the starting and finishing algorithms respectively. (star = Nelder-Doerr, triangle = DE-AUTO, cross = BIPOP-aCMA-STEP, square = HMLSL and pentagon = PSO-BFGS).

5.2 Switching Between Algorithm Variants

To achieve dynAS, we need to tackle the problem of *warmstarting*: initializing the internal state of the secondary algorithm after the first has been terminated. Depending on the used algorithms, this can be an extremely challenging task. To limit the effort needed to warmstart an algorithm, we can ensure all algorithms share the same internal state, as is the case when we limit ourselves to a single modular algorithm framework. In this section, we work within the modCMA framework to implement the single-switch version of dynAS, where we exploit the complementarity between the many module combinations, as was illustrated in Chapter 4. In particular, we aim to switch between different configurations of modCMA (without the local restart module).

5.2.1 Selecting Adaptive Configurations

To determine which switches we should make, we start by gathering benchmark data from all 24 BBOB functions (5-dimensional versions only). We then gather the AHTs for targets $\Phi = \{10^{2-(0.2 \cdot i)} \mid i \in \{0 \dots 50\}\}$. Based on these AHT values, the adaptive configurations suggested in [232] are chosen as follows:

- For each configuration c, each of the 24 BBOB functions f, and each of the 51 target values ϕ, we calculate the AHT over all 25 runs (5 runs for each of the first five instances).
- From this data, we determine the best target value ϕ_{min} for which there exists at least one configuration whose 25 runs all reached this target.
- For every target value $\phi \in \Phi$ satisfying $\phi > \phi_{\text{min}}$ we calculate the best configuration before this target, i.e., we select the configuration c for which $AHT(c, \phi)$ is minimized. We denote this configuration C_1. We then compute the best configuration c from this target until ϕ_{min}, which we denote as C_2, i.e., C_2 is the configuration for which $AHT(f, c, \phi_{\text{min}}) - AHT(f, c, \phi_{\text{min}})$ is minimized. In [232], the theoretical performance (TH for 'theoretical hitting time') is then calculated as $TH(f, C_1, C_2, \phi) = AHT(f, C_1, \phi) - AHT(f, C_2, \phi) + AHT(f, C_2, \phi_{\text{min}})$.
- From this data we compute the target value τ for which the overall performance $TH(f, C_1, C_2, \phi)$ is minimized. This gives us the adaptive configuration (C_1, C_2, τ). We refer to τ as the 'switchpoint' of the adaptive configuration.

5.2.2 Two-Stage Configuration Selection

We introduce a procedure to make the selection process more robust to noise in the performance data. This is based on the finding that the static configurations are not quite stable enough to be used as a baseline. The first step in this process consists of selecting some static configurations for which we should gather more data. The configurations we will consider are made up of two parts. The first part consists of the 50 best-performing static configurations.[2] We then extend this set by looking at the configurations which have been selected to be a part of the 50 theoretically best adaptive configurations. Since this might not be a diverse set of configurations, as one

[2]The best static configurations are determined by their AHT at the final reached target. If fewer than 50 configurations reach this target for a function, we extend these configurations by the ones that have the lowest AHT for the previous target. We repeat this process until we have selected 50 configurations.

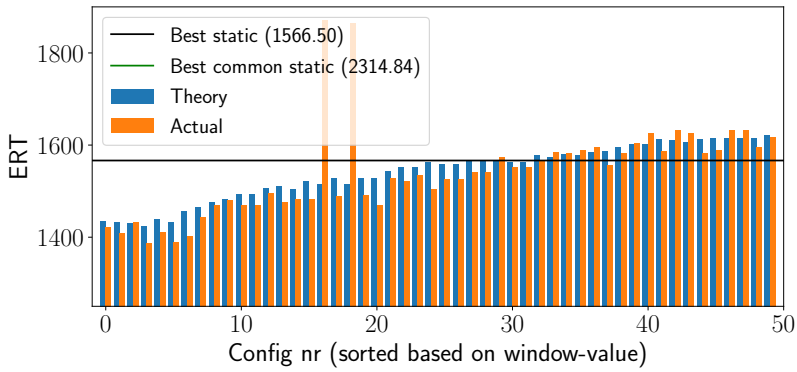


Figure 5.7: F10: ERT of adaptive configurations compared to the best static and "common" static configurations.

configuration might be chosen as C_1 50 times, we decide to limit the number of times a certain configuration can be selected as C_1 and as C_2 to three times each ('limited selection method'). This should give us a more diverse set of configurations which might contribute to good adaptive configurations. We then rerun these configurations using 50 runs on each of the 5 instances, for a total of 250 runs each.

5.2.3 Performance Comparison

The results of the two-stage method are shown in more detail in Figure 5.7 for F10. From this figure, we can see that the fit between theory and practice is quite good, and many of the adaptive configurations manage to outperform the best static configuration by around 10%. Some outliers are present, but the general trend is positive. In this figure, we also note the ERT of the best "common" CMA-ES variant as defined in [234].

An overview of the performance comparison between these groups of configurations can be seen in Figure 5.8. One important point to note is the fact that the best "common" static configuration can outperform the general best static. This is caused by the fact that these common configurations can have (B)IPOP enabled, which is not the case for the best static. In these cases, we assume that this (B)IPOP module is important to finding the optimum, and an adaptive configuration without this module will not be able to perform very well.

Next, we consider the functions for which the best static ERT is lower than that of the common variants. For these functions, we manage to improve upon this best static configuration when using an adaptive configuration. More specifically, we can

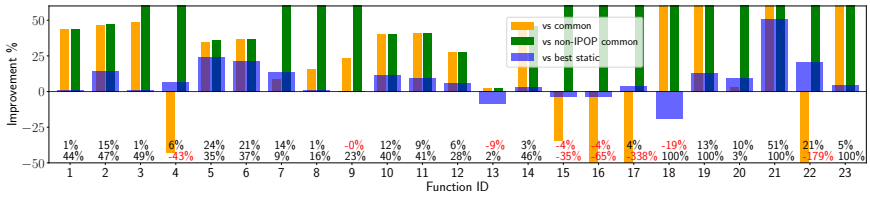


Figure 5.8: Comparison of the best achieved switching ERT relative to the ERTs of the common statics (with and without IPOP; 5×5 runs) and the best non-IPOP static on 5×50 runs. Improvements are cut off at 60% and -50%, respectively. The precise values of the improvements are shown above the x-axis for the improvement relative to the best static (top) and relative to the best common (bottom) configuration.

see that when the best static configuration from the entire configuration space does not have (B)IPOP enabled, we can reliably achieve an improvement when using adaptive configurations.

We also note that when the best static configuration with (B)IPOP significantly outperforms the best rerun configuration, we do not manage to get the same improvements. If we consider the best static configurations to include those with (B)IPOP and compare the performance of the adaptive configurations to those, no improvement is made at all.

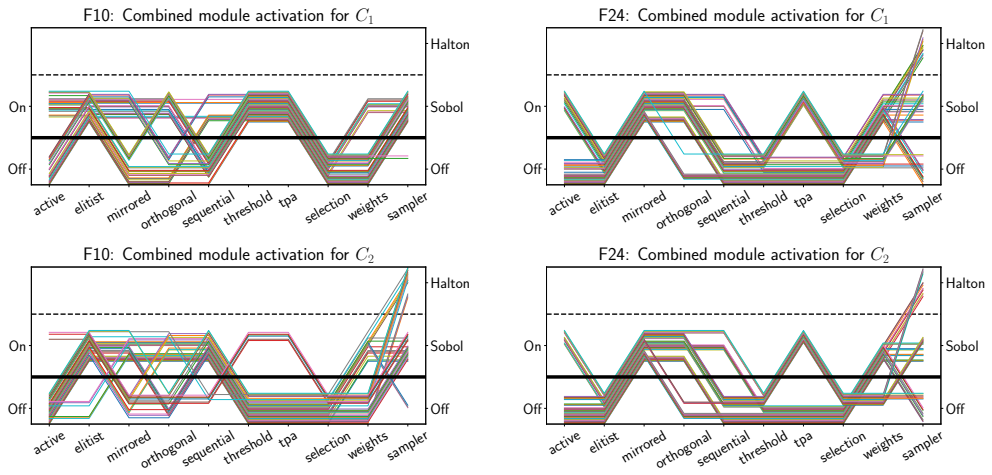
In total, we find performance gains on 18 out of 24 functions of the BBOB benchmark, with stable advantages of up to 23%.

5.2.4 Module Activation Plots

We will now study two functions in more detail. The functions we will analyze are F10, for which we see a decent improvement for most adaptive configurations, and F24, for which we see very negative results.

First, we look at which static configurations have been selected, and how they are used within the adaptive configurations. To do this, we introduce what we call *combined module activation plots*. These plots consist of two parts, corresponding to C_1 and C_2 respectively. In each of these subplots, every line indicates a configuration. The lowest line corresponds with the theoretically best adaptive configuration, increasing from there.

In Figure 5.9a and 5.9b we see these combined module activation plots for the selected adaptive configurations for F10 and F24 respectively. These figures clearly show that for F10 there is a pattern present among the adaptive configurations: the



(a) F10 **(b)** F24

Figure 5.9: Combined module activation plots for the 50 best adaptive configurations for F10 and F24.

modules TPA and threshold start activated and in almost all cases get turned off after the switchpoint. Such patterns are not present in the adaptive configurations for F24. This seems to indicate that for F24 the switches are mostly chosen because of small variances between the different configurations, instead of actual inherent properties of the configurations to perform well at certain points of the search.

5.2.5 Summary of Results

From our experiments, we found large differences in the potential of our approach between functions. For some functions, such as F10, our approach seems quite stable, resulting in improvements of over 10% for several adaptive configurations, as can be seen in Figure 5.7. However, this is not representative of all functions, as for several functions few (or any) adaptive configurations manage to outperform the static configurations.

5.3 Per-run Dynamic Algorithm Selection

Since we have now verified that switching between configurations of a single algorithm can achieve performance gains on some benchmark functions, we now place our focus on the problem of switching between different algorithm families. Our goal here is

to create a switching procedure where the algorithm to switch to is based on information collected during the optimization process, rather than a fixed, predetermined algorithm. We coin this **per-run algorithm selection** to refer to the fact that we make use of information gained by running an initial optimization algorithm ($A1$) during a single run to determine which algorithm should be selected for the remainder of the search. This second algorithm ($A2$) can then be warm-started, i.e., initialized appropriately using the knowledge of the first one. The pipeline of the approach is shown in Figure 5.10.

To extract relevant information about the problem instances, we rely on ELA features computed using samples and evaluations observed by the initial algorithm's search trajectory, i.e., *local* landscape features. Intuitively, we consider the problem instance as perceived from the algorithm's viewpoint. In addition, we make use of an alternative aspect that seems to capture critical information during the search procedure – the algorithm's internal state, quantified through a set of state variables at every iteration of the initial algorithm. To this end, we choose to track their evolution during the search by computing their corresponding *time-series* features.

Using the aforementioned values to characterize problem instances, we build algorithm selection models based on the prediction of the fixed-budget performance of the second solver on those instances, for different budgets of function evaluations. We train and test our algorithm selectors on the BBOB problems, and extend the testing on the YABBOB collection of the Nevergrad platform [200]. We show that our approach leads to promising results with respect to the selection accuracy and we also point out interesting observations about the particularities of the approach.

5.3.1 Data Collection

Problem Instance Portfolio. To implement and verify our proposed approach, we make use of a set of black-box, single-objective, noiseless problems. The data set is the BBOB suite from the COCO platform [95], which is a very common benchmark set within numerical optimization community. This suite consists of a total of 24 functions, and each of these functions can be changed by applying pre-defined transformations to both its domain and objective space, resulting in a set of different instances of each of these problems that share the same global characteristics [96].
Another considered benchmark set is the YABBOB suite from the Nevergrad platform [200], that contains 21 black-box functions, out of which we keep 17. By definition, YABBOB problems do not allow for generating different instances.

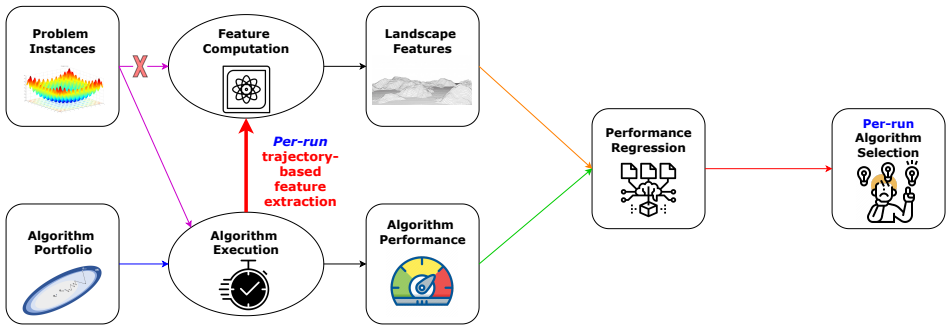


Figure 5.10: Per-run algorithm selection pipeline. The overhead cost of computing ELA features per problem instance is circumvented via collecting information about the instance during the default optimization algorithm run.

Algorithm Portfolio. As our algorithm portfolio, we consider the one used in [110, 210]. This gives us a set of 5 black-box optimization algorithms: MLSL [115, 116], BFGS [29, 77, 85, 212], PSO [123], DE [219] and CMA-ES [88]. Since for the CMA-ES we consider two versions from the modular CMA-ES framework [51] (elitist and non-elitist), this gives us a total portfolio of 6 algorithm variants.

Warm-starting. To ensure we can switch from our initial algorithm (A1) to any of the others (A2), we make use of a basic warm-starting approach specific to each algorithm. For the two versions of modular CMA-ES, we do not need to explicitly warm-start, since we can just continue the run with the same internal parameters and turn on elitist selection if required. The detailed warm-start mechanisms are discussed in [110].**Performance Data.** For our experiments, we consider a number of data collection settings, based on the combinations of dimensionality of the problem, where we use both 5- and 10-dimensional versions of the benchmark functions, and budget for A1, where we use $30 \cdot d$ budget for the initial algorithm. This is then repeated for all functions of both the BBOB and the YABBOB suite. For BBOB, we collect 100 runs on each of the first 10 instances, resulting in 1 000 runs per function. For YABBOB (only used for testing), we collect 50 runs on each function (due to no instances in Nevergrad).

In Figure 5.11, we show the performance of the six algorithms in our portfolio in the 5-dimensional case. Since the A1 budget is $30 \cdot d = 150$, the initial part of the search is the same for all switching algorithms until this point. In the figure, we can see that, for some functions, clear differences in performance between the algorithms appear very quickly, while for other functions the difference only becomes apparent after some more

evaluations are used. This difference leads us to perform our experiments with three budgets for the A2 algorithm, namely $20 \cdot d$, $70 \cdot d$, and $170 \cdot d$.
To highlight the differences between the algorithms for each of these scenarios, we can show in what fraction of runs each algorithm performs best. This is visualized in Figure 5.12. Here we can see that while some algorithms are clearly more impactful than others, the differences between them are still significant. This indicates that there would be a significant difference between a virtual best solver which selects the best algorithm for each run and a single best solver which uses only one algorithm for every run.

5.3.2 Experimental Setup

Adaptive Exploratory Landscape Analysis. As previously discussed, the *per-run* trajectory-based algorithm selection method consists of extracting ELA features from the search trajectory samples during a single run of the initial solver. A vector of numerical ELA feature values is assigned to each run on the problem instance, and can be then used to train a predictive model that maps it to different algorithms' performances on the said run. To this end, we use the ELA computation library named FLACCO [127].
Among over 300 different features (grouped in feature sets) available in FLACCO, we only consider features that do not require additional function evaluations for their computation, also referred to as *cheap features* [10]. They are computed using the fixed initial sample, while *expensive features*, in contrast, need additional sampling during the run, an overhead that makes them more inaccessible for practical use. For the purpose of this work, as suggested in preliminary studies [109, 110], we use 38 cheap features most commonly used in the literature, namely those from *y-Distribution*, *Levelset*, *Meta-Model*, *Dispersion*, *Information Content* and *Nearest-Better Clustering* feature sets.
We perform this per-run feature extraction using the initial $A1 = 30 \cdot d$ budget of samples and their evaluations per each run of each of the first 10 instances of each of the 24 BBOB problems, as well as 17 YABBOB problems (that have no instances) in dimensionalities 5 and 10.
Time-Series Features. In addition to ELA features computed during the optimization process, we consider an alternative – *time-series* features of the internal states of the CMA-ES algorithm. Since the internal variables of an algorithm are adapted during the optimization, they could potentially contain useful information about the

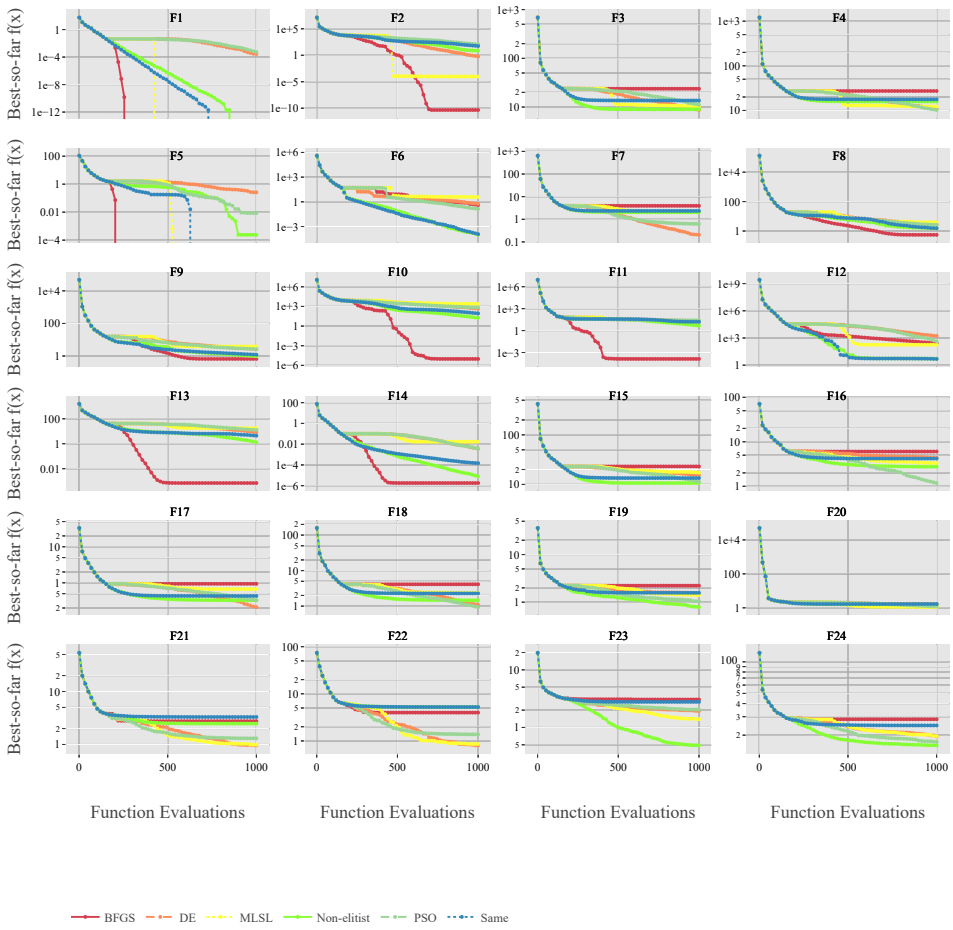


Figure 5.11: Mean best-so-far function value (precision to global optimum) for each of the six algorithms in the portfolio. For computational reasons, each line is calculated based on a subset of 10 runs on each of the 10 instances used, for a total of 100 runs. Note that the first 150 evaluations for each algorithm are identical, since this is the budget used for A1. Figure generated using IOHanalyzer [255].

current state of the optimization. Specifically, we consider the following internal variables: the step-size σ, the eigenvalues of covariance matrix $\vec{v}$, the evolution path $\vec{p}_c$ and its conjugate $\vec{p}_\sigma$, the Mahalanobis distances from each search point to the center of the sampling distribution $\vec{\gamma}$, and the log-likelihood of the sampling model $\mathcal{L}\left(\vec{m}, \sigma^2, \mathbf{C}\right)$. We consider these dynamic strategy parameters of the CMA-ES as a multivariate real-valued time series, for which at every iteration of the algorithm, we compute one data point of the time series as follows: $\forall t \in [L]$:

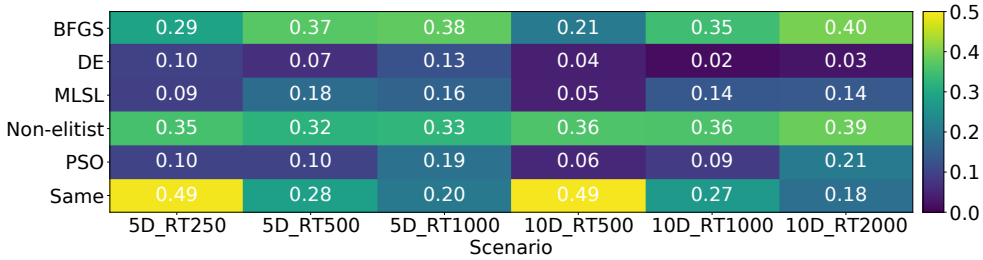


Figure 5.12: Heatmap showing for each scenario (with respect to the dimensionality and A2 budget, encoded in that order in the x-axis labels) in what proportion of runs each algorithm reaches the best function value. Note that these value per scenario can add to more than 1 because of ties.

$\vec{\psi}_t := \left(\sigma, \mathcal{L}(\vec{m}, \sigma^2, \mathbf{C}), ||\vec{v}||, ||\vec{p}_\sigma||, ||\vec{p}_c||, ||\vec{\gamma}||, \text{mean}\,\vec{v}, \text{mean}\,\vec{p}_\sigma, \text{mean}\,\vec{p}_c, \text{mean}\,\vec{\gamma}\right)^\top$,
where L represents the number of iterations these data points were sampled, which equals the A1 budget divided by the population size of the CMA-ES. In order to store information invariant to the problem dimensionality, we compute the component-wise average $\text{mean}\,\vec{x}$ and norm $||\vec{x}|| = \sqrt{\vec{x}^\top \vec{v}}$ of each vector variable.
Given a set of m feature functions $\{\phi_i\}_{i=1}^m$ from TSFRESH [42] (where $\phi_i \colon \mathbb{R}^L \to \mathbb{R}$), we apply each feature function over each variable in the collected time series. Examples of such feature functions are autocorrelation, energy and continuous wavelet transform coefficients. In this paper, we take this entire time series (of length L) as the feature window. We employ all 74 feature functions from the TSFRESH library, to compute a total of 9 444 time-series features per run. After the feature generation, we perform a feature selection method using a Random Forests classifier trained to predict the function ID, for computing the feature importance. We then select only the features whose importance is larger than 2×10^{-3}. This selection procedure yields 129 features, among which features computed on the Mahalanobis distance and the step-size σ are dominant. More details on this approach can be found in [52].
Regression Models. To predict the algorithm performance after the A2 budget, we use as performance metric the target precision reached by the algorithm in the fixed-budget context (i.e., after some fixed number of function evaluations). We create a mapping between the input feature data, which can be one of the following: (1) the trajectory-based representation with 38 ELA features per run (ELA-based AS), (2) the trajectory-based representation with 129 time-series (TS) features per run (TS-based AS), or (3) a combination of both (ELA+TS-based AS), and the target precision of different algorithm runs. We then train supervised machine learning (ML) regression

models that are able to predict target precision for different algorithms on each of the trajectories involved in the training data. Following some strong insights from [107] and subsequent studies, we aim at predicting the logarithm ($\log_{10}$) of the target precision, in order to capture the order of magnitude of the distance from the optimum. In our case, since we are dealing with an algorithm portfolio, we have trained a separate single target regression (STR) model for each algorithm involved in our portfolio. We opt for using a random forest (RF) regression, as previous studies have shown that it provides promising results for automated algorithm performance prediction [108]. To this end, we use the RF implementation from the Python package SCIKIT-LEARN [186].

Evaluation Scenarios. To find the best RF hyperparameters and to evaluate the performance of the algorithm selectors, we have investigated two evaluation scenarios:

(1) Leave-instance out validation: in this scenario, 70% of the instances from each of the 24 BBOB problems are randomly selected for training and 30% are selected for testing. Put differently, all 100 runs for the selected instance will either appear in the training or the test set. We thus end up with 16 800 trajectories used for training and 7 200 trajectories for testing.

(2) Leave-run out validation: in this scenario, 70% of the runs from each BBOB problem instance are randomly selected for training and 30% are selected for testing. Again, we end up with 16 800 trajectories used for training and 7 200 trajectories for testing.

We repeat each evaluation scenario five independent times, in order to analyze the robustness of the results. Each time, the training data set was used to find the best RF hyperparameters, while the test set was used only for evaluation of the algorithm selector.

Hyperparameter Tuning for the Regression Models. The best hyperparameters are selected for each RF model via grid search for a combination of an algorithm and a fixed A2 budget. The training set for finding the best RF hyperparameters for each combination of algorithm and budget is the same. Four different RF hyperparameters are selected for tuning: (1) *n_estimators*: the number of trees in the random forest; (2) *max_features*: the number of features used for making the best split; (3) *max_depth*: the maximum depth of the trees, and (4) *min_samples_split*: the minimum number of samples required for splitting an internal node in the tree. The search spaces of the hyperparameters for each RF model utilized in our study are presented in Table 5.1.

Per-run Algorithm Selection. In real-world dynamic AS applications, we rely on the information obtained within the current run of the initial solver on a particular

Table 5.1: RF hyperparameter names and their corresponding values considered in the grid search.

Hyperparameter	Search space
n_estimators	$[100, 300]$
max_features	$[\text{AUTO}, \text{SQRT}, \text{LOG2}]$
max_depth	$[3, 5, 15, \text{NONE}]$
min_samples_split	$[2, 5, 10]$

problem instance to make our decision to switch to a better suited algorithm. A randomized component of black-box algorithms comes into play here, as one algorithm's performance can vastly differ from one run to another on the very same problem instance.

We estimate the quality of our algorithm selectors by comparing them to standard baselines, the virtual best solver (VBS) and the single best solver (SBS). As we make a clear distinction between per-run and per-instance perspective, in order to compare we need to suitably aggregate the results. Our baseline is the *per-run VBS*, which is the selector that always chooses the real best algorithm for a particular run on a certain problem (i.e., function) instance. We then define VBS_{iid} and VBS_{fid} as virtual best solvers on instance and problem levels, i.e., selectors that always pick the real best algorithm for a certain instance (across all runs) or a certain problem (across all instances). Last, we define the SBS as the algorithm that is most often the best one across all runs.

For each of these methods, we can define their performance relative to the *per-run VBS* by considering their performance ratio, which is defined on each run as taking the function value achieved by the VBS and dividing it by the value reach by the considered selector. As such, the performance ratio for the *per-run VBS* is 1 by definition, and in $[0, 1]$ for each other algorithm selector.

To measure the performance ratio for the algorithm selectors themselves, we calculate this performance ratio on every run in the test-set of each of the 5 folds, and average these values. We point out here that the performance of different AS models are not statistically compared, since the obtained performance values from the folds are not independent [55].

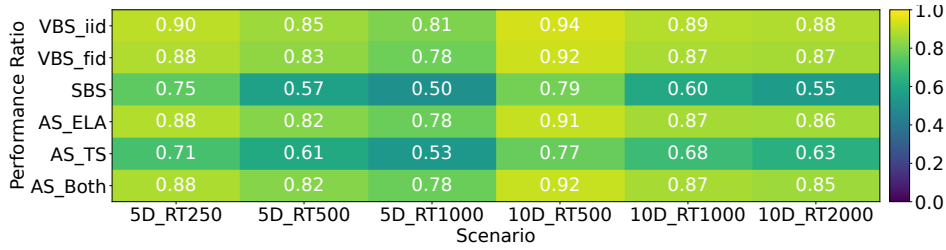


Figure 5.13: Heatmap showing for each scenario the average performance ratio relative to the per-run virtual best solver of different versions of VBS, SBS, and algorithm selectors (based on the per-instance folds). Scenario names show the problem dimensionality and the total used budget.

5.3.3 Evaluation Results: BBOB

For our first set of experiments, we train our algorithm selectors on BBOB functions using the evaluation method described in Section 5.3.2. Since we consider 2 dimensionalities of problems and 3 different A2 budgets, we have a total of 6 scenarios for each of the 3 algorithm selectors (ELA-, TS-, and ELA+TS-based). In Figure 5.13, we show the performance ratios of these selectors, as well as the performance ratios of the previously described VBS and SBS baselines. Note that for this figure, we make use of the *per-instance* folds, but results are almost identical for the *per-run* case.
Based on Figure 5.13, we can see that the ELA-based algorithm selector performs almost as well as the per-function VBS, which itself shows only minor performance differences to the per-instance VBS. We also notice that as the total evaluation budget increases, the performance of every selector deteriorates. This seems to indicate that as the total budget becomes larger, there are more cases where runs on the same instance have different optimal switches.
To study the performance of the algorithm selectors in more detail, we can consider the performance ratios for each function separately, as is visualized in Figure 5.14. From this figure, we can see that for the functions where there is a clearly optimal A2, all algorithm selectors are able to achieve near-optimal performance. However, for the cases where the optimal A2 is more variable, the discrepancy between the ELA and TS-based algorithm selectors increases.

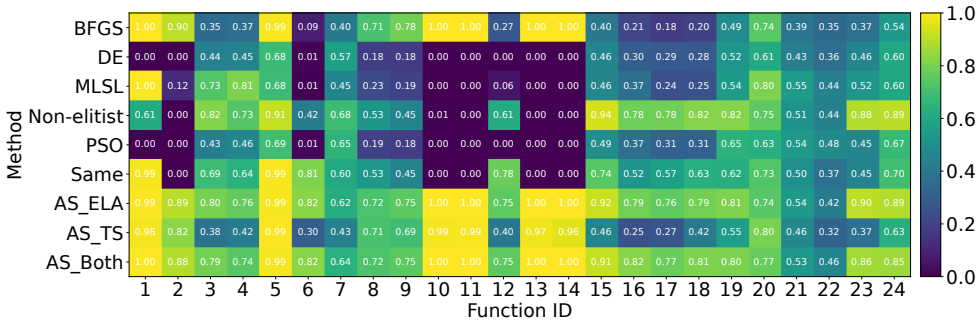


Figure 5.14: Heatmap showing for each 5-dimensional BBOB function the mean performance ratio at 500 total evaluations relative to the per-run virtual best solver, as well as the average performance ratio of each of the 3 algorithm selectors.

5.3.4 Evaluation Results: YABBOB

We now study how a model trained on BBOB problem trajectories can be used to predict the performances on trajectories not included in the training. We do so by considering the YABBOB suite from the Nevergrad platform. While there is some overlap between these two problem collections, introducing another sufficiently different validation/test suite allows us to verify the stability of our algorithm selection models. We recall that for the performance data of the same algorithm portfolio on YABBOB functions, we have target precisions for 850 runs, 50 runs per 17 problems, in all considered A2 budgets.

Training on COCO, testing on Nevergrad. This experiment has resulted in somewhat poorer performance of the algorithm selection models on an inherently different batch of problems. The comparison of the similarity between BBOB and YABBOB problems presented below nicely shows how the YABBOB problems are structurally more similar to one another than to the BBOB ones. To investigate performance flaws of our approach when testing on Nevergrad, we compare, for each YABBOB problem, how often a particular algorithm is selected by the algorithm selection model trained on the BBOB data with how often that algorithm was actually the best one. This comparison is exhibited in Figure 5.15. We observe that MLSL in particular is not selected often enough in the case of a large A2 budget, as well as a somewhat strong preference of the selector towards BFGS. An explanation for these results may be the (dis)similarities between the benchmarks. An analysis of the Pearson correlation between the trajectories on the BBOB and YABBOB suites showed limited correlation between these two suites, which might explain the poor

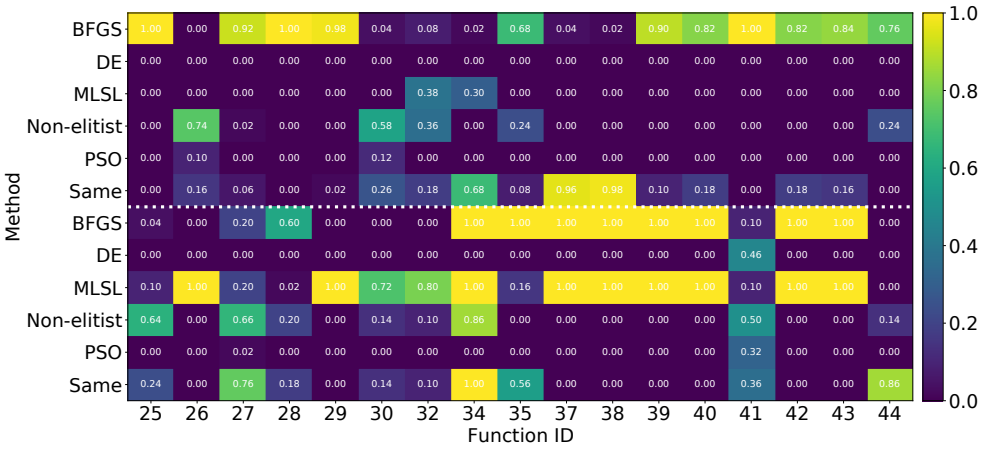


Figure 5.15: Heatmap showing for each 5-dimensional YABBOB/Nevergrad function the fraction of times each algorithm was optimal to switch to when considering a total budget of 500 evaluations (bottom) and how often each of these algorithm was selected by the algorithm selector trained on BBOB/COCO (top). Note that the columns of the bottom part can sum to more than 1 in case of ties.

generalization results [135].

5.4 When to Switch?

In the previous section, we have illustrated a way in which we can perform a single switch between optimization algorithm by using information collected during the search. While this information was only used to determine which algorithm should be switched to, we can extend this usage by not just deciding what to switch to, but whether to switch at all. In this final section of the DynAS chapter, we look at whether the search trajectories contain sufficient information to predict how beneficial a switch would be in the near future. Such a predictive model would be a first step towards a truly dynamic switching algorithm, as the model can be applied consistently during the search to detect whether switching is useful, without being restricted to a single pre-determined switching point.

5.4.1 Algorithm Portfolio

Since the potential of switching between algorithms seems to be highly dependent on the set of algorithms considered in the used portfolio [245], we consider a set of 5

algorithms:

- Covariance Matrix Adaptation Evolution Strategy *CMA-ES* [97] (implementation from the modCMA package [51])
- Differential Evolution *DE* [219] (implementation from nevergrad [200])
- Particle Swarm Optimization *PSO* [123] (implementation from nevergrad)
- Success-History based Adaptive Differential Evolution *SHADE* [223] (implementation from pyade [198])
- Constrained Optimization By Linear Approximation *Rcobyla* [190] (implementation from nevergrad)

We show the performance of these 5 algorithms on the 10-dimensional BBOB problems from the fixed-budget perspective in Figure 5.16. We see that there are significant differences in the performances of these algorithms, with no algorithm consistently dominating all others.

In addition to the algorithms, we implement warm-starting mechanisms to be able to switch between them. For the Nevergrad-based algorithms, we make use of the built-in *ask-not-told* functionality, which adapts the state of the algorithm based on a set of observations $(\{x, f(x)\})$. For starting the CMA-ES we use the warmstarting mechanism proposed in [210], which sets the center of mass and stepsize based on the 3 best solutions found so far. For switching to SHADE, we initialize the population as the last N points seen by the previous algorithm, where N is the population size.

To illustrate the usability of these warmstarting mechanisms, we investigate the performance achieved by switching from each algorithm to itself, using the described warm-starting mechanism. Since each of these warmstarting mechanisms inherently loses some information about the search process, we assume the warm-started versions will have slightly worse performance than their equivalent non-warmstarted runs. The results of running each of the 5 algorithms with 5 different points at which they are warm-started, are visualized in Figure 5.17. From this figure, we see that the performance loss from warm-starting is relatively minor, indicating that most of the relevant information is passed to the second part of the search.

5.4.2 Finding use cases using irace

To identify whether the selected portfolio can benefit from dynamically switching between algorithms, we view the problem of dynamic algorithm selection from the

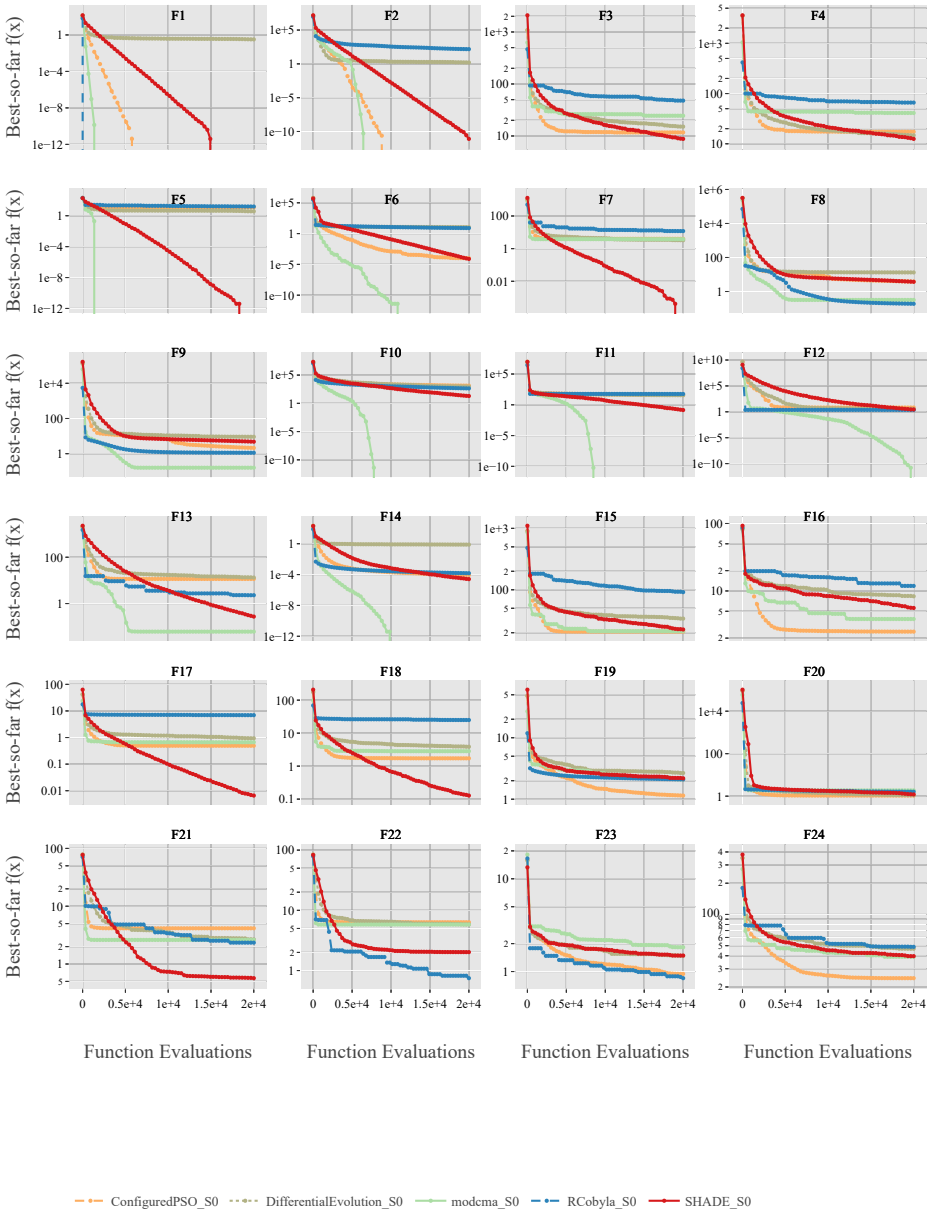


Figure 5.16: Mean function value reached, relative to the used budget, for 24 BBOB functions. Figure generated using IOHanalyzer [255]. Data available for interactive visualization at iohanalyzer.liacs.nl (IOHanalyzer dataset source 'DynAS_EvoStar23').

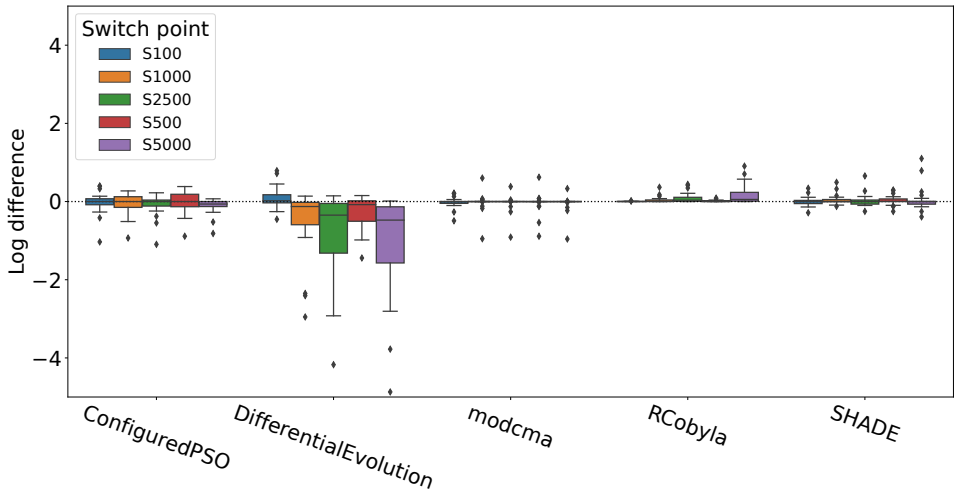


Figure 5.17: The log-distance between geometric mean of function value reached after 10 000 evaluations (limited to 10^{-8}). Differences are computed as mean with restart minus mean without restart, so negative values indicate restarts improve performance. Each box represents 24 10-dimensional BBOB problems, for each of the 5 algorithms in the portfolio for the set of 5 tested switching points.

perspective of hyperparameter tuning. We consider the dynamic algorithm to consist of three distinct parts: the first algorithm, the point at which to switch, and the second algorithm. We use irace [152] to find the configurations which reach the best function value after 5 000 function evaluations. Since irace is inherently stochastic, we perform 5 independent runs, and for each of the sets of elite configurations we perform 250 verification runs (50 runs on 5 instances). The performance of these configurations is then compared to the best static algorithm in the portfolio for each function (virtual best solver). This relative measure is visualized in Figure 5.18.

From this figure, we can see that on most problems, there are sets of configurations which seem to outperform the static algorithms. However, for some cases we see deterioration in performance compared to the VBS, indicated by negative values. This can be explained partly by the stochasticity of the algorithms: the performance observed by irace is based on a limited number of runs, and by selecting based on these limited samples can be sub-optimal when looking at the true performance distribution [247]. Additionally, there might be some cost associated with the warmstarting when the samples are collected from an initial algorithm which is not the same as the algorithm being switched to.

Since we see that there are some cases where a switch between algorithms appears

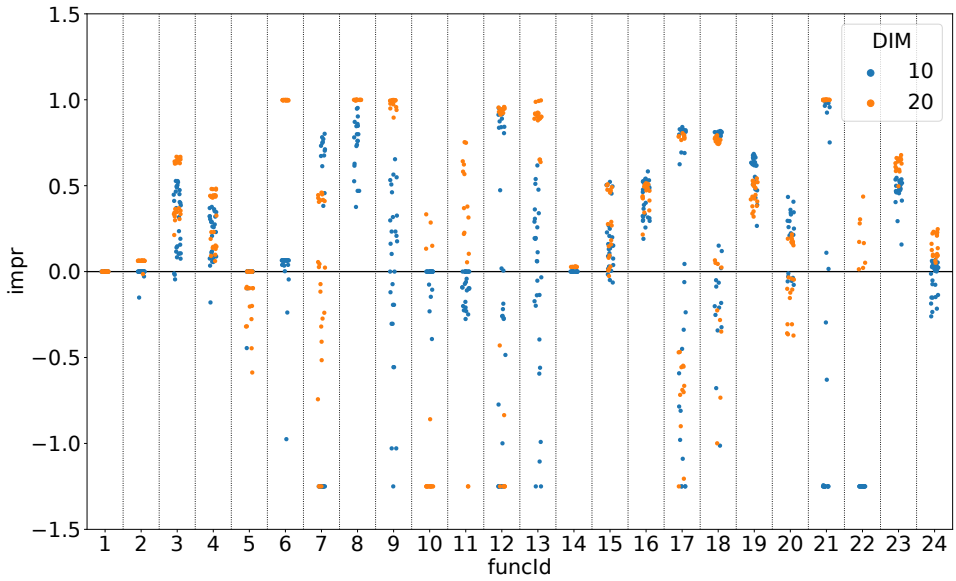


Figure 5.18: Relative improvement in terms of geometric mean of final function value of the elite configurations of irace against the virtual best solver (best static algorithm per function/dimensionality). Negative improvements are capped at -1.25 for visibility.

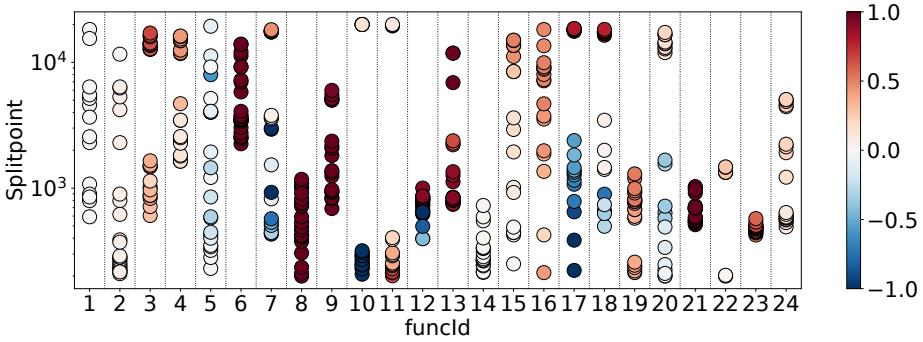


Figure 5.19: Distribution of the switch point in the elite configurations found by irace, for the 20-dimensional versions of the BBOB functions. The color of the dots corresponds to the relative improvement over VBS as shown in Figure 5.18

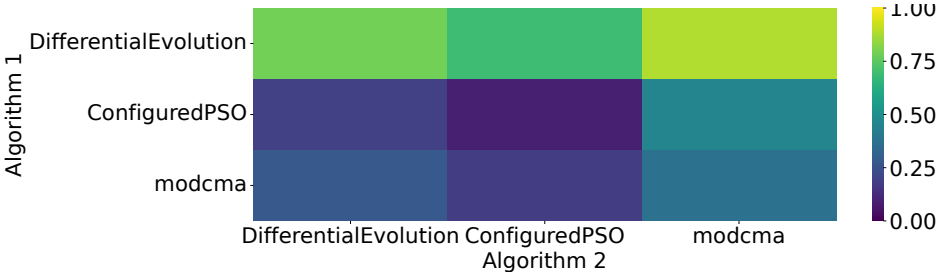


Figure 5.20: Fraction of cases in which a switch from algorithm 1 (y-axis) to algorithm 2 (x-axis) is beneficial.

beneficial, we can delve deeper into the configurations which show these benefits. In particular, we can look at the distribution of the used switch point and its correlation to the relative performance improvement, as is shown in Figure 5.19. Here, we observe that the switch points are fairly widely distributed, and that multiple different switching points can lead to similar improvements in performance.

5.4.3 Predicting Benefits of Switching

While the setup as described in Section 5.4.1 allows us to investigate the dependence of performance of a dynamic algorithm selection on the time at which the switch occurs, it does not provide directly usable insights into how this switch might be detected during the search. In order to investigate this online detection, we require a set of data where multiple switching points are attempted, such that we are able to identify on a per-run basis how beneficial each decision is. In addition, we collect features at each decision point, which can then be used to create a model to predict the observed benefits.

5.4.4 Setup

To achieve these insights into the impact of the switching point, we set up a large-scale experiment collecting the performance data for a reduced portfolio of 3 algorithms (CMA-ES, PSO and DE) on all 24 10-dimensional BBOB problems. This reduction is done to reduce computation costs. We collect 5 runs on each of the first 5 instances, and collect the full trajectory of the static algorithm up to 10000 evaluations. Then, for all switch points linearly spaced from 50 to 9500, we collect the performance data achieved when switching to each of the 3 algorithms (so we include a switch to the selected algorithm to itself) in the portfolio after another 500 evaluations.We then consider the

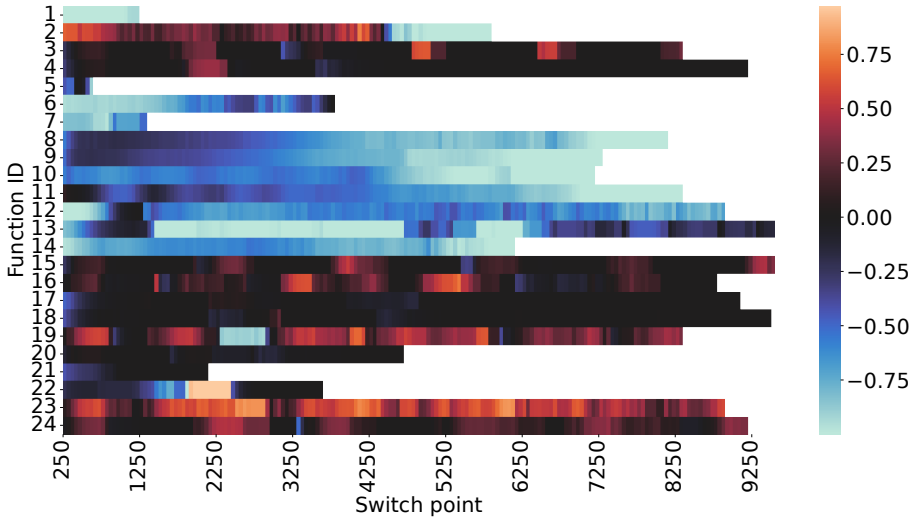


Figure 5.21: Mean relative benefit of switching from CMA-ES to DE at each of the selected switching points, for each of the 24 BBOB functions.

best fitness value reached in these 500 evaluations as the achieved performance of the dynamic algorithm. This short time-window is used to allow for the eventual creation of a dynamic switching regime which can perform more than one change during the optimization process.

In Figure 5.20 we show the fraction of cases in which a switch provides benefit over continuing the first algorithm in these 500 evaluations. From this, we see that switching is often beneficial, particularly in the case of switching to CMA. This matches our observations from Section 5.4.1, where we saw that our chosen version of DE often benefits from restarts, while the CMA-ES is the best preforming algorithm overall.

To enable an easier comparison between the algorithms, we define the target value for our model to be the relative benefit of switching after 500 evaluations, which is defined as follows:

$$r(a_s, a_r) = \left(1 - \frac{\min(a_s, a_r)}{\max(a_s, a_r)}\right)(2 \cdot \mathbb{1}_{a_s < a_r} - 1) \tag{5.1}$$

where a_s is the performance when a switch is performed, and a_r is the performance when no switch occurs. This measure takes values in $[-1, 1]$, where positive values correspond to situations where switching is beneficial, while a negative value indicates

detrimental effect of the switch.

To highlight the overall importance of the switching point, we can visualize the mean relative benefit of switching at each point in a heatmap, as is done in Figure 5.21 for the case of switching from CMA-ES to DE. Here, we see that even though the individual algorithm performance from Figure 5.16 showed that CMA-ES dominates DE in most problems, and Figure 5.20 showed that this combination is not the most promising overall, there are still many cases where a switch would still be beneficial for the performance in the next 500 evaluations. In particular, we see some clear distinctions between functions where switching is detrimental and some functions where benefits are observed, although not for all possible switching points.

In order to predict the benefit of switching at each decision point, we train a random forest model for each switch combination which outputs the relative benefit of performing the switch. The input for this model consists of the ELA features calculated on the trajectory of the first algorithm during the last $\{50, 150, 250\}$ evaluations before the switching point. We exclude the ELA features that require addition sample points, e.g., the so-called cell mapping features, resulting in 68 features in total.

This set is extended by including the diversity in the samples, both the mean component-wise standard deviation of the full set of samples ($pop_\ div$) and the standard deviation from their corresponding fitness values ($fit_\ div$).

Features which are constant for all samples or give NaN values for more than 90% of samples are removed from consideration. Features are then normalized (to zero mean and unit variance).

The random forest models use the default hyperparameters from sklearn [186]. Their performance is evaluated using the leave-one-function out strategy, where we train on the data from 23 BBOB functions and use the remaining one for testing. This is repeated for each function, and the results shown in this section are always on this unseen function. For our accuracy measure, we make use of the mean square error.

5.4.5 Results

In Figure 5.22, we show the overall model quality per decision point, aggregated over the algorithm which is being switched to. This aggregation allows us to gain an overview of the potential to learn the relative benefit of switching from data, which illustrates significant differences among test functions and the choice of the first algorithm. From this figure, we can see that some settings lead to very poor MSE values.

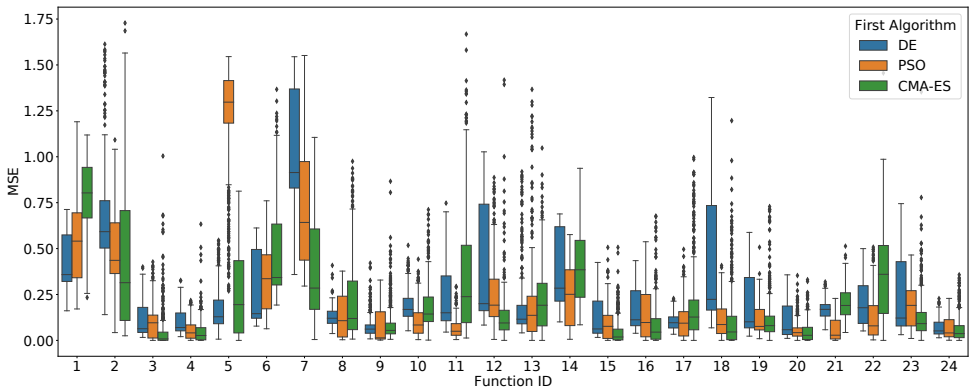


Figure 5.22: Distribution of model quality (Mean Square Error) for each function, colored according to the algorithm from which the switch occurs. Aggregated across the secondary algorithms and switch points.

This can either indicate that the model is not able to extract the needed information from the training features, or that the set of features seen on the validation-function is not consistent with the ones in the training set. For the former, it could be attributed by highly noisy feature values coming from the randomness of the first algorithm; For the latter, it is very likely that the landscape (hence the ELA features) of the test function is dissimilar to the ones in the training set. Further analyses per function/algorithm pair (Figure 5.23) aims to investigate these two possible factors. This could in part be an artifact of the leave-one-function-out validation, since the BBOB function have been originally created such that each function has distinct high-level properties [96]. However, we should note that the features we consider are trajectory-based, and are thus not necessarily as different between functions as the global version of the same features would be.

Figure 5.23 show this dependence on F7 and F15. In the top subfigure (DE to PSO on F7) we see that the actual switch (blue dots) is mostly detrimental, while the predicted value is somewhat positive, which is also reflected by quite high MSE scores of the model. Note that, the relative benefit values are not considerably noisy from the chart as the majority the sample concentrates at the very bottom, which should be learnable if the RF model were trained on this function. Hence, in this case, we conclude that, in our leave-one-function-out procedure, the model fails to generalize to function F7.

In contrast, in the bottom part of Figure 5.23 (PSO to CMA on F15), we see that

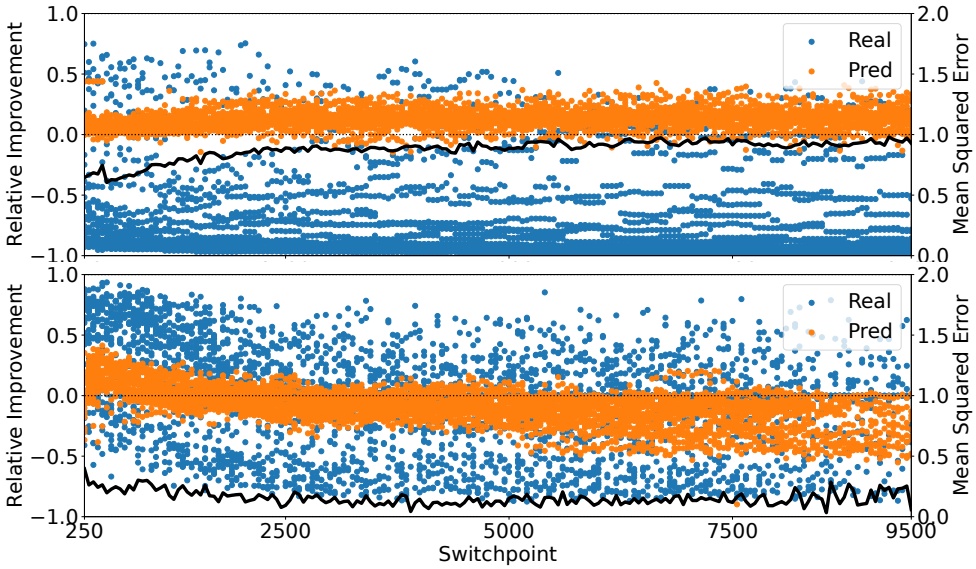


Figure 5.23: Relation between improvement and point at which the switch occurs in, for both the real improvement and the improvement predicted by the RF model. Top: switching from DE to PSO on F7. Bottom: switching from PSO to CMA on F15. The thick black line shows the MSE of the model evaluated on the selected switch point only. X-axis is shared between the two subfigures.

the overall behavior of benefit decreasing as the search continues is quite well captured by the predictions. There are two interesting aspects of the results: (1) the model seems to yield unbiased predictions of the relative benefit, which is strong support that the model generalizes well to F15; (2) The variance of the predictions are much smaller than that of the actual values, implying the possibility of a substantially large random noise when measuring the relative benefits (this observation matches with previous studies on the intrinsic large stochasticity of iterative optimization heuristics [247]). The impact of this noise might be reduced in future by performing the switch multiple times from the same switching point, leading to more stable training data.

5.4.6 Impact of Features

In addition to considering the accuracy of the trained models, we can also use the models themselves to get insights into the underlying structure of the local landscapes as seen by the algorithms. In particular, we make use of Shapley additive explanations (SHAP [157]) to gain insight into the contribution of the ELA features to the final

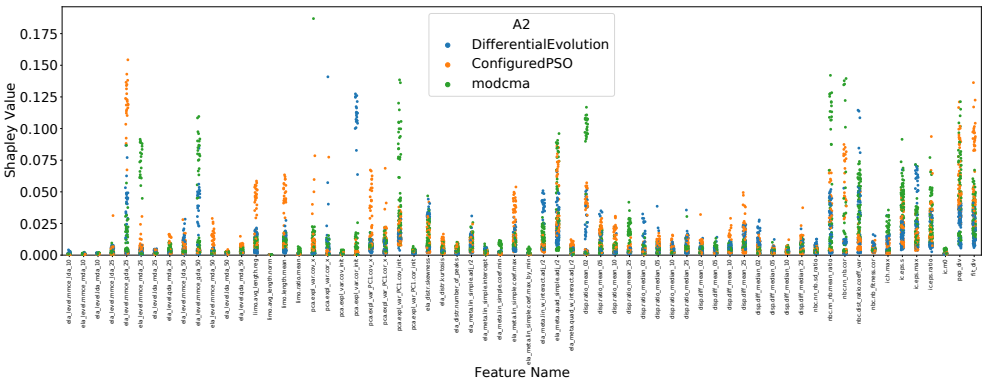


Figure 5.24: Shapley values of the features in each of the models which predict the real-valued improvement of the switch. Each dot corresponds to one model, trained on 23 functions, where the SHAP-values are calculated on the function which has been left out. Since we have 3 A1 algorithms, this leads to a total of 72 data points per feature.

predictions. Since we consider a multitude of models, we consider the distributions of Shapley values of each feature, aggregated across functions and algorithms. This is visualized in Figure 5.24.

Since Figure 5.24 is colored according to the algorithm being switched to, we can observe some interesting differences. Specifically, we see that the largest Shapley values are clearly present for different features depending on the A_2 algorithm considered. This seems to indicate that the state of the local landscape has a different effect on each algorithm. Thus, the models are indeed taking into account some specific information about the potential performance of the specific algorithm combination on which it is trained, rather than only identifying whether continuing with the current algorithm is useful in general.

By considering the local landscape features themselves without taking the models into account, we can perform dimensionality reduction to judge whether there are any patterns present in the landscape which could potentially be exploited. We make use of UMAP [163], and visualize the features obtained during the runs of CMA-ES in Figure 5.25. While this figure shows some clear clusters of similar values of the relative benefit of switching, there exist some regions where this distinction is not as clear. Based on this observation, it seems likely that the model quality can be further improved, although it is still limited by the inherent stochasticity in the dynamic algorithm selection task.

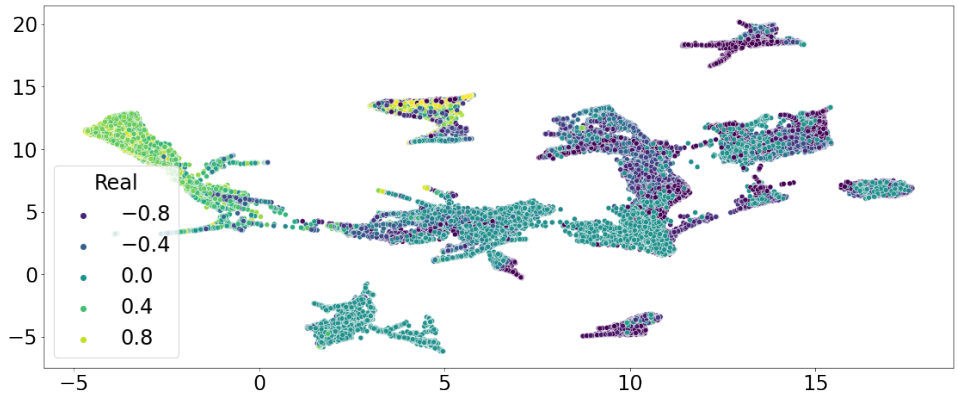


Figure 5.25: UMAP embedding of all datapoints from the CMA to DE model, where the color corresponds to the relative benefit of performing the switch.

Chapter 6

Testing Generalizability: MA-BBOB

Algorithm complementarity can be exploited in various ways, as illustrated in previous chapters. In this thesis, most of our experimentation has been using ELA features to represent the problem landscape and use this information to choose the most promising algorithm (combination) to run. The ability of ELA to differentiate between BBOB functions indeed suggests that these features are a useful representation for the algorithm selection problem which is confirmed by several studies on algorithm selection for continuous optimization heuristics which use BBOB as their benchmark suite [126, 124, 135, 18, 138, 174]. However, a key challenge with using BBOB for this type of algorithm selection lies in the evaluation of the results. One method is a leave-one-function-out technique [180], which uses 23 functions for training and the remaining one for testing. This approach tends to show poor performance since each problem has been designed to represent different high-level challenges for the optimization algorithm. As such, another technique of cross-validation by splitting function instances is commonly used [126]. However, this is likely to overfit and overestimate the performance of the selector, since the instances of different problems are inherently very similar. Thus, overfitting to biases of the instance design is an often overlooked risk [135].

One potential way in which this bias can be reduced is by creating new, larger sets of benchmark problems (e.g. using genetic programming to fill the instance space [172, 149]), or by creating a problem generator (e.g., the GNBG generator [263] or the

W-model in pseudo-Boolean optimization [257]). Such problem generators can then create arbitrarily many benchmark functions, to be used in the common train/test or cross-validation mechanisms from the machine learning community [188].

This chapter is a shortened version of the journal paper [249], which accumulates and extends work presented at the GECCO [251] and the AutoML [250] conferences. Our focus is on describing the Many-Affine BBOB function generator (MA-BBOB). To construct new functions, we create affine combinations between existing BBOB functions, building on the work of Dietrich and Mersmann [58]. Our generator is a generalization of their approach, designed to create unbiased combinations of problems where the contribution of the components can be smoothly varied. We highlight the core design choices made in the construction of MA-BBOB in Section 6.1 and illustrate their impact on the types of problems which can be created.

The parameterization of the MA-BBOB generator allows us to investigate controlled, potentially small differences in functions from both the ELA and algorithm performance perspectives. This is illustrated in Section 6.2 by investigating the addition of global structure (sphere function) to all other BBOB problems, as well as transitioning between pairs of BBOB problems.

Finally, in Section 6.3 we make use of a set of 1 000 functions generated with MA-BBOB to illustrate an algorithm selection scenario, and show that the generalization from BBOB to MA-BBOB fails to meet expectations. A comparison of the ELA-based algorithm selection approach with an artificial baseline using the weights of the affine combinations indicates that there is room to improve on the current ELA-based setup, especially when trying to generalize from BBOB to MA-BBOB.

6.1 Many-Affine BBOB

6.1.1 Pairwise Affine Combinations

To create affine combinations between two BBOB functions, we use a slightly modified version of the procedure proposed in [58]. Specifically, we define the combination C as follows:

$$
\begin{aligned}
C(F_{1,I_1}, F_{2,I_2}, \alpha)(x) = 10^X, \text{ with} \\
X = \Big(\alpha \log_{10}\left(F_{1,I_1}(x) - F_{1,I_1}(O_{1,I_1})\right) + \\
(1-\alpha)\log_{10}\left(F_{2,I_2}(x - O_{1,I_1} + O_{2,I_2}) - F_{2,I_2}(O_{2,I_2})\right)\Big)
\end{aligned}
$$

Here, F_1, I_1, F_2, and I_2 are the two base functions and their instance numbers, as defined in BBOB [96]. O_{1,I_1} and O_{2,I_2} represent the location of the optimum of functions F_{1,I_1} and F_{2,I_2} respectively. The transformation to x when evaluating F_{2,I_2} is performed to make sure the location of the optimum is at O_{1,I_1}. As opposed to the original definition, we subtract the optimal values before aggregating and take a logarithmic mean between the problems. This way, we can use consistent values for α across problems, without having to perform the entropy-based selection performed in [58]. It has the additional benefit of ensuring the objective value of the optimal solution is always 0, so the comparison of performance across instances and problems is simplified. In Figure 6.1, we illustrate the change in the landscape for the combination of F21 and F1, for different values of α.

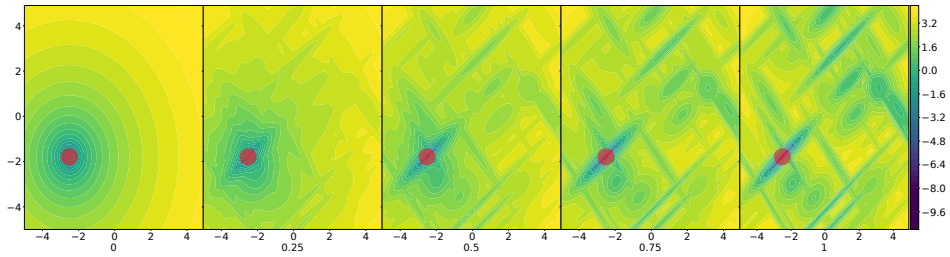


Figure 6.1: Evolution of the landscape (log-scaled function-values) of the affine combination between F21 ($\alpha = 1$) and F1 ($\alpha = 0$), instance 1 for both functions, for varying α in 0.25 increments. The red circle highlights the location of the global optimum.

6.1.2 Combining Multiple BBOB Functions

We extend the pairwise affine combinations from Section 6.1.1 to create a function generator which uses affine combinations of multiple BBOB functions. In particular, our generator is defined as follows:

$$\textit{MA-BBOB}(\vec{W}, \vec{I}, \vec{X}_{\texttt{opt}})(x) = R^{-1}\Big(\sum_{i=1}^{24} W_i \cdot R_i\big(F_{i,I_i}(x - \vec{X}_{\texttt{opt}} + O_{i,I_i}) - F_{i,I_i}(O_{i,I_i})\big)\Big)$$

Here, $\vec{W}$ and $\vec{I}$ are 24-dimensional vectors containing the weight and instance identifiers respectively, and $\vec{X}_{\texttt{opt}}$ is the location of the optimum, which we generate uniformly at random in the domain $[-5, 5]^d$. Finally, R_i and R^{-1} are rescaling functions, defined in Section 6.1.2. We highlight the motivation behind each of these design choices in the following subsections.

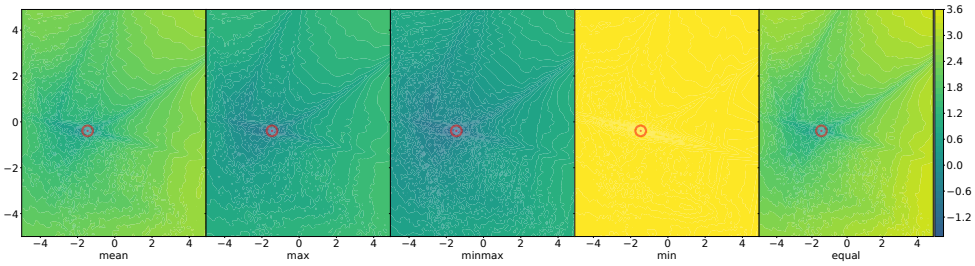


Figure 6.2: Log-scaled fitness values of an example of a single many-affine function with 5 different ways of scaling. The first 4 are taking the mean, max, $(\max + \min)/2$ and min of 50 000 random samples to create the scale factor, while the fifth ('equal') option does not make use of this scaling.

Scaling of Function Values

While the geometric weighted average used in Section 6.1.1 between component functions reduces the impact of small differences in scale, some BBOB problems vary by orders of magnitude, which can still cause one function to dominate the combined landscape. To address this, we add a rescaling function to the MA-BBOB definition, which transforms the log-precision on each component function into approximately $[0, 1]$ before the transformation. This is done by capping the log-precision at -8, adding 8 so the minimum is at 0 and dividing by a *scale factor* S_i. This procedure aims to make the target precision of 10^2 similarly easy to achieve on all component problems. We thus get the following scaling functions:

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

To determine practical scale factors, we collect a set of 50 000 random samples and evaluate them. We then aggregate the resulting function values (transformed to log-precision) in several ways: min, mean, max, $(\max + \min)/2$. In Figure 6.2, we show the differences between these methods for a selected problem in $2d$. Somewhat subjectively, we select the $(\max + \min)/2$ scaling as the technique to use for the MA-BBOB generator. To ensure we don't have to repeat this sampling procedure each time we instantiate the problem in a new dimensionality, we investigate the relation between dimensionality and the chosen scale factor calculation. This is visualized in Figure 6.3, where we see that, with an exception for the smallest dimensionalities, the

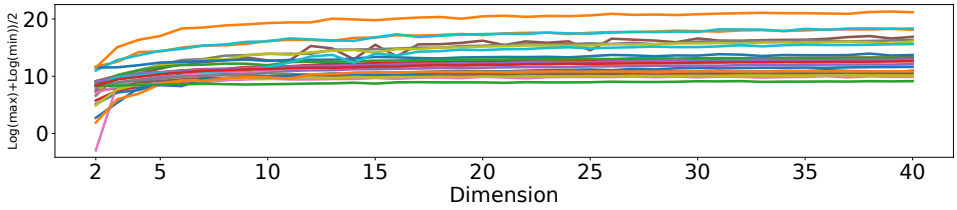


Figure 6.3: Evolution of the log-scaled $(\max + \min)/2$ scaling factor, relative to the problem dimensionality. The values are based on 50 000 samples. Each line corresponds to one of the 24 BBOB functions.

Function ID	1	2	3	4	5	6	7	8	9	10	11	12
Scale Factor	11.0	17.5	12.3	12.6	11.5	15.3	12.1	15.3	15.2	17.4	13.4	20.4
Function ID	13	14	15	16	17	18	19	20	21	22	23	24
Scale Factor	12.9	10.4	12.3	10.3	9.8	10.6	10.0	14.7	10.7	10.8	9.0	12.1

Table 6.1: Final scale factors used to generate MA-BBOB problems.

values remain quite stable. Because of this, we make use of a static scale factor rather than defining one for each dimensionality individually. The final factors used are calculated as a rounded median of the values from Figure 6.3, and shown in Table 6.1.

Instance Creation

Another design choice we made was to place the optimum of the combined function uniformly in the domain ($[-5, 5]^d$). This differs from the earlier versions used for pairwise combinations of BBOB functions [58, 251], where the optimum of one of the component functions was re-used. However, the biases in the original BBOB instance generation procedure would then be transferred into the combinations as well [150]. Since our function generator does not have to guarantee the preservation of global function properties, we take the risk of moving parts of the regions of interest outside the domain to have a less biased location of the global optimum. Figure 6.4 shows how a $2d$-function changes when moving the optimum location.

Sampling Random Functions

To allow for the usage of MA-BBOB as a function generator, we need to create a default setting to generate useful weight-vectors. This could be done uniformly at random (given a normalization step). However, in this way, the weight for every component is likely to be non-zero, so most functions contribute to the final combination, erasing

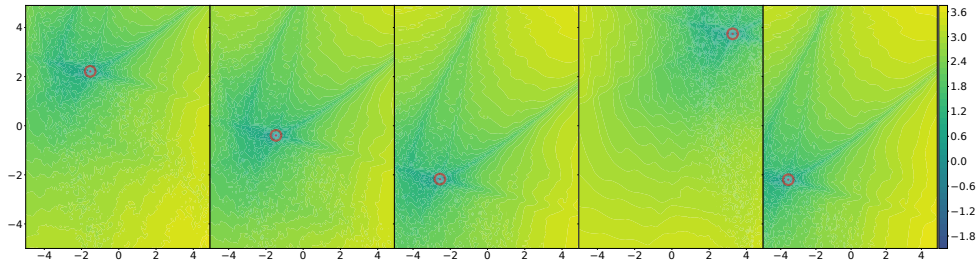


Figure 6.4: Log-scaled fitness values of an example of a single many-affine function with changed location of optimum.

the possibility of generating unimodal problems since some multimodality will always be included from some of the multimodal component functions.

To address this issue, we adapt the sampling technique to combine fewer component functions on average. Our approach is based on a threshold value to determine which functions contribute to the problem. The procedure for generating weights is thus as follows: (1) Generate initial weights uniformly at random, (2) adapt the threshold to be the minimum of the user-specified threshold and the third-highest weight, (3) this threshold is subtracted from the weights, all negative values are set to 0. The second step is to ensure that at least two problems always contribute to the new problem. We decide to set the default value at $T = 0.85$, such that on average 3.6 (i.e., 15% of 24) problems will have a non-zero weight.

6.2 Pairwise Affine Combinations

For the first analysis of the MA-BBOB functions, we limit ourselves to the combination of pairs of functions. This allows a more low-level investigation into the transition of both landscape features and algorithm performance.

6.2.1 Setup

For the algorithm performance-based analysis, we make use of a portfolio of five algorithms. Of these, three are accessed through the Nevergrad framework [200]:

- Differential Evolution (DE) [219]
- Constrained Optimization By Linear Approximation (Cobyla) [191]

- Diagonal Covariance Matrix Adaptation Evolution Strategy (dCMA-ES) [98]

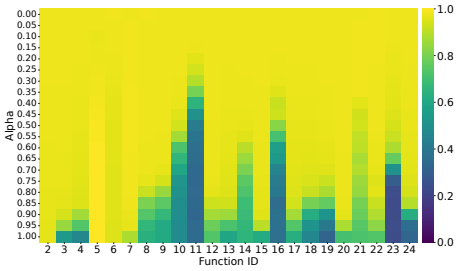
The remaining algorithms are two modular algorithm families: modular Differential Evolution [240] (modDE) and modular CMA-ES [51] (modCMA). All algorithms, including the modular ones, use default parameter settings. Each run we perform has a budget of $2\,000d$, where d is the dimensionality of the problem. We perform 50 independent runs per function. For the pairwise function combinations, we stick to the terminology introduced in Section 6.1.1 for easier comparison with the previous results in [251].

In the remainder of this section, we set $I_2 = 1$. As such, when discussing the *instance* of a pairwise affine combination $C(F_1, I_1, F_2, I_2, \alpha)$, we are referring to I_1. Note that we also introduce the uniform sampling of the optima for these experiments, following the description in Section 6.1.2. For our performance measure, we make use of the normalized area over the convergence curve (AOCC), to be maximized. The AOCC is an anytime performance measure, which is equivalent to the area under the cumulative distribution curve (AUC) given infinite targets for the construction of the ECDF. This measure is thus slightly more precise than the AUC, and can easily be computed online. To remain consistent with the performance measures used in our previous work, and analysis of results on BBOB in general, we use 10^2 and 10^{-8} as the bounds for our function values, and perform a log-scaling before calculating the AOCC. We thus calculate the normalized AOCC of a single run as follows:

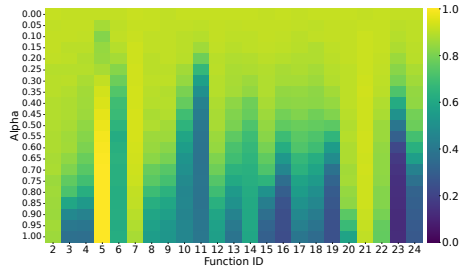
$$AOCC(\vec{y}) = \frac{1}{B}\sum_{i=1}^{B} 1 - \frac{\min(\max((\log_{10}(y_i), -8), 2) + 8}{10}$$

where $\vec{y}$ is the sequence of best-so-far function values reached, B is the budget of the run. To obtain the AOCC over multiple runs, we simply take the average.

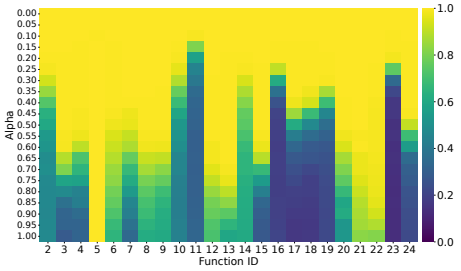
For the landscape analysis, we make use of the pFlacco [192] package to calculate the ELA features. We use a sample size of $1\,000d$ points, sampled using a Sobol' sequence. We note this large sample size is used to remove some of the inherent variability in the ELA features, even though practical applications usually rely on much smaller budgets. To be consistent with our previous work [250], we don't include features which require additional samples and remove all features which lead to NaN-values or remain static for all functions, resulting in a set of 44 features.



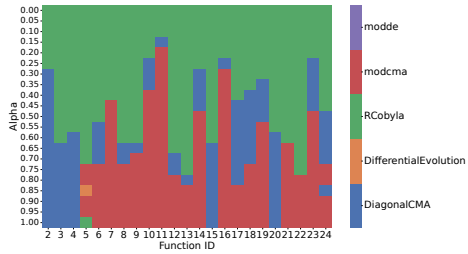
(a) AOCC values for Diagonal CMA-ES.



(b) AOCC values for Differential Evolution.



(c) AOCC values for Cobyla.



(d) Best performing algorithm from the portfolio, based on AOCC.

Figure 6.5: Normalized area under the ECDF curve of three selected algorithms (a-c) and best-ranking algorithm from the full portfolio (d) for each combination of the BBOB-function (x-axis) with a sphere model, for given values of α (y-axis) AOCC is calculated after 10 000 function evaluations, based on 50 runs on 50 instances (and location of optimum). Note that $\alpha = 0$ corresponds to the sphere function.

6.2.2 Adding Global Structure

For the first set of experiments, we make use of affine combinations where we combine each function with F1: the sphere model (as the F_2 function in the combination). As can be seen in Figure 6.1, adding a sphere model to another function creates an additional global structure that can guide the optimization toward the global optimum. As such, these kinds of combinations might allow us to investigate the influence of an added global structure on the performance of optimization algorithms. While to some extent this can already be investigated by comparing results on the function groups of the original BBOB with different levels of global structure, the affine function combinations allow for a much more fine-grained investigation.

In Figure 6.5a, we can see that the performance of CMA-ES does indeed seem to move smoothly between the sphere and the function with which it is combined. It is

however interesting to note the differences in speed at which this transition occurs. For example, while the final performance on functions 3 and 10 seems similar, the transition speed differs significantly. This seems to indicate that for F10, the addition of some global structure has a relatively weak influence on the challenges of this landscape from the perspective of the CMA-ES, while even small amounts of global structure significantly simplify the landscape of F3.

We can perform a similar analysis on other optimization algorithms. In Figures 6.5b and 6.5c, we show the same heatmap as Figure 6.5a, but for Differential Evolution and Cobyla, respectively. It is clear from these heatmaps that the performance of DE is more variable than that of CMA-ES, while Cobyla's performance drops off much more quickly. The overall trendlines for DE do seem to be somewhat similar to those seen for diagonal CMA-ES: the transition points between high and low AOCC in Figure 6.5b are comparable to those seen in Figure 6.5a. There are however still some differences in behavior, especially relative to Cobyla. These differences then lead to the question of whether there exist transition points in ranking between algorithms as well. Specifically, if one algorithm performs well for $\alpha = 0$ but gets overtaken as $\alpha \to 1$, exploring this change in ranking would give further insight into the relative strengths and weaknesses of the considered algorithms.

To study the impact on the relative ranking of algorithms, we make use of the full portfolio of 5 algorithms and rank them based on AOCC on each affine function combination. We then visualize the top ranking algorithm on each setting in Figure 6.5d. From this figure, we can see that Cobyla deals well with the sphere model, managing to outperform the other algorithms when the weighting of the sphere is relatively high. Then, after a certain threshold, the CMA-ES variants consistently outperform the rest of the portfolio, with dCMA taking over when Cobyla is no longer preferred. However, as α increases further, and the influence of the sphere model diminishes, an interesting pattern seems to occur. For several problems, there is a second transition point to modCMA, indicating that the differences in default parameterizations between the used libraries have a large impact on the algorithms' behavior. One significant factor is related to the initial stepsize, which is smaller for dCMA, and thus might lead to it becoming more easily stuck in local optima when the global structure is not as strong.

In order to better understand what the transitions in algorithm ranking look like, we can zoom in on one of the functions and plot the distribution of AOCC for all values of α. This is done in Figure 6.6, where we look at the combination between F10 and the sphere model. In this figure, we observe that Cobyla is very effective at optimizing the sphere and the combinations with low α. However, when α increases,

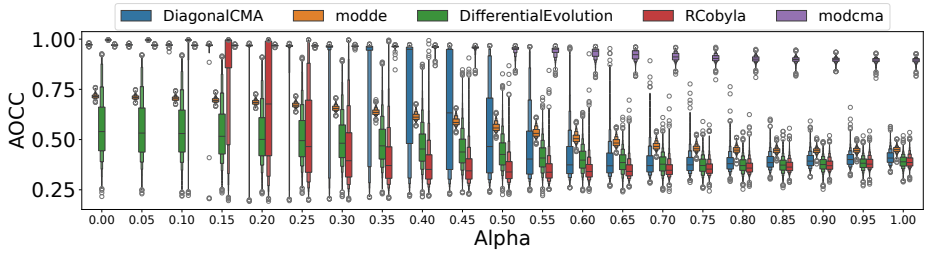


Figure 6.6: Distribution of AOCC values for 5 algorithms on the affine combinations between F10 ($\alpha = 1$) and F1 ($\alpha = 0$), for selected values of α.

Cobyla quickly starts to fail, while, for example, DiagonalCMA still manages to solve most instances at $\alpha = 0.25$ with similar AOCC. As α increases further, the modCMA's performance remains stable, showing only a minor drop in performance relative to the one seen in dCMA.

6.2.3 Impact on ELA Features

In addition to the performance perspective, we can also look at what happens to the landscape feature of the BBOB functions as we add increasingly more influence from the sphere function. Since we measure 44 different ELA features, our analysis of the impact is rather more high-level than the algorithm performance viewpoint, as we first aim to capture the overall stability of the features for increasing α values. This is measured as the sum of absolute differences in feature mean for consecutive α's, which is plotted in Figure 6.7. From this figure, we can see that the mean of most features remains quite stable, with a few notable exceptions. In particular, functions 16 and 23 show many feature changes from the sphere, which matches observations from e.g. [203, 216].

In addition to the differences between functions, it is also clear to see that features don't all behave in the same way. Since Figure 6.7 only shows absolute changes in mean, the deviation between instances might also play a role. To analyze this in some more detail, we select a single function (F10) and look at the evolution of both the feature mean and its standard deviation in Figure 6.8. From this figure, we can see that the mean of each feature seems to transition rather smoothly between the two component functions, in a similar way to the performance plots in Section 6.2.2. However, we should note that the standard deviation of many features is relatively

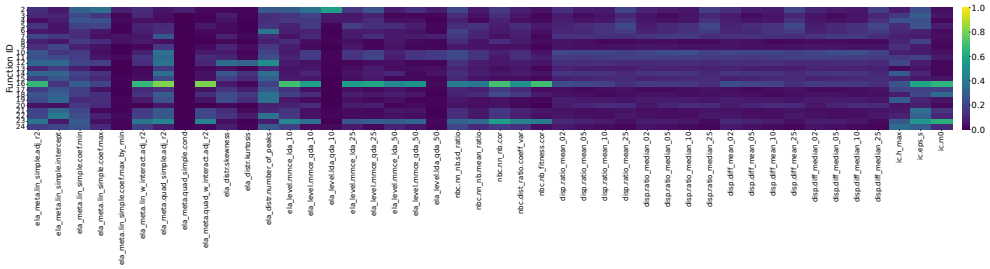


Figure 6.7: Total changes in each ELA feature when transitioning from the sphere to the function indicated in the row. Each cell represents the sum of differences in mean between pairs of consecutive values of α, so high values indicate a large total change in mean, while low values indicate features which remain stable throughout the transition.

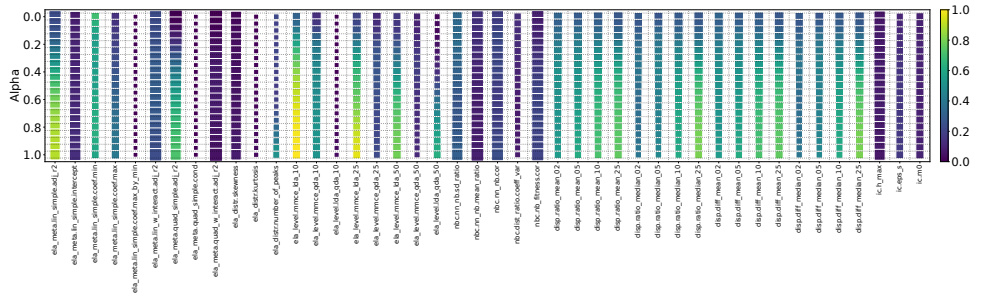


Figure 6.8: Evolution of ELA features with changing between Sphere ($\alpha = 0$) and F10 ($\alpha = 1$). The color indicates the mean of the feature over the 50 instances (lighter = larger), while the size indicates the variance (larger = higher variance).

large for each α value.

6.2.4 Impact of Optimum Location and the Instance

As can be seen from the relatively large variance in both ELA features and algorithm performance, the instance and location of the optimum can have a major impact on both the landscape and the corresponding algorithm behaviour. In particular, the way in which we defined an instance in our setup is not necessarily equivalent to the common interpretation, e.g., from BBOB. Since we allow the optimum of a component function to be moved anywhere in the domain, this can lead to large parts of the original function no longer being reachable. This is the main reason why some BBOB functions have very restricted distributions for their optima, which can be seen by

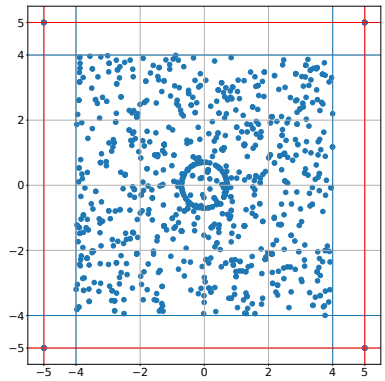


Figure 6.9: Location of optima of the 24 2d BBOB functions (1000 random instances). The red lines mark the commonly used box-constraints of $[-5, 5]^d$.

analyzing the overall distribution of optima across all BBOB functions, visualized in Figure 6.9 and previously observed in e.g.[150].

To analyse how much the algorithms in our portfolio are influenced by the choice of instance and location of optimum, we determine the relative impact of different instances for each function (F_1 and α value). This is done by first averaging the performance across all 50 runs on each instance. By dividing this by the total variance present across all runs on all instances of that function, we obtain a relative measure of 'stability' across instances, which is visualized in Figure 6.10. This figure shows that some algorithms are inherently more impacted by the instance/location of the optimum (modDE), while, for example, for Cobyla, the variance increases with increasing α, which suggests that it is very stable on the sphere problem, but becomes much more impacted by variations in the landscape when more non-sphere influence is added.

To further analyse the impact of this increased flexibility in terms of function generation, we perform an experiment involving two versions of the original BBOB functions. The first is the data from Section 6.2.2 where $\alpha \in \{0, 1\}$, which corresponds to data for 50 instances of each function, with each of them having its optimum moved to a random point in the domain. The second set of instances are created by taking the same instances of the BBOB functions, but not shifting their optimum (the rescaling from Section 6.1.2 is still applied). This allows us to compare the influence of moving the optimum on the landscape of the resulting problem. In Figure 6.11, we compare the distribution of all features on these two versions of BBOB function 23.

Finally, we can take an aggregated view of the features, and project the 44 dimen-

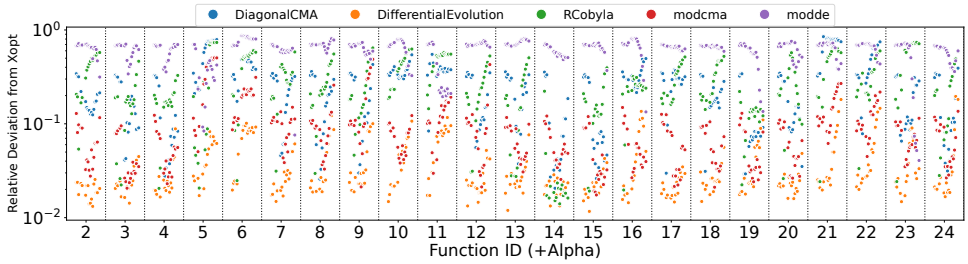


Figure 6.10: Relative deviation in AOCC caused by the change of the global optimum for all combinations of BBOB functions with the sphere model (calculated as deviation per instance divided by deviation across all instances). The x-axis indicates changing function ID, and within each function ID the transition goes from $\alpha = 0$ to $\alpha = 1$ (left to right).

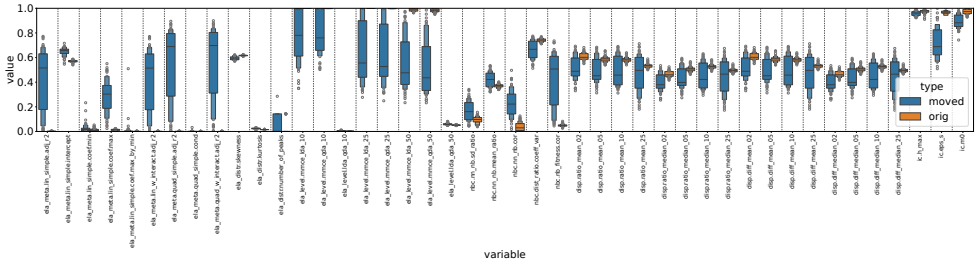
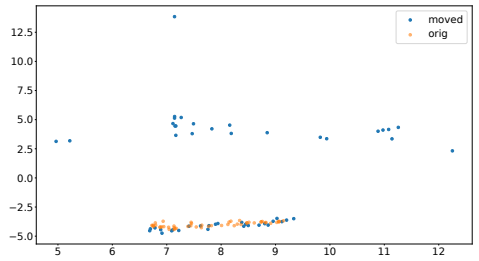
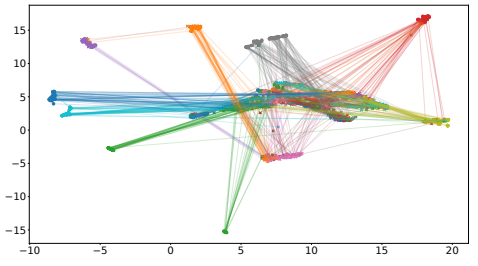


Figure 6.11: Distribution of normalized ELA features for the BBOB instance creation procedure and the same instance moved to have an optimum location uniformly in the domain, for F23.



(a) Projection of the moved and original version of F18 (50 instances).



(b) Projection of all BBOB functions connecting the original (circles) and moved (crosses) versions of each function.

Figure 6.12: UMAP projection trained on original BBOB, then used to plot both the moved and original functions in 2D.

sional space into two dimensions using PCA on the original BBOB versions. Then, we can plot the moved versions of the function into the same space, and observe the differences. The result of this projection is shown in Figure 6.12, where we see that many functions are moved much closer to the center of the projected space. This suggests that some of the 'unique' feature combinations present in the original BBOB functions are being lost when moving their optimum. This happens because large parts of the function are moved outside of the domain, and replaced by parts which were originally located outside the bounds. For some functions, these components are exponentially increasing, leading to a large part of the space which is dominated by these artifacts, which is represented in the ELA-features.

6.2.5 Pairwise Combinations

While combining functions with a sphere model can be viewed as adding global structure to a problem, combinations between other functions can provide interesting insights into the transition points between different types of problems. To illustrate the kinds of insights that can be gained from these combinations, we select a subset of 5 functions and collect performance data on each combination with the same 21 α values (with both orderings of the function). We show the performance in terms of normalized AOCC of diagonal CMA-ES on these function combinations in Figure 6.13. Note that for $\alpha = 1$, we are using the function specified in the column label, while for $\alpha = 0$ we have the function specified in the row label.

When comparing this figure to its equivalent from the GECCO paper [251], it is

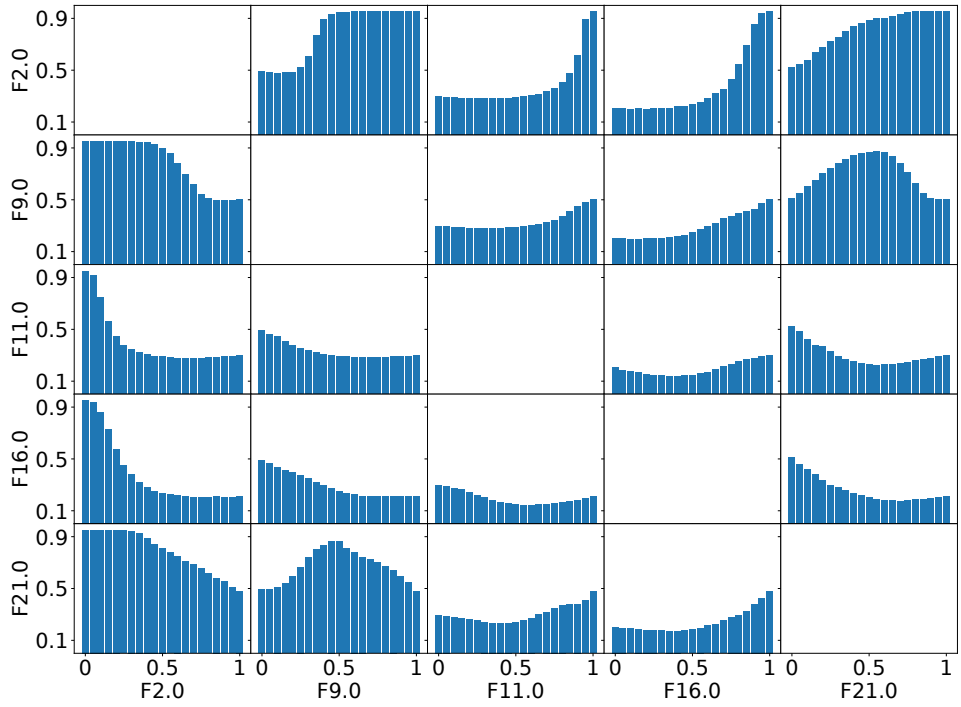


Figure 6.13: Normalized area over the convergence curve for Diagonal CMA-ES on each of the affine combinations between the selected BBOB problems. Each facet corresponds to the combination of the row and column function, with the x-axis indicating the used α. AOCC values are calculated based on 50 runs on 25 instances, with a budget of 10 000 function evaluations.

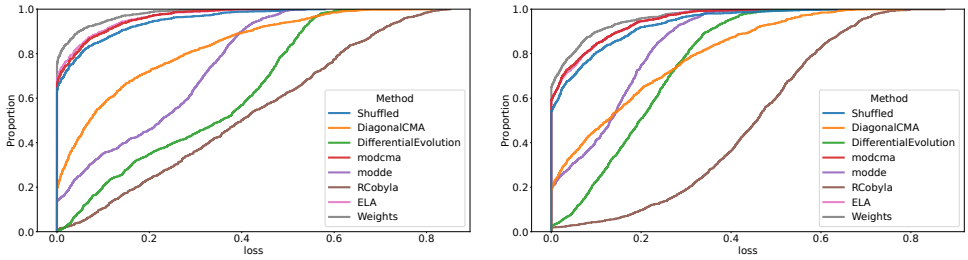
important to note the fact that Figure 6.13 is almost fully symmetric around the diagonal, which was not the case in the GECCO paper. Even though we might expect (F_1, F_2, α) to be similar to $(F_2, F_1, 1-\alpha)$, this was not the case when the location of the global optimum was selected as the optimum of one of the component functions, as different BBOB functions can have significantly different distributions of potential global optima [150]. This is in large part the reason why we enabled the MA-BBOB generator to sample the optimum uniformly at random in the domain. While Section 6.2.4 showed that this can potentially move interesting parts of some component functions outside the domain, we view this as a worthwhile tradeoff to achieve fully unbiased global optima.

From Figure 6.13, we can also see that the transition of performance between the two extreme α values is mostly smooth. While there are some rather quick changes, e.g., for the transition between F2 and F11, these seem to be the exception rather than the rule. Particularly interesting are the settings where the performance of affine combinations between two functions proves to be much easier or harder than the functions which are being combined. For example, this is the case for the combinations of F21 and F9.

6.3 Combining Multiple Functions: Testing Generalizability

For our final set of experiments, we make use of a set of 1000 functions generated using the setup described in Section 6.1.2. This data is taken directly from [250], and contains both ELA and performance data (for the same set of algorithms described in Section 6.2.1, but using the original AUC measure instead of the AOCC). In [250] we analyzed this data to understand the MA-BBOB instance generation procedure, with the goal of generating a wide set of benchmark problems on which algorithm selection and other automated machine learning techniques can be tested.

In this experiment, we take the perspective of algorithm selection and train a random forest model to predict the best algorithm to use for each function, based either on the ELA features of the problem or the weights of the component functions. We can then compare the loss in terms of AUC relative to the virtual best solver (VBS) for both of these models, in different training contexts. We can either use the common cross-validation setup, or attempt to test for generalization ability based only on the original BBOB functions. In Figure 6.14a we show the cumulative loss



(a) AUC loss for 5 dimensional functions. **(b)** AUC loss for 2 dimensional functions.

Figure 6.14: Cumulative loss (AUC) for different models: cross-validation (mixture of BBOB + MA-BBOB generated combinations) based on weights and ELA, and each of the single-algorithm models.

for the cross-validation setup on the 5-dimensional functions. From this, we can see that the ELA-based selector performs worse than the one based on the weights. This confirms the previous observation that the ELA features might not be sufficiently representative to accurately represent the problems in a way which is relevant for ranking optimization algorithms.

In order to better estimate how much the structure of the ELA features helps the prediction, we can add in a naive baseline. This is created by shuffling the labels (best ranked algorithm) of all samples before training. This shuffled model is in essence just a selector based on the frequency of labels in the training data, and the difference between this version and the original ELA-based selector shows how much the structure of the ELA-features helps improve the predictions. The results for the cross-validation setup in 2D are shown in Figure 6.14b, where we see that the benefit over most of the individual algorithms is inherent to the selected portfolio, since the shuffled model outperforms all algorithms except modCMA. This suggests that our algorithm portfolio is severely unbalanced.

When looking at the generalization task, this imbalance is exacerbated further, since for MA-BBOB the modCMA is ranked first on an even larger fraction of functions than on BBOB, as shown in Figure 6.15. In combination with the added challenge of transferring to a new suite, this leads to our algorithm selection models being outperformed by the modCMA, which is the Single Best Solver (SBS) in this case, as illustrated in Figure 6.16. While the algorithm portfolio is partly responsible for this shortcoming, the generalization ability does not significantly improve when removing the modCMA from our portfolio. This suggests that training on the original BBOB

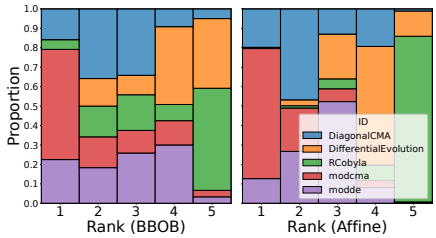


Figure 6.15: Distribution of ranks based on per-function AUC after 10 000 evaluations.

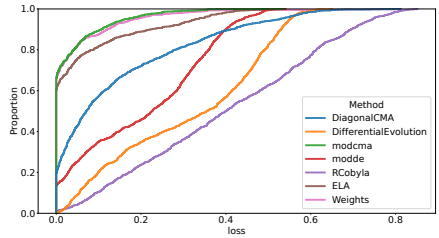


Figure 6.16: Cumulative loss (AUC) on the 5 dimensional MA-BBOB problems for models trained on the BBOB functions, and each of the single-algorithm models.

instances does not sufficiently represent the challenges faced in the MA-BBOB suite. An important aspect of the challenge of this transfer is the location of the optima, as discussed in Section 6.2.4.

Chapter 7

Conclusions

Throughout this thesis, we have explored iterative optimization heuristics for continuous optimization. We have shown that rigorous benchmarking does not only improve our understanding of the algorithm but also highlights avenues for further exploration. Through the use of modular design spaces for algorithms, we can fine-tune our optimizers to specific landscape properties, or create high-level selectors which exploit complementarity between the available solvers. In fact, observing the anytime performance of large algorithm portfolios shows that a dynamic approach to the algorithm selection problem has the potential to lead to even larger performance gains.

In Chapter 3, we addressed our first research question: *How can robust benchmarking pipelines be made accessible and resulting data be made usable by the wider community?* We introduced IOHprofiler as a modular environment for benchmarking iterative optimization heuristics, which provides both a way for setting up benchmarking studies via IOHexperimenter as well as an accessible interface for the analysis of the resulting benchmark data in IOHanalyzer. Using this framework, we showed how a robust benchmarking pipeline can be used in combination with a wide variety of problems. By focusing on the BBOB suite, we investigated some commonly overlooked aspects of the instance generation procedure and how this interacts with commonly used landscape analysis methods. This highlights the relation between the setup of a benchmark study and the ways in which we draw conclusions from the resulting data.

To close out Chapter 3, we note that benchmarking is more than just performance-oriented comparisons between algorithms. By looking at the concept of structural bias, we illustrate how behavior-based benchmarking can be used to gain insights into the inner workings of an algorithm, and the potential biases therein.

In Chapter 4, we explored our second research question: *How can a modular design aid in the exploration of interactions between different algorithmic ideas?* We discussed two modular algorithms: modCMA and modDE, each of which encompasses a design space with thousands of potential configurations. By making use of algorithm configuration techniques, we illustrated the complementarity which exists between different modules, since the best-performing configuration differs significantly per benchmark function. We also highlight that tuned configurations of these modular algorithms can clearly outperform existing versions of the respective algorithms. Additionally, the tuning procedure can be used as a way of incrementally assessing a new modules contribution to the existing design space. While these results show a promising direction for future assessment of algorithmic ideas, there are several challenges inherent to our proposed approach. Most critically, we showed that the inherent stochasticity of the considered algorithms has a drastic impact on the stability of the results obtained by algorithm configuration methods, suggesting a need for better noise-hanling methods.

The research question discussed in Chapter 5 was: *To what extent can we exploit performance complementary between different algorithms by switching between them?* Starting from a large set of benchmarking data, we showed that different algorithms perform well during different parts of the optimization process. By assuming we can freely switch between them, we observed significant potential performance gains from dynamic algorithm selection. By first focussing on switching between configurations of a modular algorithm, we sidestepped the question of warmstarting to show that performance can indeed improve by changing the configuration during the search. We then investigated per-run dynamic algorithm selection, where we utilize information from the first algorithm to determine which algorithm to switch to, where the switch incorporates a warmstart of the state of the secondary algorithm. Finally, we tackled the question of when the switch should occur, by transitioning to a sliding-window approach where we predict the relative benefit of performing a switch versus sticking with the original algorithm. Such a model could in future be used to create fully dynamic algorithm selectors which can switch multiple times throughout the search.

The final research question, discussed in Chapter 6, was: *How can we fairly judge the performance of meta-learning methods in the context of black-box optimization?* This question was inspired by observations from the previous chapter, where different algorithm selection techniques showed promising performance on the BBOB suite, but failed to generalize to a similar suite from a different platform. To better understand this seeming lack of generalizability, we introduced MA-BBOB, a problem generator based on the BBOB suite which uses affine recombination to create new benchmark

problems with known global optima. This generator was subsequently used to investigate the link between function landscapes and algorithm performance, resulting in an experiment which shows that generalizability of algorithm selection results is still lacking.

7.1 Key Findings

7.1.1 Chapter 3

- Benchmarking can be made accessible, and by doing so we can gain new insights into both algorithms and problems. Specifically, IOHprofiler allows us to investigate problems from new domains, such as star-discrepancy computation, and tackle them using state-of-the-art solvers. This highlights where further algorithm development is required, since the used algorithm portfolio failed to convincingly outperform a random search baseline.

- Benchmarking environments can not only function to interface problems with algorithms, with data annotation and sharing options such as ontologies, which allow for a wide variety of ways in which to combine data from separate sources and gain new insights.

- Instance generation mechanisms such as the one used in the BBOB suite can be very useful to test algorithm invariances, but care should be taken when using them in a box-constrained setting, since the used transformations necessarily change the part of the landscape the algorithm interacts with. This can be seen when looking at the ELA features of different BBOB instances, many of which are different in a statistically significant way. Looking at the instance generation procedure also highlights potential biases in the BBOB suite, e.g. with regard to the possible locations of the global optima, which should be kept in mind when using them in future benchmarking studies.

- Benchmarking is not limited to only looking at the performance of an algorithm. We can design more behavior-oriented benchmarks to gain an understanding of different algorithmic aspects, for example, its structural bias to certain regions of its domain. Knowing whether an algorithm is biased towards e.g. the center of the domain can be combined with knowledge about the biases of different benchmark suites to identify potentially misleading comparisons.

7.1.2 Chapter 4

- In addition to enabling a fair comparison of different algorithmic ideas within the same overarching framework, modular algorithm design also creates opportunities to explore interactions between many algorithm modifications which have been proposed in isolation. By combining this design principle with algorithm configuration tools, we find that well-configured module settings can lead to significant improvement over common variants of the same base algorithm.

- While algorithms can be configured to perform well over a large set of benchmark problems, configuring for performance on individual functions leads to additional improvements, highlighting the inherent complementarity present in these large configuration spaces.

- Algorithm configuration is an inherently noisy problem, and while most algorithm configurations incorporate various strategies to overcome this noise, the relation between the level of variance and the best noisy selection technique is not yet clear, which can lead to 'lucky' configurations being selected over those with actual better performance.

7.1.3 Chapter 5

- Just as algorithm selection can take advantage of complementarity between solvers on different types of problems, dynamic algorithm selection can take advantage of complementarity on different parts of the search. While some of these advantages remain theoretical, we can obtain improvements in performance on several functions by switching between algorithms at certain stages.

- Switching between algorithm variants can be simplified by working with modular algorithms, where the question of warm-starting the second algorithm can be addressed by preserving the internal state of the algorithm. While finding optimal dynamic combinations of parameter settings becomes challenging because of the increased noise, we have shown that dynamic combinations can outperform their static counterparts on the majority of used benchmark problems.

- A dynamic algorithm combination does not always have to be determined before running the algorithm. By utilizing information collected from the trajectory of an initial algorithm, the most promising secondary algorithm can be selected

from a larger portfolio. This could allow us to take advantage of the per-run stochasticity of our algorithms, leading to a per-run algorithm selection scenario.

- If we can accurately predict an algorithm's performance from a small initial trajectory, we can use these models to identify whether a switch at any given point in the search would be worthwhile, which could form the basis for a truly online dynamic algorithm selector. Unfortunately, this adds another level of complexity to the meta-learning task and our current models are not robust enough for this purpose.

7.1.4 Chapter 6

- The Many-Affine BBOB generator provides a new set of problems to validate current results on the generalizability of algorithm selection methods trained on BBOB. By showcasing that this generalizability is severely lacking for simple ELA-based models, we highlight the importance of further research into both the features we use to represent problems based on limited samplings, and the potential over-reliance on the same set of problems to guide algorithmic developments.

7.2 Future Work

As this thesis takes a wide view on benchmarking and its implications for various meta-learning scenarios, many open questions and areas for further research remain. Here, we highlight a few of the most relevant ones:

- **Data Accessibility.** As this thesis illustrates, there are many potential uses of benchmark data beyond the comparison of algorithm performance. As such, data which is collected for one study can, and often should, be re-used in other areas. Benchmarking tools play a critical role in facilitating this aspect, and while tools such as OpenML [237] have been widely adopted in the machine learning community, examples of data repositories in the optimization domain are less common. Repositories such as COCO's data archive [4], while valuable sources of performance data, are still somewhat limited in their usability because of a lack of meta-data, specifically information about the used algorithm implementation, and available code. From the metadata perspective, data ontologies such as OPTION [136] could provide a way to link different repositories together, while

efforts to increase reproducibility [151] within the wider community might make sharing code and data the norm rather than an exception.

- **Judging New Algorithms.** As the field of iterative optimization heuristics continues to grow, an ever-increasing number of algorithms will be proposed. While many of these might contain interesting algorithmic novelties, the way in which these contributions are judged has to adapt to keep up with the increases in scale. Current papers too often rely on comparison to a very limited set of baselines, where only average performance over a large set of functions is considered. This comparison should be extended to include established algorithms and be judged not on aggregate performance, but for example on contribution to this overall portfolio of established algorithms. In addition to this, a larger focus should be placed on unambiguous descriptions of the algorithmic components, rather than metaphors used to motivate their novelty.

- **Achieving DynAS.** In Chapter 5, we showed the potential performance to be gained from different versions of dynamic algorithm selection. However, throughout the experiments described in that chapter, we notice that the resulting performance is not yet stable, sometimes failing to beat even the static algorithms. Since DynAS consists of several interacting components, improvements in each of these aspects are needed in order to achieve usable dynamic switching behavior, with the two core components being:

  - Warmstarting: when performing a switch, we need to be careful not to lose information obtained at the beginning of the search. Depending on the algorithms used, we might need to initialize a population, stepsizes, covariance information... The way in which this information transfer is performed has a significant impact on the final performance, and as such should be carefully designed.
  - Deciding when to switch: in order to achieve optimal performance, a switch between algorithms might need to occur at different points in the search for different runs. As such, the current algorithm needs to identify the point at which it is most promising to transition to a new algorithm, based only on the information it has obtained so far. While trajectory-based ELA features seem to be somewhat promising, their variance poses a significant challenge, and other techniques might need to be considered as well.

- **Understanding Generalizability.** With the growing popularity of methods

such as algorithm selection or even dynamic switching, the question of how to fairly judge these meta-algorithms becomes more prominent. We have seen that classical methods such as leave-one-problem-out don't fit well with the available benchmark sets, and performance on one suite does not necessarily translate to others. In order to create a larger testbed, more attention has to be placed on the analysis of different benchmark suites and their complementarity, resulting in larger sets of training and testing functions. While problem generators like MA-BBOB might be a useful component in this process, further analysis into the wide variety of benchmarks is required to make more informed decisions.

Bibliography

[1] Theodore W. Anderson and Donald A. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *The annals of mathematical statistics*, pages 193–212, 1952.

[2] Warren Armstrong, Peter Christen, Eric McCreath, and Alistair P Rendell. Dynamic algorithm selection using reinforcement learning. In *2006 international workshop on integrating ai and data mining*, pages 18–25. IEEE, 2006.

[3] Anne Auger, Dimo Brockhoff, and Nikolaus Hansen. Mirrored sampling in evolution strategies with weighted recombination. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 861–868. ACM, 2011.

[4] Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Tea Tušar, and Konstantinos Varelas. Data from BBOB-workshops on 24 noiseless functions, 2020. https://numbbo.github.io/data-archive/bbob/.

[5] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, pages 1769–1776. IEEE, 2005.

[6] Anne Auger, Mohamed Jebalia, and Olivier Teytaud. Algorithms (x, sigma, eta): Quasi-random mutations for evolution strategies. In El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer, editors, *Artificial Evolution, 7th International Conference, Evolution Artificielle, EA 2005, Lille, France, October 26-28, 2005, Revised Selected Papers*, volume 3871 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 2005.

[7] Amine Aziz-Alaoui, Carola Doerr, and Johann Dréo. Towards large scale automated algorithm design by integrating modular benchmarking frameworks. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 1365–1374. ACM, 2021.

[8] Thomas Bartz-Beielstein, Carola Doerr, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, Manuel López-Ibáñez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. Benchmarking in optimization: Best practice and open issues. *CoRR*, abs/2007.03488, 2020.

[9] Nacim Belkhir. *Per Instance Algorithm Configuration for Continuous Black Box Optimization*. PhD thesis, University of Paris-Saclay, France, 2017.

[10] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. Per instance algorithm configuration of CMA-ES with limited budget. In Peter A. N. Bosman, editor, *Proceedings of Genetic and Evolutionary Computation (GECCO'17)*, pages 681–688. ACM, 2017.

[11] Yoav Benjamini. Discovering the false discovery rate. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):405–416, 2010.

[12] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

[13] Yoav Benjamini and Daniel Yekutieli. The Control of the False Discovery Rate in Multiple Testing under Dependency. *The Annals of Statistics*, 29(4):1165–1188, 2001.

[14] Przemyslaw Biecek and Teresa Ledwina. ddst: Data driven smooth tests. R package version 1.0.2.

[15] André Biedenkapp, H. Furkan Bozkurt, Frank Hutter, and Marius Lindauer. Towards white-box benchmarks for algorithm control. *CoRR*, abs/1906.07644, 2019.

[16] Rafal Biedrzycki. Comparison with state-of-the-art: Traps and pitfalls. In *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021*, pages 863–870. IEEE, 2021.

[17] Rafał Biedrzycki, Jarosław Arabas, and Dariusz Jagodziński. Bound constraints handling in differential evolution: An experimental study. *Swarm and Evolutionary Computation*, 50:100453, 2019.

[18] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Mike Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In Terence Soule and Jason H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 313–320. ACM, 2012.

[19] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936.

[20] Jakob Bossek, Carola Doerr, Pascal Kerschke, Aneta Neumann, and Frank Neumann. Evolving sampling strategies for one-shot optimization tasks. In Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2020.

[21] Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical hyper-parameters: No random, no cry. *CoRR*, abs/1706.03200, 2017.

[22] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[23] Janez Brest, Saso Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[24] Janez Brest and Mirjam Sepesy Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, 2008.

[25] Janez Brest, Mirjam Sepesy Maučec, and Borko Boskovic. Single objective real-parameter optimization: Algorithm jso. In *2017 IEEE Congress on Evolutionary Computation, CEC 2017, Donostia, San Sebastián, Spain, June 5-8, 2017*, pages 1311–1318. IEEE, 2017.

[26] Dimo Brockhoff. A bug in the multiobjective optimizer IBEA: salutary lessons for code release and a performance re-assessment. In António Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos A. Coello Coello, editors, *Evolutionary Multi-Criterion Optimization - 8th International Conference, EMO 2015, Guimarães, Portugal, March 29 -April 1, 2015. Proceedings, Part I*, volume 9018 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2015.

[27] Dimo Brockhoff. Comparing boundary handling techniques of CMA-ES on the bbob and sbox-cost test suites. In Sara Silva and Luís Paquete, editors, *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*, pages 2318–2325. ACM, 2023.

[28] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V. Arnold, and Tim Hohm. Mirrored sampling and sequential selection for evolution strategies. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*, volume 6238 of *Lecture Notes in Computer Science*, pages 11–21. Springer, 2010.

[29] Charles G. Broyden. The Convergence of a Class of Double-rank Minimization Algorithms: 2. The New Algorithm. *IMA Journal of Applied Mathematics*, 6(3):222–231, 09 1970.

[30] H. D. Brunk. On the Range of the Difference between Hypothetical Distribution Function and Pyke's Modified Empirical Distribution Function. *The Annals of Mathematical Statistics*, 33(2):525 – 532, 1962.

[31] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[32] Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[33] Borja Calvo, Josu Ceberio, and José Antonio Lozano. Bayesian inference for algorithm ranking analysis. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 324–325. ACM, 2018.

[34] Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 26(3):402–416, 2022.

[35] Christian Leonardo Camacho-Villalón, Thomas Stützle, and Marco Dorigo. Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In Marco Dorigo, Thomas Stützle, Maria J. Blesa, Christian Blum, Heiko Hamann, Mary Katherine Heinrich, and Volker Strobel, editors, *Swarm Intelligence - 12th International Conference, ANTS 2020, Barcelona, Spain, October 26-28, 2020, Proceedings*, volume 12421 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2020.

[36] Felipe Campelo and Moisés Botelho. Experimental investigation of recombination operators for differential evolution. In Tobias Friedrich, Frank Neumann, and Andrew M. Sutton, editors, *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 221–228. ACM, 2016.

[37] Guogang Cao, Cong Cao, Qing Zhang, and Wenju Li. Differential evolution improved with intelligent mutation operator based on proximity and ranking. In *11th International Symposium on Computational Intelligence and Design, ISCID 2018, Hangzhou, China, December 8-9, 2018, Volume 2*, pages 196–201. IEEE, 2018.

[38] Fabio Caraffini and Giovanni Iacca. The SOS Platform: Designing, Tuning and Statistically Benchmarking Optimisation Algorithms. *Mathematics*, 8(5):785, May 2020.

[39] Fabio Caraffini and Anna V. Kononova. Structural bias in differential evolution: A preliminary study. *AIP Conference Proceedings*, 2070(1):020005, 2019.

[40] Fabio Caraffini, Anna V. Kononova, and David Corne. Infeasibility and structural bias in differential evolution. *Information Sciences*, 496:161–179, 2019.

[41] Javier Castro, Daniel Gómez, and Juan Tejada. Polynomial calculation of the shapley value based on sampling. *Computers & Operations Research*, 36(5):1726–1730, 2009.

[42] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh - A python package). *Neurocomputing*, 307:72–77, 2018.

[43] François Clément, Diederick Vermetten, Jacob de Nobel, Alexandre D. Jesus, Luís Paquete, and Carola Doerr. Computing star discrepancies with numerical black-box optimization algorithms. In Sara Silva and Luís Paquete, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pages 1330–1338. ACM, 2023.

[44] Harald Cramér. On the composition of elementary errors: First paper: Mathematical deductions. *Scandinavian Actuarial Journal*, 1928(1):13–74, 1928.

[45] Noel Cressie. An optimal statistic based on higher order gaps. *Biometrika*, 66(3):619–627, 12 1979.

[46] Noel Cressie and Timothy R. C. Read. Multinomial goodness-of-fit tests. *Journal of the Royal Statistical Society. Series B (Methodological)*, 46(3):440–464, 1984.

[47] Sandor Csorgo and Julian J. Faraway. The Exact and Asymptotic Distributions of Cramer-von Mises Statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):221–234, 1996.

[48] Swagatam Das, Ajith Abraham, Uday K. Chakraborty, and Amit Konar. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, 2009.

[49] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N. Suganthan. Recent advances in differential evolution - an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016.

[50] Pierre Lafaye de Micheaux and Viet Anh Tran. PoweR: A reproducible research tool to ease monte carlo power simulation studies for goodness-of-fit tests in r. *Journal of Statistical Software, Articles*, 69(3):1–44, 2016.

[51] Jacob de Nobel, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 1375–1384. ACM, 2021.

[52] Jacob de Nobel, Hao Wang, and Thomas Bäck. Explorative data analysis of time series based algorithm features of CMA-ES variants. In Francisco Chicano and Krzysztof Krawiec, editors, *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 510–518. ACM, 2021.

[53] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Iohexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation*, pages 1–6, 2024.

[54] James Demmel and Krešimir Veselić. Jacobi's method is more accurate than qr. *SIAM Journal on Matrix Analysis and Applications*, 13(4):1204–1245, 1992.

[55] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[56] Luc Devroye. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1972.

[57] Konstantin Dietrich and Pascal Kerschke. Evaluation of algorithms from the nevergrad toolbox on the strictly box-constrained SBOX-COST benchmarking suite. In Sara Silva and Luís Paquete, editors, *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*, pages 2326–2329. ACM, 2023.

[58] Konstantin Dietrich and Olaf Mersmann. Increasing the diversity of benchmark function sets through affine recombination. In Günter Rudolph, Anna V. Kononova, Hernán E. Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tusar, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN'22)*, volume 13398 of *LNCS*, pages 590–602. Springer, 2022.

[59] David P. Dobkin, David Eppstein, and Don P. Mitchell. Computing the discrepancy with applications to supersampling patterns. *ACM Trans. Graph.*, 15(4):354–376, 1996.

[60] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020.

[61] Benjamin Doerr, Mahmoud Fouz, Martin Schmidt, and Magnus Wahlström. BBOB: nelder-mead with resize and halfruns. In Franz Rothlauf, editor, *Proceedings of Genetic and Evolutionary Computation (GECCO'09)*, pages 2239–2246. ACM, 2009.

[62] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281*, abs/1810.05281, 2018.

[63] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with IOHprofiler. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)*, volume 88, pages 1798–1806. ACM, 2019. Full version to appear with *Applied Soft Computing*.

[64] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.

[65] Johann Dréo, Arnaud Liefooghe, Sébastien Vérel, Marc Schoenauer, Juan Julián Merelo Guervós, Alexandre Quemy, Benjamin Bouvier, and Jan Gmys. Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of paradiseo. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 1522–1530. ACM, 2021.

[66] Gunter Dueck and Tobias Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[67] James Durbin and R Brown. Tests of serial independence based on the cumulated periodogram. *Bulletin of the International Statistical Institute*, 42:1039–1048, 1967.

[68] Tome Eftimov, Peter Korosec, and Barbara Korousic-Seljak. A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics. *Information Sciences*, 417:186–215, 2017.

[69] Tome Eftimov, Gasper Petelin, and Peter Korosec. DSCTool: A web-service-based framework for statistical comparison of stochastic optimization algorithms. *Applied Soft Computing*, 87:105977, 2020.

[70] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[71] Agoston Endre Eiben and James E. Smith. *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer, 2015.

[72] Ouassim Ait ElHara, Anne Auger, and Nikolaus Hansen. A median success rule for non-elitist evolution strategies: study of feasibility. In Christian Blum and Enrique Alba, editors, *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 415–422. ACM, 2013.

[73] Michael G. Epitropakis, Dimitris K. Tasoulis, Nicos G. Pavlidis, Vassilis P. Plagianakos, and Michael N. Vrahatis. Enhancing differential evolution utilizing proximity-based mutation operators. *IEEE Transactions on Evolutionary Computation*, 15(1):99–119, 2011.

[74] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander J. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *CoRR*, abs/2003.06505, 2020.

[75] Julian Faraway, George Marsaglia, John Marsaglia, and Adrian Baddeley. goftest: Classical goodness-of-fit tests for univariate distributions. R package version 1.2.3.

[76] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *J. Mach. Learn. Res.*, 23:261:1–261:61, 2022.

[77] Roger Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

[78] Carlos M. Fonseca, Andreia P. Guerreiro, Manuel López-Ibáñez, and Luís Paquete. On the computation of the empirical attainment function. In Ricardo H. C. Takahashi, Kalyanmoy Deb, Elizabeth F. Wanner, and Salvatore Greco, editors, *Evolutionary Multi-Criterion Optimization - 6th International Conference, EMO 2011, Ouro Preto, Brazil, April 5-8, 2011. Proceedings*, volume 6576 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2011.

[79] Alexandre Fréchette, Lars Kotthoff, Tomasz P. Michalak, Talal Rahwan, Holger H. Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3397–3403. AAAI Press, 2016.

[80] Tobias Friedrich and Frank Neumann. Maximizing submodular functions under matroid constraints by multi-objective evolutionary algorithms. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipic, and Jim Smith, editors, *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*, volume 8672 of *Lecture Notes in Computer Science*, pages 922–931. Springer, 2014.

[81] Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Ann. Math. Artif. Intell.*, 47(3-4):295–328, 2006.

[82] Roger Gämperle, Sibylle D Müller, and Petros Koumoutsakos. A parameter study for differential evolution. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 10(10):293–298, 2002.

[83] Tobias Glasmachers, Tom Schaul, Yi Sun, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In Martin Pelikan and Jürgen Branke, editors, *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*, pages 393–400. ACM, 2010.

[84] Michael Gnewuch, Magnus Wahlström, and Carola Winzen. A new randomized algorithm to approximate the star discrepancy based on threshold accepting. *SIAM J. Numer. Anal.*, 50(2):781–807, 2012.

[85] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[86] Major Greenwood. The statistical study of infectious diseases. *Journal of the Royal Statistical Society*, 109(2):85–110, 1946.

[87] Shu-Mei Guo and Chin-Chang Yang. Enhancing differential evolution utilizing eigenvector-based crossover operator. *IEEE Transactions on Evolutionary Computation*, 19(1):31–49, 2015.

[88] Nikolaus Hansen. CMA-ES with two-point step-size adaptation. *CoRR*, abs/0805.0231, 2008.

[89] Nikolaus Hansen. Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In Franz Rothlauf, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*, pages 2389–2396. ACM, 2009.

[90] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[91] Nikolaus Hansen and Anne Auger. Principled design of continuous stochastic search: From theory to practice. In Yossi Borenstein and Alberto Moraglio, editors, *Theory and Principled Methods for the Design of Metaheuristics*, Natural Computing Series, pages 145–180. Springer, 2014.

[92] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tusar, and Tea Tusar. COCO: performance assessment. *CoRR*, abs/1605.03560, 2016.

[93] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tusar. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Transactions on Evolutionary Computation*, 26(6):1293–1305, 2022.

[94] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Posík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In Martin Pelikan and Jürgen Branke, editors, *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010, Companion Material*, pages 1689–1696. ACM, 2010.

[95] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[96] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

[97] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of 1996 IEEE International Conference on Evolutionary Computation, Nayoya University, Japan, May 20-22, 1996*, pages 312–317. IEEE, 1996.

[98] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[99] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo-optimization modeling in python*, volume 67. Springer, 2017.

[100] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* Springer Series in Statistics. Springer, 2009.

[101] Y. A. S. Hegazy and J. R. Green. Some new goodness-of-fit tests using order statistics. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(3):299–308, 1975.

[102] Edmund Hlawka. Funktionen von beschrankter Variation in der Theorie der Gleichverteilung. *Annali di Matematica Pura ed Applicata*, 54:325–333, 1961.

[103] Myles Hollander, Douglas A Wolfe, and Eric Chicken. *Nonparametric statistical methods*, volume 751. John Wiley & Sons, 2013.

[104] Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Automated configuration and selection of SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 481–507. IOS Press, 2021.

[105] Alfred Inselberg. The plane with parallel coordinates. *The visual computer*, 1(2):69–91, 1985.

[106] Sk. Minhazul Islam, Swagatam Das, Saurav Ghosh, Subhrajit Roy, and Ponnuthurai Nagaratnam Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):482–500, 2012.

[107] Anja Jankovic and Carola Doerr. Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pages 841–849. ACM, 2020.

[108] Anja Jankovic, Tome Eftimov, and Carola Doerr. Towards feature-based performance regression using trajectory data. In Pedro A. Castillo and Juan Luis Jiménez Laredo, editors, *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7-9, 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 601–617. Springer, 2021.

[109] Anja Jankovic, Gorjan Popovski, Tome Eftimov, and Carola Doerr. The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection. In Francisco Chicano and Krzysztof Krawiec, editors, *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 687–696. ACM, 2021.

[110] Anja Jankovic, Diederick Vermetten, Ana Kostovska, Jacob de Nobel, Tome Eftimov, and Carola Doerr. Trajectory-based algorithm selection with warm-starting. In *IEEE Congress on Evolutionary Computation, CEC 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE, 2022.

[111] Carlos M. Jarque and Anil K. Bera. A test for normality of observations and regression residuals. *International Statistical Review/Revue Internationale de Statistique*, 55(2):163–172, 1987.

[112] Grahame A. Jastrebski and Dirk V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*, pages 2814–2821. IEEE, 2006.

[113] Donald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[114] Tomas Kadavy, Adam Viktorin, Anezka Kazikova, Michal Pluhacek, and Roman Senkerik. Impact of boundary control methods on bound-constrained optimization benchmarking. *IEEE Transactions of Evolutionary Computation*, 26(6):1271–1280, 2022.

[115] Alexander H. G. Rinnooy Kan and G. T. Timmer. Stochastic global optimization methods part I: clustering methods. *Math. Program.*, 39(1):27–56, 1987.

[116] Alexander H. G. Rinnooy Kan and G. T. Timmer. Stochastic global optimization methods part II: multi level methods. *Mathematical Programming*, 39(1):57–78, 1987.

[117] Leonid Kantorovitch. On the translocation of masses. *Management Science*, 5(1):1–4, 1958.

[118] Sitanshu S. Kar and Archana Ramalingam. Is 30 the magic number? issues in sample size estimation. *National Journal of Community Medicine*, 4(1):175–179, 2013.

[119] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2015.

[120] Zohar Shay Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1238–1246. JMLR.org, 2013.

[121] Maarten Keijzer, Juan Julián Merelo Guervós, Gustavo Romero, and Marc Schoenauer. Evolving objects: A general purpose evolutionary computation library. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, and Marc Schoenauer, editors, *Artificial Evolution, 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29-31, 2001, Selected Papers*, volume 2310 of *Lecture Notes in Computer Science*, pages 231–244. Springer, 2001.

[122] Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[123] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*, pages 1942–1948. IEEE, 1995.

[124] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[125] Pascal Kerschke and Heike Trautmann. The r-package flacco for exploratory landscape analysis with applications to multi-objective optimization problems. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5262–5269. IEEE, IEEE, July 2016.

[126] Pascal Kerschke and Heike Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1):99–127, 2019.

[127] Pascal Kerschke and Heike Trautmann. *Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-Package Flacco*, pages 93–123. Studies in Classification, Data Analysis, and Knowledge Organization. Springer International Publishing, Cham, 2019.

[128] Scott Kirkpatrick, C. D. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[129] Jurjen F. Koksma. Een algemeene stelling uit de theorie der gelijkmatige verdeeling modulo 1. *Mathematica (Zutphen)*, 11:7–11, 1942.

[130] Andrey Nikolaevich Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Giornale dell'Istituto Italiano degli Attuari*, 4:83–91, 1933.

[131] Anna V. Kononova, Fabio Caraffini, and Thomas Bäck. Differential evolution outside the box. *Information Sciences*, 581:587–604, 2021.

[132] Anna V. Kononova, Fabio Caraffini, Hao Wang, and Thomas Bäck. Can compact optimisation algorithms be structurally biased? In Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, volume 12269 of *LNCS*, pages 229–242, Cham, 2020. Springer.

[133] Anna V. Kononova, David W. Corne, Philippe De Wilde, Vsevolod Shneer, and Fabio Caraffini. Structural bias in population-based algorithms. *Information Sciences*, 298:468–490, 2015.

[134] Anna V. Kononova, Diederick Vermetten, Fabio Caraffini, Madalina-Andreea Mitran, and Daniela Zaharie. The importance of being constrained: Dealing with infeasible solutions in differential evolution and beyond. *Evolutionary Computation*, 32(1):3–48, 2024.

[135] Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, and Carola Doerr. Per-run algorithm selection with warm-starting using trajectory-based features. In Günter Rudolph, Anna V. Kononova, Hernán E. Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tusar, editors, *Parallel Problem Solving from Nature - PPSN XVII - 17th International Conference, PPSN 2022, Dortmund, Germany, September 10-14, 2022, Proceedings, Part I*, volume 13398 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2022.

[136] Ana Kostovska, Diederick Vermetten, Carola Doerr, Sašo Džeroski, Panče Panov, and Tome Eftimov. OPTION: OPTImization Algorithm Benchmarking ONtology. *IEEE Transactions on Evolutionary Computation*, 27(6):1618–1632, 2023.

[137] Oswin Krause, Tobias Glasmachers, and Christian Igel. Qualitative and quantitative assessment of step size adaptation rules. In *Proceedings of the 14th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '17, page 139–148, New York, NY, USA, 2017. Association for Computing Machinery.

[138] Benjamin Lacroix and John McCall. Limitations of benchmark sets and landscape features for algorithm selection and performance prediction. In *Proceedings of Genetic and Evolutionary Computation (GECCO'19)*, page 261–262, New York, NY, USA, 2019. Association for Computing Machinery.

[139] Pedro Larrañaga and José Antonio Lozano, editors. *Estimation of Distribution Algorithms*. Genetic Algorithms and Evolutionary Computation. Springer, 2002.

[140] Pedro Larrañaga and José Antonio Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[141] Rui Li, Michael T. M. Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan H. C. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, 2013.

[142] Shutao Li, Mingkui Tan, Ivor W. Tsang, and James Tin-Yau Kwok. A hybrid PSO-BFGS strategy for global optimization of multimodal functions. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 41(4):1003–1014, 2011.

[143] Xiaodong Li, Ke Tang, Mohammad N Omidvar, Zhenyu Yang, Kai Qin, and Hefei China. Benchmark functions for the cec 2013 special session and competition on large-scale global optimization. *gene*, 7(33):8, 2013.

[144] Jing J Liang, Bo Y Qu, and Ponnuthurai N Suganthan. Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 635:490, 2013.

[145] Jialin Liu, Antoine Moreau, Mike Preuss, Jérémy Rapin, Baptiste Rozière, Fabien Teytaud, and Olivier Teytaud. Versatile black-box optimization. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pages 620–628. ACM, 2020.

[146] Junhong Liu and Jouni Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Comput.*, 9(6):448–462, 2005.

[147] Marco Locatelli and Fabio Schoen. Random linkage: a family of acceptance/rejection algorithms for global optimisation. *Mathematical Programming*, 85(2), 1999.

[148] Fu Xing Long, Bas van Stein, Moritz Frenzel, Peter Krause, Markus Gitterle, and Thomas Bäck. Learning the characteristics of engineering optimization problems with applications in automotive crash. In Jonathan E. Fieldsend and Markus Wagner, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 1227–1236, New York, NY, USA, 2022. Association for Computing Machinery.

[149] Fu Xing Long, Diederick Vermetten, Anna V. Kononova, Roman Kalkreuth, Kaifeng Yang, Thomas Bäck, and Niki van Stein. Challenges of ela-guided function evolution using genetic programming. In Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe, editors, *Proceedings of the 15th International Joint Conference on Computational Intelligence, IJCCI 2023, Rome, Italy, November 13-15, 2023*, pages 119–130. SCITEPRESS, 2023.

[150] Fu Xing Long, Diederick Vermetten, Bas van Stein, and Anna V. Kononova. BBOB instance analysis: Landscape properties and algorithm performance across problem instances. In João Correia, Stephen L. Smith, and Raneem Qaddoura, editors, *Applications of Evolutionary Computation - 26th European Conference, EvoApplications 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12-14, 2023, Proceedings*, volume 13989 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2023.

[151] Manuel López-Ibáñez, Jürgen Branke, and Luís Paquete. Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(4):14:1–14:21, 2021.

[152] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[153] Manuel López-Ibáñez and Thomas Stützle. The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[154] Ilya Loshchilov. A computationally efficient limited memory CMA-ES for large scale optimization. In Dirk V. Arnold, editor, *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, pages 397–404. ACM, 2014.

[155] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. Bi-population CMA-ES agorithms with surrogate models and line searches. In Christian Blum and Enrique Alba, editors, *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material Proceedings*, pages 1177–1184. ACM, 2013.

[156] Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4J: a modular framework for meta-heuristic optimization. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 1723–1730. ACM, 2011.

[157] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on*

Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 4765–4774, 2017.

[158] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[159] M. A. Marhuenda, Y. Marhuenda, and D. Morales. Uniformity tests under quantile categorization. *Kybernetes*, 34(6):888–901, 2005.

[160] Oded Maron and Andrew W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

[161] Frank J. Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[162] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, and Michèle Sebag. Adaptive operator selection and management in evolutionary algorithms. In Youssef Hamadi, Eric Monfroy, and Frédéric Saubion, editors, *Autonomous Search*, pages 161–189. Springer, 2012.

[163] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[164] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 829–836. ACM, 2011.

[165] Olaf Mersmann, Mike Preuss, and Heike Trautmann. Benchmarking evolutionary algorithms: Towards exploratory landscape analysis. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*, volume 6238 of *Lecture Notes in Computer Science*, pages 73–82. Springer, 2010.

[166] Laurent Meunier, Herilalaina Rakotoarison, Pak-Kan Wong, Baptiste Rozière, Jérémy Rapin, Olivier Teytaud, Antoine Moreau, and Carola Doerr. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking. *IEEE Transactions on Evolutionary Computation*, 26(3):490–500, 2022.

[167] Madalina-Andreea Mitran, Anna V. Kononova, Fabio Caraffini, and Daniela Zaharie. Patterns of convergence and bound constraint violation in differential evolution on SBOX-COST benchmarking suite. In Sara Silva and Luís Paquete, editors, *Companion Proceedings of the Conference on Genetic and Evolutionary*

Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023, pages 2337–2345. ACM, 2023.

[168] Domingo Morales, Leandro Pardo, María C. Pardo, and Igor Vajda. Limit laws for disparities of spacings. *Journal of Nonparametric Statistics*, 15(3):325–342, 2003.

[169] Patrick A. P. Moran. The random division of an interval–part ii. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):147–150, 1951.

[170] Mario A. Muñoz, Michael Kirley, and Saman K. Halgamuge. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87, 2015.

[171] Mario A. Muñoz and Kate Smith-Miles. Effects of function translation and dimensionality reduction on landscape analysis. In *IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, May 25-28, 2015*, pages 1336–1342. IEEE, 2015.

[172] Mario A. Muñoz and Kate Smith-Miles. Generating new space-filling test instances for continuous black-box optimization. *Evol. Comput.*, 28(3):379–404, 2020.

[173] Mario A. Muñoz and Kate Amanda Smith-Miles. Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary computation*, 25(4), 2017.

[174] Mario A. Muñoz, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245, 2015.

[175] Mario Andrés Muñoz, Michael Kirley, and Kate Smith-Miles. Analyzing randomness effects on the reliability of exploratory landscape analysis. *Natural Computing*, 21(2):131–154, 2022.

[176] Aneta Neumann, Frank Neumann, and Chao Qian. Evolutionary submodular optimisation. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 918–940. ACM, 2021.

[177] Frank Neumann, Aneta Neumann, Chao Qian, Anh Viet Do, Jacob de Nobel, Diederick Vermetten, Saba Sadeghi Ahouei, Furong Ye, Hao Wang, and Thomas Bäck. Benchmarking algorithms for submodular optimization problems using IOHProfiler. In *IEEE Congress on Evolutionary Computation, CEC 2023, Chicago, IL, USA, July 1-5, 2023*, pages 1–9. IEEE, 2023.

[178] Jerzy Neyman. Smooth test for goodness of fit. *Scandinavian Actuarial Journal*, 1937(3-4):149–199, 1937.

[179] Harald Niederreiter. Discrepancy and convex programming. *Annali di matematica pura ed applicata*, 93:89–97, 1972.

[180] Ana Nikolikj, Carola Doerr, and Tome Eftimov. RF+clust for leave-one-problem-out performance prediction. In João Correia, Stephen L. Smith, and Raneem Qaddoura, editors, *Applications of Evolutionary Computation - 26th European Conference, EvoApplications 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12-14, 2023, Proceedings*, volume 13989 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 2023.

[181] Eric W. Noreen. *Computer-Intensive Methods for Testing Hypotheses: An Introduction*. Wiley, 1989.

[182] Tom Packebusch and Stephan Mertens. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49(16):165001, 2016.

[183] László Pál. Benchmarking a hybrid multi level single linkagealgorithm on the bbob noiseless testbed. In Christian Blum and Enrique Alba, editors, *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material Proceedings*, pages 1145–1152. ACM, 2013.

[184] Maria C. Pardo. A test for uniformity based on informational energy. *Statistical Papers*, 44:521–534, 2003.

[185] Karl Pearson. Das Fehlergesetz und Seine Verallgemeinerungen Durch Fechner und Pearson. A Rejoinder. *Biometrika*, 4(1-2):169–212, 06 1905.

[186] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[187] Alejandro Piad-Morffis, Suilan Estevez-Velarde, Antonio Bolufé Röhler, James Montgomery, and Stephen Chen. Evolution strategies with thresheld convergence. In *IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, May 25-28, 2015*, pages 2097–2104. IEEE, 2015.

[188] Maxim Pikalov and Vladimir Mironovich. Parameter tuning for the (1 + (λ , λ)) genetic algorithm using landscape analysis and machine learning. In Juan Luis Jiménez Laredo, José Ignacio Hidalgo, and Kehinde O. Babaagba, editors, *Applications of Evolutionary Computation - 25th European Conference, EvoApplications 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20-22, 2022, Proceedings*, volume 13224 of *Lecture Notes in Computer Science*, pages 704–720. Springer, 2022.

[189] Petr Posik. Bbob-benchmarking the DIRECT global optimization algorithm. In Franz Rothlauf, editor, *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*, pages 2315–2320. ACM, 2009.

[190] Michael J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 01 1964.

[191] Michael J. D. Powell. *A direct search optimization method that models the objective and constraint functions by linear interpolation.* Springer, 1994.

[192] Raphael Patrick Prager and Heike Trautmann. Pflacco: Feature-based landscape analysis of continuous and constrained optimization problems in python. *Evolutionary Computation*, pages 1–25, 2023.

[193] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization.* Springer Science & Business Media, 2006.

[194] Ronald Pyke. Spacings. *Journal of the Royal Statistical Society: Series B (Methodological)*, 27(3):395–436, 1965.

[195] A. Kai Qin, Vicky Ling Huang, and Ponnuthurai N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[196] Charles P. Quesenberry and F.L. Miller Jr. Power studies of some tests for uniformity. *Journal of Statistical Computation and Simulation*, 5(3):169–191, 1977.

[197] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M. A. Salama. Opposition-based differential evolution algorithms. In *IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*, pages 2010–2017. IEEE, 2006.

[198] David Criado Ramón. Python advanced differential evolution (pyade). https://github.com/xKuZz/pyade, 2019. Accessed on 31/12/2021.

[199] Jérémy Rapin, Marcus Gallagher, Pascal Kerschke, Mike Preuss, and Olivier Teytaud. Exploring the MLDA benchmark on the nevergrad platform. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 1888–1896. ACM, 2019.

[200] Jeremy Rapin and Olivier Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[201] Ingo Rechenberg. *Evolutionsstrategie.* Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart, 1973.

[202] Quentin Renau, Carola Doerr, Johann Dréo, and Benjamin Doerr. Exploratory landscape analysis is strongly sensitive to the sampling strategy. In Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II*, volume 12270 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2020.

[203] Quentin Renau, Johann Dréo, Carola Doerr, and Benjamin Doerr. Expressiveness and robustness of landscape features. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 2048–2051. ACM, 2019.

[204] Quentin Renau, Johann Dréo, Carola Doerr, and Benjamin Doerr. Towards explainable exploratory landscape analysis: Extreme feature selection for classifying BBOB functions. In Pedro A. Castillo and Juan Luis Jiménez Laredo, editors, *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7-9, 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 17–33. Springer, 2021.

[205] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[206] André Luis Debiaso Rossi, André Carlos Ponce de Leon Ferreira de Carvalho, Carlos Soares, and Bruno Feres de Souza. Metastream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, 127:52–64, 2014.

[207] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, Springer, 2003.

[208] Maria Laura Santoni, Elena Raponi, Renato De Leone, and Carola Doerr. Comparison of high-dimensional bayesian optimization algorithms on BBOB. *CoRR*, abs/2303.00890, 2023.

[209] Lennart Schäpermeier, Christian Grimme, and Pascal Kerschke. To boldly show what no one has seen before: A dashboard for visualizing multi-objective landscapes. In Hisao Ishibuchi, Qingfu Zhang, Ran Cheng, Ke Li, Hui Li, Handing Wang, and Aimin Zhou, editors, *Evolutionary Multi-Criterion Optimization - 11th International Conference, EMO 2021, Shenzhen, China, March 28-31, 2021, Proceedings*, volume 12654 of *Lecture Notes in Computer Science*, pages 632–644. Springer, 2021.

[210] Dominik Schröder, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Chaining of numerical black-box algorithms: Warm-starting and switching points. *CoRR*, abs/2204.06539, 2022.

[211] Michael A. Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13:270–276, 1968.

[212] David F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[213] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[214] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[215] Lloyd S. Shapley. A value for n-person games. In H. Kuhn and A. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, 1953.

[216] Urban Škvorc, Tome Eftimov, and Peter Korošec. Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, 90:106138, 2020.

[217] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):1–25, 2009.

[218] Kenneth Sörensen. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18, 2015.

[219] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[220] Genaro Sucarrat. AutoSEARCH: General-to-Specific (GETS) Modelling. R package version 1.5.

[221] Ponnuthurai Nagaratnam Suganthan, Mostafa Ali, J. J. Liang, B. Y. Qu, Cai Tong Yue, and Kenneth Price. Competition on single objective bound constrained numerical optimization. https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark, 2020.

[222] T. Swartz. Goodness-of-fit tests using kullback-leibler information. *Communications in Statistics: Theory and Methods*, 21:711–729, 1992.

[223] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*, pages 71–78. IEEE, 2013.

[224] Ryoji Tanabe and Alex S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*, pages 1658–1665. IEEE, 2014.

[225] Dirk Thierens and Tobias van Driessel. A benchmark generator of tree decomposition mk landscapes. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 229–230. ACM, 2021.

[226] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012.

[227] Tea Tusar, Dimo Brockhoff, and Nikolaus Hansen. Mixed-integer benchmark problems for single- and bi-objective optimization. In Anne Auger and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 718–726. ACM, 2019.

[228] Tea Tusar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. COCO: the bi-objective black box optimization benchmarking (bbob-biobj) test suite. *CoRR*, abs/1604.00359, 2016.

[229] Tjeerd van Campen, Herbert Hamers, Bart Husslage, and Roy Lindelauf. A new approximation method for the shapley value applied to the WTC 9/11 terrorist attack. *Social Network Analysis and Mining*, 8(1):3:1–3:12, 2018.

[230] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[231] Jan N. van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. Having a blast: Meta-learning and heterogeneous ensembles for data streams. In Charu C. Aggarwal, Zhi-Hua Zhou, Alexander Tuzhilin, Hui Xiong, and Xindong Wu, editors, *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 1003–1008. IEEE Computer Society, 2015.

[232] Sander van Rijn, Carola Doerr, and Thomas Bäck. Towards an adaptive CMA-ES configurator. In Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley, editors, *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part I*, volume 11101 of *Lecture Notes in Computer Science*, pages 54–65. Springer, 2018.

[233] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. Evolving the structure of evolution strategies. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, pages 1–8. IEEE, 2016.

[234] Sander van Rijn, Hao Wang, Bas van Stein, and Thomas Bäck. Algorithm configuration data mining for CMA evolution strategies. In Peter A. N. Bosman,

editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, pages 737–744. ACM, 2017.

[235] Bas van Stein, Fabio Caraffini, and Anna V. Kononova. Emergence of structural bias in differential evolution. In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 1234–1242. ACM, 2021.

[236] Bas van Stein, Diederick Vermetten, Fabio Caraffini, and Anna V. Kononova. Deep BIAS: detecting structural bias using explainable AI. In Sara Silva and Luís Paquete, editors, *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*, pages 455–458. ACM, 2023.

[237] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. Openml: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

[238] Konstantinos Varelas, Ouassim Ait ElHara, Dimo Brockhoff, Nikolaus Hansen, Duc Manh Nguyen, Tea Tusar, and Anne Auger. Benchmarking large-scale continuous optimizers: The bbob-largescale testbed, a COCO software guide and beyond. *Applied Soft Computing*, 97(Part):106737, 2020.

[239] Oldrich Vasicek. A test for normality based on sample entropy. *Journal of the Royal Statistical Society. Series B (Methodological)*, 38(1):54–59, 1976.

[240] Diederick Vermetten, Fabio Caraffini, Anna V. Kononova, and Thomas Bäck. Modular differential evolution. In Sara Silva and Luís Paquete, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pages 864–872. ACM, 2023.

[241] Diederick Vermetten, Anna V. Kononova, Fabio Caraffini, Hao Wang, and Thomas Bäck. Is there anisotropy in structural bias? In Krzysztof Krawiec, editor, *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, pages 1243–1250. ACM, 2021.

[242] Diederick Vermetten, Manuel López-Ibáñez, Olaf Mersmann, Richard Allmendinger, and Anna V. Kononova. Analysis of modular CMA-ES on strict box-constrained problems in the SBOX-COST benchmarking suite. In Sara Silva and Luís Paquete, editors, *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*, pages 2346–2353. ACM, 2023.

[243] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. Online selection of CMA-ES variants. In Anne Auger and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 951–959. ACM, 2019.

[244] Diederick Vermetten, Bas van Stein, Fabio Caraffini, Leandro L. Minku, and Anna V. Kononova. BIAS: A toolbox for benchmarking structural bias in the continuous domain. *IEEE Transactions on Evolutionary Computation*, 26(6):1380–1393, 2022.

[245] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pages 654–662. ACM, 2020.

[246] Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Integrated vs. sequential approaches for selecting and tuning CMA-ES variants. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pages 903–912. ACM, 2020.

[247] Diederick Vermetten, Hao Wang, Manuel López-Ibáñez, Carola Doerr, and Thomas Bäck. Analyzing the impact of undersampling on the benchmarking and configuration of evolutionary algorithms. In Jonathan E. Fieldsend and Markus Wagner, editors, *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022*, pages 867–875. ACM, 2022.

[248] Diederick Vermetten, Hao Wang, Kevin Sim, and Emma Hart. To switch or not to switch: Predicting the benefit of switching between algorithms based on trajectory features. In João Correia, Stephen L. Smith, and Raneem Qaddoura, editors, *Applications of Evolutionary Computation - 26th European Conference, EvoApplications 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12-14, 2023, Proceedings*, volume 13989 of *Lecture Notes in Computer Science*, pages 335–350. Springer, 2023.

[249] Diederick Vermetten, Furong Ye, Thomas Bäck, and Carola Doerr. MA-BBOB: A problem generator for black-box optimization using affine combinations and shifts. *CoRR*, abs/2312.11083, 2023.

[250] Diederick Vermetten, Furong Ye, Thomas Bäck, and Carola Doerr. MA-BBOB: many-affine combinations of BBOB functions for evaluating automl approaches in noiseless numerical black-box optimization contexts. In Aleksandra Faust, Roman Garnett, Colin White, Frank Hutter, and Jacob R. Gardner, editors, *International Conference on Automated Machine Learning, 12-15 November 2023, Hasso Plattner Institute, Potsdam, Germany*, volume 224 of *Proceedings of Machine Learning Research*, pages 7/1–14. PMLR, 2023.

[251] Diederick Vermetten, Furong Ye, and Carola Doerr. Using affine combinations of BBOB problems for performance assessment. In Sara Silva and Luís Paquete, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pages 873–881. ACM, 2023.

[252] Costas Voglis, Grigoris S. Piperagkas, Konstantinos E. Parsopoulos, Dimitris G. Papageorgiou, and Isaac E. Lagaris. MEMPSODE: an empirical assessment of local search algorithm impact on a memetic algorithm using noiseless testbed. In Terence Soule and Jason H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*, pages 245–252. ACM, 2012.

[253] Urban Škvorc, Tome Eftimov, and Peter Korošec. The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis. In *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021*, pages 1139–1146. IEEE, 2021.

[254] Hao Wang, Michael Emmerich, and Thomas Bäck. Mirrored orthogonal sampling for covariance matrix adaptation evolution strategies. *Evolutionary Computation*, 27(4):699–725, 2019.

[255] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. IOHanalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization*, 2(1):3:1–3:29, 2022.

[256] Thomas Weise, Yan Chen, Xinlu Li, and Zhize Wu. Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing*, 92:106269, 2020.

[257] Thomas Weise, Stefan Niemczyk, Hendrik Skubch, Roland Reichle, and Kurt Geihs. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In Conor Ryan and Maarten Keijzer, editors, *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, GA, USA, July 12-16, 2008*, pages 795–802. ACM, 2008.

[258] Thomas Weise and Zijun Wu. Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In Hernán E. Aguirre and Keiki Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 1769–1776. ACM, 2018.

[259] L. Darrell Whitley, Francisco Chicano, and Brian W. Goldman. Gray box optimization for mk landscapes (NK landscapes and MAX-kSAT). *Evolutionary Computation*, 24(3):491–519, 2016.

[260] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*, pages 3381–3387. IEEE, 2008.

[261] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008.

[262] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT'12)*, volume 7317 of *LNCS*, pages 228–241. Springer, 2012.

[263] Danial Yazdani, Mohammad Nabi Omidvar, Delaram Yazdani, Kalyanmoy Deb, and Amir H. Gandomi. GNBG: A generalized and configurable benchmark generator for continuous numerical optimization. *CoRR*, abs/2312.07083, 2023.

[264] Furong Ye, Carola Doerr, Hao Wang, and Thomas Bäck. Automated configuration of genetic algorithms by tuning for anytime performance. *IEEE Transactions on Evolutionary Computation*, 26(6):1526–1538, 2022.

[265] Jin Zhang. Powerful goodness-of-fit tests based on the likelihood ratio. *Journal of the Royal Statistical Society*, Series B, 64:281–294, 2002.

[266] Jingqiao Zhang and Arthur C. Sanderson. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.

Acronyms

AOCC Area Over the Convergence Curve
AOC Area Over the (ECDF) Curve
AUC Area Under the (ECDF) Curve
BBOB Black-Box Optimization Benchmarking
CMA-ES Covariance-Matrix Adaptation Evolution Strategy
COCO COmparing Continuous Optimizers
DE Differential Evolution
DynAS Dynamic Algorithm Selection
ECDF Empirical Cumulative Distribution Function
ERT Expected Running time
ELA Exploratory Landscape Analysis
IOH Iterative Optimization Heuristic
KS Kolmogovor-Smirnov
MA-BBOB Many-Affine BBOB
MIP-EGO Mixed-Integer Parallell Efficient Global Optimization
PAR Penalized Average Running time
PCA Principal Component Analysis
SB Structural Bias
SMAC Sequential Model-based Algorithm Configuration
TSNE T-distributed Stochastic Neighbor Embedding
UMAP Uniform Manifold Approximation Projection
VBS Virtual Best Solver

Samenvatting

In de context van optimalisatie kan benchmarking worden gezien als een verbinding tussen het theoretisch analyseren van algoritmes en de praktijk. De empirische aard van benchmarking zorgt ervoor dat experimenten op grote schaal kunnen worden uitgevoerd om vragen te beantwoorden die te complex zijn voor de huidige theoretische methoden. Tegelijkertijd hoeft benchmarking niet in een specifiek praktisch gebied te passen, maar kan het een breder scala aan fundamentele vragen bestuderen. Deze combinatie zorgt dat benchmark-studies een grote variëteit aan nieuwe inzichten over de relatieve sterktes en zwaktes van optimisatie algoritmes kunnen opleveren.

Gezien de potentiële voordelen van robuuste benchmarking-opstellingen is het belangrijk dat er laagdrempelige tools beschikbaar worden gemaakt. Deze opstellingen moeten flexibel genoeg zijn om tegemoet te komen aan de grote verscheidenheid aan vragen die worden gesteld, terwijl ze tegelijkertijd rigoureuze kaders bieden voor de rest van de benchmarking-pijplijn. In deze thesis focussen we op IOHprofiler, een benchmarking pakket ontwikkeld als hulpmiddel voor de bredere onderzoeksgemeenschap. Met behulp van een modulaire structuur kan IOHprofiler worden geïntegreerd met veel bestaande tools die op grote schaal worden gebruikt. Deze integraties vormen de ruggengraat voor de experimenten die in deze thesis worden gebruikt, omdat de bijbehorende data, zowel over de prestatie van de algoritmes als over hun gedrag, eenvoudig kan worden hergebruikt. Op deze manier kunnen we verder gaan dan de gebruikelijke 'competitieve' instelling, waarbij alleen gelet word op het algoritme met de beste gemiddelde prestaties, en in plaats daarvan inzicht krijgen in de complementariteit tussen verschillende algoritmes.

Een van de belangrijkste manieren waarop complementariteit tussen algoritmes nuttig kan zijn, is in de context van algoritmeselectie en -configuratie. In plaats van te vertrouwen op één enkel algoritme voor een breed scala aan problemen, gebruiken we een reeks test-functies om te bepalen welk algoritme of welke algoritmeconfiguratie

we moeten gebruiken op nieuwe problemen. Deze meta-leertaak profiteert van de variatie tussen algoritmen, die kan worden verkregen door naar verschillende soorten algoritmen te kijken, maar ook binnen een algoritmefamilie door rekening te houden met een grote hoeveelheid modificaties die de afgelopen decennia zijn voorgesteld. We laten zien hoe twee modularisaties van populaire evolutionaire algoritmen kunnen leiden tot nieuwe inzichten over combinaties van algoritmische ideeën, resulterend in verbeterde prestaties ten opzichte van eerdere, met de hand ontworpen versies van dezelfde algoritmen.

Hoewel er op probleemniveau al sprake is van complementariteit van algoritmen, blijkt uit de benchmarkgegevens van verschillende bronnen ook dat er ook sprake is van complementariteit in de prestaties binnen individuele functies. Hoewel sommige algoritmen uitstekend zijn in het vinden van veelbelovende regio's, blinken andere uit in snelle convergentie zodra deze regio is gevonden. Als zodanig wordt het begrip dynamische algoritmeselectie, waarbij we tijdens de optimalisatieprocedure tussen verschillende algoritmen kunnen wissellen, in detail bestudeerd. We benadrukken het inherente potentieel van deze aanpak, terwijl we tegelijkertijd de aspecten van deze wisseling benadrukken die verder ontwikkeld moeten worden om betrouwbare dynamische algoritme-combinaties te creëren.

Dynamische algoritmeselectie is een veelbelovend onderzoeksgebied binnen optimalisatie, maar studies naar de prestaties ervan zijn nog enigszins beperkt. Dit is grotendeels een gevolg van de moeilijkheid om betrouwbare benchmarking-opstellingen voor dit scenario te creëren. Hoewel verschillende suites voor benchmarking-optimalisatie-algoritmen beschikbaar zijn, zijn deze niet bijzonder geschikt voor meta-leerscenario's, omdat er geen echt eerlijke manier is om het algemeen vereiste onderscheid tussen train- en testproblemen te creëren dat wordt gebruikt door de vereiste machine learning-technieken. Om deze reden stellen we een benchmark-probleemgenerator voor, gebaseerd op veelgebruikte black-box-optimalisatieproblemen, die kan worden gebruikt om willekeurige hoeveelheden train- en testproblemen te genereren om deze meta-leermechanismen te benchmarken.

Summary

Within the context of optimization, benchmarking is sometimes viewed as a bridge between theory and practice. Its empirical nature allows for the creation of large-scale experiments to investigate a wide variety of questions, which can be intractable using theoretical approaches. At the same time, benchmarking does not have to fit neatly into a specific application domain but can cover a broader range of fundamental questions. This combination allows benchmarking studies to provide a range of new insights into the strengths and weaknesses of different optimization heuristics.

Given the potential benefits of robust benchmarking setups, it is critical that tools are created which lower the barrier to entry for its use. These setups should be flexible enough to fit the wide variety of questions being asked while providing rigorous frameworks for the rest of the benchmarking pipeline. In this thesis, we focused on the IOHprofiler framework, which aims to be such a tool for the wider research community. Using a modular design structure, IOHprofiler can be integrated with many commonly used tools for black-box optimization and benchmarking. These integrations form the backbone for the experiments used throughout this thesis, as the corresponding performance and algorithm behavior data can be easily reused. This way, we can go beyond the common 'competitive' benchmarking practice, where we only care about the algorithm with the best average performance, to gaining insights about the complementarity between different algorithms.

One of the key ways in which algorithm complementarity can be beneficial is in the context of algorithm selection and configuration. Instead of relying on a single algorithm for a wide set of problems, we use a set of features to determine which algorithm or algorithm configuration to use. This meta-learning task benefits from variety between algorithms, which can be achieved by looking at different types of algorithms, but also within an algorithm family by considering a wide range of modifications proposed over the last decades. We show how two modularizations of popular

evolutionary algorithms, CMA-ES and Differential Evolution, can lead to new insights about combinations of algorithmic ideas, resulting in improved performance over previous hand-designed versions of these same algorithms.

While algorithm complementarity exists on the problem-level, looking at benchmark data from a variety of sources also reveals a complementarity in performance within individual functions. While some algorithms are great at finding promising regions, others excel at fast convergence once this region is found. As such, the notion of dynamic algorithm selection, where we can switch between different algorithms during the optimization procedure, is studied in detail. We highlight the inherent potential in this approach, while simultaneously highlighting the aspects of this switching approach which need to be further developed to create reliably dynamic algorithm combinations.

Dynamic algorithm selection is a promising field of research within optimization, but current studies into its performance are still somewhat limited. This is largely a result of the difficulty of creating reliable benchmarking setups for this scenario. While collections of problems for benchmarking optimization algorithms are widely available, they are not particularly suited for meta-learning scenarios, as there is no truly fair way to create the commonly required train-test set distinction used by the required machine learning techniques. For this reason, we propose a benchmark problem generator based on commonly used black-box optimization problems, which can be used to generate arbitrary amounts of training and testing problems to benchmark these meta-learning mechanisms.

Acknowledgements

Doing a PhD is often viewed as a somewhat lonely experience, but I have been fortunate to find this absolutely untrue in my journey. Throughout the years I spent on this trajectory, I have been able to count on the friendship and support of many amazing people, who I would like to thank here explicitly.

First, I would like to thank my supervisors: Thomas Bäck, Hao Wang and Carola Doerr. Thomas, for fostering a wonderful working environment, and providing me with many incredible opportunities. Hao, for the great discussions and for providing the guidance I needed, especially in the early stages. And Carola, for always being available for discussions even while being located in different countries. Even through online calls, I always feel supported and motivated whenever we interacted.

Even when I was a master's student, I always felt welcomed in the natural computing group, so I want to thank Sander van Rijn for introducing me to the group and the topic which started as a short master's project but eventually turned into this PhD thesis.

Within the NaCo group, I have been lucky to meet many people who over the years became good friends. Especially Jacob de Nobel and Furong Ye, whom I was lucky enough to share offices with, in the various buildings we were put in over the years. The many collaborations we have together, with IOHprofiler as the center point, were the source of a lot of fun.

The NaCo group is very diverse and dynamic, and I want to thank everyone involved who made it into the great environment I experienced it to be. Niki, Kirill, Qi, Elena, Maarten, Roy, Sylvia, Shuaiqun, Roman, Jingjie, Hugo, Thodoris, Marios, Charles, Anh, Ivan, Haoran and everyone I forgot here. It was a pleasure to work with you, both scientifically and beyond.

As a PhD, I was lucky to be able to teach alongside Anna Kononova. Sharing the teaching duties of our courses helped me grow into the role, and during the many

discussions we had about teaching I was introduced to the ideas of structural bias, which proved to be the start of a nice research collaboration as well.

If there is one thing I learned while doing this PhD, it is that research is a collaborative process. Many of the projects which I worked on would not have been possible without the many collaborations with wonderful people from all around the world. Anja, Francois and Quentin in Paris; Tome, Ana, Ana and Gjorgjina in Ljubliana; Fu Xing and Andre in Munich; Kevin and Emma in Edinburgh; Jeroen, Oliver and Heike in Paderborn; Kate and Andres in Melbourne; Konstantin and Pascal in Dresden; Fabio in Swansea; Manuel in Manchester, and many more. Collaborating is the most fun part of research, and working with all of you has been a great experience which I hope will continue into the future.

Lastly, I would like to thank the people outside academia who supported me on this journey. First, my teammates at Docos and TTC Molenbeersel, playing table tennis with you has been a large aspect in keeping me sane. Finally, I want to thank my family, in particular my parents and sisters, for always supporting me. I can not imagine a more supportive (yet chaotic) group of people to share this journey with.

Curriculum Vitae

Diederick Vermetten was born on August 23rd, 1996 in Maastricht, the Netherlands. He completed high school at Maaseik, Belgium in 2014 and then went on to study at Leiden University. There, he completed Bachelor's degrees in both Computer Science and Mathematics in 2017. He went on to complete the Master's degree in Computer Science, with distinction, in 2019. His Master's thesis introduced him to IOHprofiler, and he was subsequently hired as a part-time scientific developer on this project, as well as as a teaching assistant, before starting his PhD in January 2020. As a PhD, Diederick was involved in teaching the courses Natural Computing (Bsc) and Introduction to Machine Learning (Msc). In 2022, he was awarded a SPECIES scholarship to visit Dr. Kevin Sim and Prof. Emma Hart at Edinburgh Napier University, Schotland.

Publications

Journal Publications

1. Diederick Vermetten, Furong Ye, Thomas Bäck, Carola Doerr. MA-BBOB: A Problem Generator for Black-Box Optimization Using Affine Combinations and Shifts. *Transactions on Evolutionary Learning and Optimization.* ACM, 2024

2. Ana Kostovska, Diederick Vermetten, Peter Korošec, Sašo Džeroski, Carola Doerr, Tome Eftimov. Using Machine Learning Methods to Assess Module Performance Contribution in Modular Optimization Frameworks. *Evolutionary Computation Journal.* MIT Press, 2024.

3. Anna V. Kononova, Diederick Vermetten, Fabio Caraffini, Madalina-A. Mitran, Daniela Zaharie. The Importance of Being Constrained: Dealing with Infeasible Solutions in Differential Evolution and Beyond *Evoluationary Computation Journal.* MIT Press, 2023.

4. Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Thomas Bäck. IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. *Evolutionary Computation Journal.* MIT Press, 2023.

5. Xavier Bonet-Monroig, Hao Wang, Diederick Vermetten, Bruno Senjean, Charles Moussa, Thomas Bäck, Vedran Dunjko, and Thomas E. O'Brien. Performance comparison of optimization methods on variational quantum algorithms. *Physical Review A*. 107, 032407 2023.

6. Thomas H. W. Bäck, Anna V. Kononova, Bas van Stein, Hao Wang, Kirill A. Antonov, Roman T. Kalkreuth, Jacob de Nobel, Diederick Vermetten, Roy de Winter, Furong Ye. Evolutionary Algorithms for Parameter Optimization—Thirty Years Later. *Evolutionary Computation Journal.* MIT Press, 2023.

7. Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, Thomas Bäck. IOHanalyzer: Detailed Performance Analyses for Iterative Optimization Heuristics. *ACM Transactions on Evolutionary Learning and Optimization*. 2(1): 3:1-3:29. 2022.

8. Diederick Vermetten, Bas van Stein, Fabio Caraffini, Leandro L. Minku, Anna V. Kononova. BIAS: A Toolbox for Benchmarking Structural Bias in the Continuous Domain. *IEEE Transactions on Evolutionary Computation*. 26(6): 1380-1393. 2022

9. Ana Kostovska, Diederick Vermetten, Carola Doerr, Sašo Džeroski, Panče Panov, and Tome Eftimov. "OPTION: OPTImization Algorithm Benchmarking ONtology." *IEEE Transactions on Evolutionary Computation*. 2022.

Peer-reviewed Conference Publications: Main Tracks

1. Diederick Vermetten, Johannes Lengler, Dimitri Rusin, Thomas Bäck, Carola Doerr. Empirical Analysis of the Dynamic Binary Value Problem with IOHprofiler. *In Proc. of International Conference on Parallel Problem Solving from Nature (PPSN'24)*, 20-35. Springer, 2024.

2. Jacob de Nobel, Diederick Vermetten, Anna V. Kononova, Ofer M. Shir, and Thomas Bäck. Avoiding Redundant Restarts in Multimodal Global Optimization. *In Proc. of International Conference on Parallel Problem Solving from Nature (PPSN'24)*, 268-283. Springer, 2024.

3. Diederick Vermetten, Carola Doerr, Hao Wang, Anna V. Kononova, Thomas Bäck. Large-Scale Benchmarking of Metaphor-Based Optimization Heuristics. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'24)*, 41–49. ACM, 2024.

4. Konstantin Dietrich, Diederick Vermetten, Carola Doerr, Pascal Kerschke. Impact of Training Instance Selection on Automated Algorithm Selection Models for Numerical Black-box Optimization. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'24)*, 1007–1016. ACM, 2024.

5. Shuaiqun Pan, Diederick Vermetten, Thomas Bäck, Manuel López-Ibáñez, Hao Wang. Transfer Learning of Surrogate Models via Domain Affine Trans-

formation. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'24)*, 385–393. ACM, 2024.

6. Ana Nikolikj, Ana Kostovska, Diederick Vermetten, Carola Doerr, Tome Eftimov. Quantifying Individual and Joint Module Impact in Modular Optimization Frameworks. *In Proc. of IEEE Congress on Evolutionary Computation (CEC'24)*. IEEE, 2024.

7. Maarten C. Vonk, Diederick Vermetten, Jacob de Nobel, Sebastiaan Brand, Ninoslav Malekovic, Thomas Bäck, Alfons Laarman, and Anna V. Kononova. Optimizing Causal Interventions in Hybrid Bayesian Networks: A discretization, knowledge compilation, and heuristic optimization approach. *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU' 24)*

8. Haoran Yin, Diederick Vermetten, Furong Ye, Thomas Bäck, Anna V. Kononova: Impact of Spatial Transformations on Exploratory and Deep-learning Based Landscape Features of CEC2022 Benchmark Suite. *International Conference on Evolutionary Computation Theory and Applications (ECTA'24)*, 2024.

9. Jacob de Nobel, Diederick Vermetten, Thomas Bäck, Anna V. Kononova: Sampling in CMA-ES: Low Numbers of Low Discrepancy Points. *International Conference on Evolutionary Computation Theory and Applications (ECTA'24)*, 2024. (**Best paper award**)

10. François Clément, Diederick Vermetten, Jacob de Nobel, Alexandre D. Jesus, Luís Paquete, Carola Doerr. Computing Star Discrepancies with Numerical Black-Box Optimization Algorithms. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23)*, 1330-1338. ACM, 2023.

11. Diederick Vermetten, Furong Ye, Carola Doerr. Using Affine Combinations of BBOB Problems for Performance Assessment. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23)*, 873-881. ACM, 2023. (**Best paper award in ENUM track**)

12. Diederick Vermetten, Fabio Caraffini, Anna V. Kononova, Thomas Bäck. Modular Differential Evolution. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23)*, 864-872. ACM, 2023.

13. André Thomaser, Jacob de Nobel, Diederick Vermetten, Furong Ye, Thomas Bäck, Anna V. Kononova. When to be Discrete: Analyzing Algorithm Performance on Discretized Continuous Problems. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23)*, 856-863. ACM, 2023.

14. Roman Kalkreuth, Zdenek Vasícek, Jakub Husa, Diederick Vermetten, Furong Ye, Thomas Bäck. General Boolean Function Benchmark Suite. *In Proc. of Foundations of Genetic Algorithms (FOGA'23)*, 84-95. ACM, 2023.

15. Fu Xing Long, Diederick Vermetten, Bas van Stein, Anna V. Kononova. BBOB Instance Analysis: Landscape Properties and Algorithm Performance Across Problem Instances. *International Conference on the Applications of Evolutionary Computation (EVOSTAR'23)*, 380-395. Springer, 2023. (**Fu Xing Long and Diederick Vermetten received outstanding student awards for this paper.**)

16. Diederick Vermetten, Hao Wang, Kevin Sim, Emma Hart. To Switch or Not to Switch: Predicting the Benefit of Switching Between Algorithms Based on Trajectory Features. *International Conference on the Applications of Evolutionary Computation (EVOSTAR'23)*, 335-350. Springer, 2023.

17. Ana Kostovska, Diederick Vermetten, Saso Dzeroski, Pance Panov, Tome Eftimov, Carola Doerr. Using Knowledge Graphs for Performance Prediction of Modular Optimization Algorithms. *International Conference on the Applications of Evolutionary Computation (EVOSTAR'23)*, 253-268. Springer, 2023.

18. Frank Neumann, Aneta Neumann, Chao Qian, Anh Viet Do, Jacob de Nobel, Diederick Vermetten, Saba Sadeghi Ahouei, Furong Ye, Hao Wang, Thomas Bäck. Benchmarking Algorithms for Submodular Optimization Problems Using IOHProfiler. *In Proc. of IEEE Congress on Evolutionary Computation (CEC'23)*. IEEE, 2023.

19. Kostovska, Ana, Gjorgjina Cenikj, Diederick Vermetten, Anja Jankovic, Ana Nikolikj, Urban Skvorc, Peter Korosec, Carola Doerr, and Tome Eftimov: PS-AAS: Portfolio Selection for Automated Algorithm Selection in Black-Box Optimization. *The International Conference on Automated Machine Learning (AutoML'23)*, 2023.

20. Diederick Vermetten, Furong Ye, Thomas Bäck, and Carola Doerr: MA-BBOB: Many-Affine Combinations of BBOB Functions for Evaluating AutoML Ap-

proaches in Noiseless Numerical Black-Box Optimization Contexts. *The International Conference on Automated Machine Learning (AutoML'23)*, 2023.

21. Fu Xing Long, Diederick Vermetten, Anna V. Kononova, Roman Kalkreuth, Kaifeng Yang, Thomas Bäck and Niki van Stein: Challenges of ELA-guided Function Evolution using Genetic Programming. *International Conference on Evolutionary Computation Theory and Applications (ECTA'23)*, 2023.

22. Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, Carola Doerr: Per-run Algorithm Selection with Warm-Starting Using Trajectory-Based Features. *In Proc. of International Conference on Parallel Problem Solving from Nature (PPSN'22)*, 46-60. Springer, 2022.

23. Furong Ye, Diederick Vermetten, Carola Doerr, Thomas Bäck. Non-elitist Selection Can Improve the Performance of Irace. *In Proc. of International Conference on Parallel Problem Solving from Nature (PPSN'22)*, 32-45. Springer, 2022.

24. Diederick Vermetten, Hao Wang, Manuel López-Ibáñez, Carola Doerr, Thomas Bäck. Analyzing the impact of undersampling on the benchmarking and configuration of evolutionary algorithms. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'22)*. 867-875, ACM, 2022.

25. Ana Kostovska, Diederick Vermetten, Saso Dzeroski, Carola Doerr, Peter Korosec, Tome Eftimov. The importance of landscape features for performance prediction of modular CMA-ES variants. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'22)*. 648-656, ACM, 2022.

26. Anja Jankovic, Diederick Vermetten, Ana Kostovska, Jacob de Nobel, Tome Eftimov, Carola Doerr. Trajectory-based Algorithm Selection with Warm-starting. *In Proc. of IEEE Congress on Evolutionary Computation (CEC'22)*. 1-8, IEEE, 2022.

27. Diederick Vermetten, Hao Wang, Carola Doerr, Thomas Bäck. Integrated vs. sequential approaches for selecting and tuning CMA-ES variants. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*, 903-912. AMC, 2020.

28. Diederick Vermetten, Hao Wang, Thomas Bäck, Carola Doerr. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*. 654-662, ACM, 2020.

29. Diederick Vermetten, Sander van Rijn, Thomas Bäck, Carola Doerr. Online selection of CMA-ES variants. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*, 951-959. ACM, 2019.

Peer-reviewed Conference Publications: Workshop or Companion Track

1. Martijn Halsema, Diederick Vermetten, Thomas Bäck, Niki van Stein. A Critical Analysis of Raven Roost Optimization. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'24), Companion Volume*, 1993–2001. ACM, 2024.

2. Diederick Vermetten, Manuel López-Ibáñez, Olaf Mersmann, Richard Allmendinger, Anna V. Kononova. Analysis of modular CMA-ES on strict box-constrained problems in the SBOX-COST benchmarking suite. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23), Companion Volume*, 2346-2353. ACM, 2023.

3. Roman Kalkreuth, Zdenek Vasícek, Jakub Husa, Diederick Vermetten, Furong Ye, Thomas Bäck. Towards a General Boolean Function Benchmark Suite. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23), Companion Volume*, 591-594. ACM, 2023.

4. Ana Kostovska, Anja Jankovic, Diederick Vermetten, Saso Dzeroski, Tome Eftimov, Carola Doerr. Comparing Algorithm Selection Approaches on Black-Box Optimization Problems. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23), Companion Volume*, 495-498. ACM, 2023.

5. Bas van Stein, Diederick Vermetten, Fabio Caraffini, Anna V. Kononova. Deep BIAS: Detecting Structural Bias using Explainable AI. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23), Companion Volume*, 455-458. ACM, 2023.

6. Ana Nikolikj, Gjorgjina Cenikj, Gordana Ispirova, Diederick Vermetten, Ryan Dieter Lang, Andries Petrus Engelbrecht, Carola Doerr, Peter Korosec, Tome Eftimov. Assessing the Generalizability of a Performance Predictive Model. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'23), Companion Volume*, 311-314. ACM, 2023.

7. Jacob de Nobel, Diederick Vermetten, Hao Wang, Carola Doerr, Thomas Bäck. Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'21), Companion Volume*, 1375-1384. ACM, 2021.

8. Diederick Vermetten, Fabio Caraffini, Bas van Stein, Anna V. Kononova: Using structural bias to analyse the behaviour of modular CMA-ES. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'22), Companion Volume*, 1674-1682. ACM, 2022.

9. Diederick Vermetten, Anna V. Kononova, Fabio Caraffini, Hao Wang, Thomas Bäck: Is there anisotropy in structural bias? *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'21), Companion Volume*, 1243-1250. ACM, 2021.

10. Ana Kostovska, Diederick Vermetten, Carola Doerr, Saso Dzeroski, Pance Panov, Tome Eftimov. OPTION: optimization algorithm benchmarking ontology. *In Proc. of Genetic and Evolutionary Computation Conference (GECCO'21), Companion Volume*, 239-240. ACM, 2021.

Book Chapters

1. Diederick Vermetten, Bas van Stein, Anna V. Kononova, Fabio Caraffini. Analysis of Structural Bias in Differential Evolution Configurations. *Differential Evolution: From Theory to Practice*. 1-22, Springer, 2022.

To Appear

1. Manuel López-Ibáñez, Diederick Vermetten, Johann Dreo, Carola Doerr. Using the Empirical Attainment Function for Analyzing Single-objective Blackbox Optimization Algorithms. *Accepted at IEEE Transactions on Evolutionary Computation*

2. Diederick Vermetten, Jeroen Rook, Oliver L. Preuß, Jacob de Nobel, Carola Doerr, Manuel López-Ibañez, Heike Trautmann, Thomas Bäck. MO-IOHinspector: Anytime Benchmarking of Multi-Objective Algorithms using IOHprofiler. *Accepted at International Conference on Evolutionary Multi-criterion Optimization.*

3. Anna V. Kononova, Diederick Vermetten, Niki van Stein. XAI for benchmarking black-box metaheuristics. *Chapter to appear in: Explainable AI for Evolutionary Computation And Vice Versa*

Under Review

1. Niki van Stein, Diederick Vermetten, Anna V. Kononova, Thomas Bäck. Explainable Benchmarking for Iterative Optimization Heuristics.

2. Niki van Stein, Diederick Vermetten, Thomas Bäck. In-the-loop Hyper-Parameter Optimization for LLM-Based Automated Design of Heuristics.

3. Sarah L. Thomson, Quentin Renau, Diederick Vermetten, Emma Hart, Niki van Stein, Anna V. Kononova. Stalling in Space: Attractor Analysis for any Algorithm.