SynthACticBench: A Capability-Based Synthetic Benchmark for Algorithm Configuration

Valentin Margraf
MCML, LMU Munich
Munich, Germany
valentin.margraf@ifi.lmu.de

Anna Lappe
LMU Munich
Munich, Germany

Marcel Wever
L3S Research Center
Leibniz University Hannover
Hannover, Germany

Carolin Benjamins
Leibniz University Hannover
Hannover, Germany

Eyke Hüllermeier
LMU Munich, MCML, DFKI (DSA)
Munich, Germany

Marius Lindauer
L3S Research Center
Leibniz University Hannover
Hannover, Germany

Abstract

Algorithm configuration deals with the automatic optimization of an algorithm's parameters to maximize its performance on a distribution of problem instances, such as Boolean satisfiability or the traveling salesperson problem. While significant progress has been made in developing optimizers for algorithm configuration – so-called algorithm configurators – their evaluation remains computationally expensive and often relies on real-world scenarios with hard-to-control characteristics. This makes it challenging to analyze their strengths and weaknesses systematically. To address this, we introduce `SynthACticBench`, a synthetic benchmark specifically designed to isolate and investigate key properties of algorithm configuration problems. Our benchmark distinguishes between properties related to the configuration space and those associated with the objective function. We define a configurator's ability to handle a particular property as its *capability* – for example, the capability to manage hierarchical configuration spaces. Using `SynthACticBench`, we evaluate two state-of-the-art algorithm configurators, SMAC and irace, examining their complementary capabilities and analyzing their performances across diverse benchmark functions. By providing a controlled, scalable, and capability-based evaluation environment, `SynthACticBench` facilitates a more targeted analysis of algorithm configurators, helping to advance research in the field. The benchmark is available at: https://github.com/annaelisalappe/SynthACticBench/.

CCS Concepts

• **Mathematics of computing** → **Evolutionary algorithms**; • **General and reference** → **Evaluation**.

Keywords

Algorithm Configuration, Benchmark, Capability, Synthetic Data, Black-Box Optimization

GECCO '25, July 14–18, 2025, Málaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726330

ACM Reference Format:
Valentin Margraf, Anna Lappe, Marcel Wever, Carolin Benjamins, Eyke Hüllermeier, and Marius Lindauer. 2025. SynthACticBench: A Capability-Based Synthetic Benchmark for Algorithm Configuration. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726330

1 Introduction

Automated algorithm configuration (AC) is a critical component in the design and optimization of algorithms. It involves systematically optimizing an algorithm's parameters to maximize performance on problem instances that follow a specific distribution [36]. In many applications, ranging from combinatorial optimization to machine learning, algorithm efficiency and effectiveness are highly sensitive to parameter configurations [3, 7, 11, 23, 29], referred to as hyperparameters in the machine learning community.

Automated algorithm *configurators* have been developed to tackle this challenge, allowing practitioners to efficiently identify high-performing configurations without manual intervention [2, 18, 20, 21, 27, 28, 32]. However, evaluating and improving these configurators remains challenging due to the complexity and diversity of real-world configuration problems used for validation.

Existing benchmarks for algorithm configurators often rely on real-world use cases or adapted algorithm design problems [10, 23, 24, 38]. While these benchmarks provide practical insights and highlight the importance of the problem; they are limited in that the underlying characteristics of the configuration problems are difficult to control. This lack of control hinders the precise analysis of configurator behavior and makes it challenging to attribute performance to specific capabilities of an algorithm configurator. Consequently, identifying strengths, weaknesses, and areas for improvement becomes difficult. For example, configurators may perform well on certain benchmarks due to favorable characteristics, such as few parameters or benign search landscapes [32], but struggle in scenarios with high-dimensional spaces, noisy performance functions, or a large proportion of irrelevant parameters.

To address these limitations, we propose the synthetic benchmark `SynthACticBench`. This evaluation framework is inspired by Hernández-Orallo [16], who proposes to evaluate a system based on its *cabilities*. In our context, a *capability* refers to the extent to which an algorithm configurator can handle a specific challenge

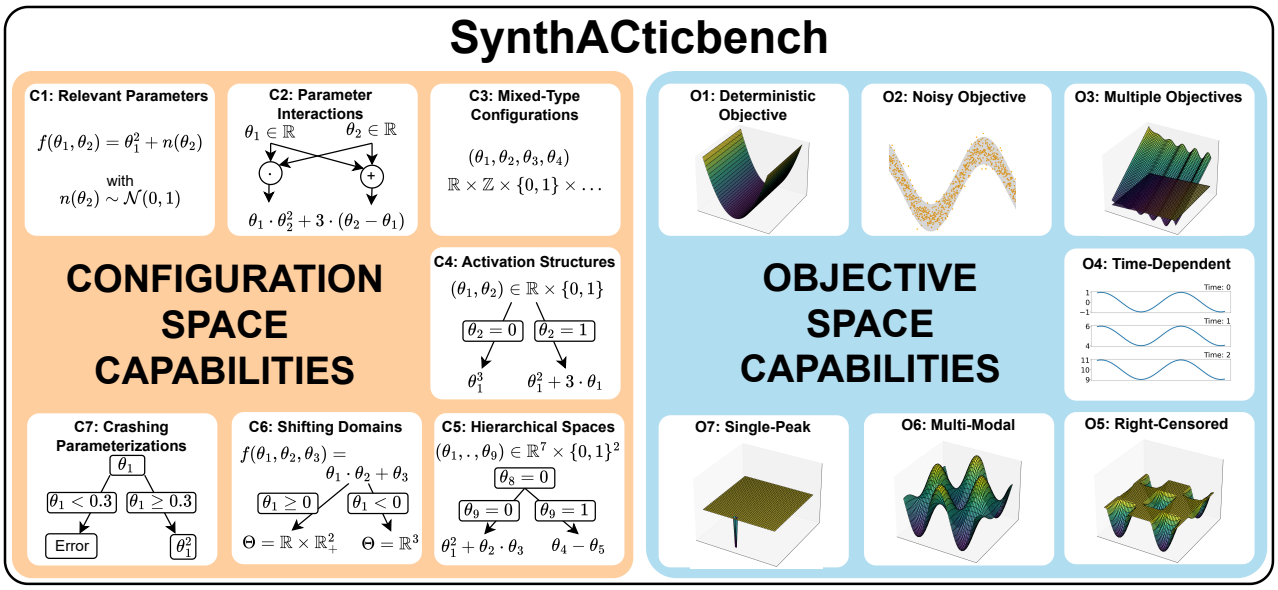

Figure 1: SynthACticBench assesses the capabilities of algorithm configurators – their abilities to handle specific challenges related to the configuration space and properties of the objective function in algorithm configuration problems.

related to an AC problem effectively. By structuring our benchmark around distinct capabilities, we can systematically isolate and analyze key challenges commonly encountered in AC. Specifically, we define and investigate 14 distinct capabilities, evaluating configurators on challenges related to the configuration space, e.g., managing mixed parameter types, and challenges arising from the objective function, e.g., dealing with censoring due to timeouts.

Contributions. We propose SynthACticBench to evaluate algorithm configurators. All in all, our contributions are threefold.

1. We define a comprehensive set of capabilities relevant to algorithm configuration tasks and introduce the synthetic benchmark SynthACticBench, designed for a targeted evaluation of algorithm configurators.
2. We conduct an exemplary evaluation of two state-of-the-art algorithm configurators, Sequential Model-based Algorithm Configuration (SMAC) [18] and Iterated Racing (irace) [28], and random search as a common baseline on SynthACticBench, providing insights into their performance across diverse problem landscapes.
3. We implement SynthACticBench in a controlled and scalable environment, advancing future research in AC.

2 Algorithm Configuration

In many computationally complex optimization problems or constraint satisfaction tasks, an algorithm's performance is highly sensitive to the choice of its parameter configurations. These parameters, which may be of different types, such as categorical, continuous, or integer, span the algorithm's *configuration space* $\mathbf{\Theta}$. The effective configuration of these parameters is crucial to achieving optimal performance on a given class of problem instances. The process of automatically identifying high-performing parameter configurations is referred to as algorithm configuration (AC) [36].

Formally, the AC problem for a target algorithm A can be defined by a 4-tuple $(\mathbf{\Theta}, \mathcal{I}, P, c)$ where $\mathbf{\Theta}$ denotes the configuration space, $\mathcal{I}$ the instance space over which a probability distribution P is defined and $c : \mathbf{\Theta} \times \mathcal{I} \to \mathbb{R}$ a cost metric, evaluating the performance of a parameter configuration $\theta \in \mathbf{\Theta}$ on an instance $i \in \mathcal{I}$. The configuration space $\mathbf{\Theta}$ comprises all (feasible) combinations of parameter configurations, and $\mathcal{I}$ represents the instance space, which encompasses the set of problem instances for which the algorithm must be configured. The cost metric c is typically related to the quality of the solution returned by A, the runtime needed for obtaining a result, or multiple of such criteria simultaneously. In the latter setting, the cost metric c maps to $\mathbb{R}^n$ where n is the number of objectives considered simultaneously [6]. For many real-world algorithms, $c(\theta, i)$ may exhibit stochastic behavior due to internal randomness or environmental noise, necessitating the use of expected values for evaluation.

The AC problem can thus be formulated as the following optimization problem:

$$\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \mathbb{E}_{i \sim P} \left[c(\theta, i) \right] .$$

In practical settings, the distribution P is typically approximated by a finite i.i.d.-sampled set of training instances $I_{\text{train}} \subset \mathcal{I}$, which reduces the above optimization problem to:

$$\hat{\theta} \in \arg\min_{\theta \in \mathbf{\Theta}} \frac{1}{|I_{\text{train}}|} \sum_{i \in I_{\text{train}}} c(\theta, i) .$$

Homogeneity Assumption. A fundamental assumption underlying the AC problem is that the configuration space $\mathbf{\Theta}$ contains a

solution that generalizes well across the instance space $\mathcal{I}$. To this end, we assume the problem instances to be *homogeneous*, meaning that a configuration performs similarly across different problem instances [37]. For some settings in the AC literature, such as per-instance algorithm configuration [25, 43], this assumption is relaxed to a set of problem instances comprising homogeneous subsets. However, in this work, we stick to the homogeneity assumption.

Generally speaking, we can distinguish between two levels of homogeneity: weak homogeneity and strong homogeneity. While the former only requires the best configuration to perform consistently the best across problem instances, the latter requires the entire ranking of configurations to be consistent. In other words, for any two configurations θ_j, θ_k, w.l.o.g. $c(\theta_j, i) < c(\theta_k, i)$ holds for all $i \in \mathcal{I}$ if strong homogeneity is assumed. In `SynthACticBench`, we create benchmarks where strong homogeneity is fulfilled.

3 Related Work

Various *synthetic* benchmarks have been proposed in the literature to evaluate and compare optimization algorithms. These benchmarks are crucial for testing algorithms in a controlled, interpretable, and scalable environment. For instance, the Black-Box Optimization Benchmarking Suite (BBOB) [14] provides a set of general synthetic benchmarking functions. The original version contains 24 functions, varying in the number of conditioning variables, modality, and global structure. It has since been extended to the Comparing Continuous Optimizers Benchmarking Suite (COCO) [15], which additionally includes noisy functions, multi-objective tasks, and discrete parameters. Further, the IOHprofiler [9] offers a well-defined benchmarking framework, with a problem set of 25 pseudo-Boolean functions selected to cover diverse characteristics of real-world combinatorial optimization problems. These benchmarks have been widely used in comparison studies [35, 41], demonstrating their importance. However, as the benchmarks were originally designed for general optimization methods, it is unclear whether they can also be used in more specific domains, e.g., comparing algorithm configurators. For instance, Doerr et al. [8] investigated the suitability of the BBOB testbed for simulating hyperparameter optimization problems and found that the BBOB functions may not adequately represent certain parameter tuning problems.

In contrast to synthetic benchmarks, *real-world* benchmarks specifically tailored for AC have also been developed [23, 24]. These benchmarks are designed to encompass diverse AC problems that vary across key dimensions, e.g., the objective function, configuration space, or the type of target algorithm objective [36]. While these benchmarks are well-suited for comparing algorithm configurators, they also come with limitations. Since the dimensions of the AC problems are often combined within benchmark tasks, it can be pretty challenging to determine why a configurator performs well on a given problem instance. Additionally, real-world benchmarks offer limited control over specific problem dimensions. For example, AClib [24] features parameter spaces ranging from 2 to 270 parameters and a single problem class consisting of 768 parameters. However, there is a significant gap in the coverage of problems with intermediate-sized parameter spaces.

Moreover, all of the aforementioned benchmarks, whether synthetic or real-world, follow a *task-oriented* evaluation paradigm, which evaluates configurators based on their performance across a set of diverse tasks. Ensuring task diversity is hence crucial for making robust and generalizable claims about the suitability of different configurators. In contrast, *capability-based* benchmarking offers a distinct approach. As outlined by Hernández-Orallo [16], this paradigm evaluates systems based on their inherent capabilities. It begins with identifying core capabilities that a configurator should possess and then specifically designs synthetic test functions to assess these capabilities. A synthetic, capability-based benchmark offers a unique combination of advantages. This approach enables a more targeted evaluation, facilitating the identification of the strengths and weaknesses of different configurators. Additionally, it enhances interpretability and control, as the synthetic nature allows for complete oversight of the dimensions within the algorithm configuration problem.

4 SynthACticBench Overview

We propose the synthetic benchmark `SynthACticBench`, inspired by the approach of José Hernández-Orallo, who advocates for evaluating a system - in our case, an algorithm configurator - "based on its cognitive abilities rather than the specific tasks it is built to solve" [16, p. 1]. Following this principle, we first collect capabilities that we assume to be essential for algorithm configuration and then design synthetic test functions to evaluate these capabilities.

Capabilities. We group the capabilities into *configuration space capabilities (C1-C7)* and *objective function capabilities (O1-O7)*, as illustrated in Figure 1. While the former investigates different structures in the configuration space, e.g., parameter interactions, mixed-type parameters, or conditional parameter domains, the latter explores characteristics of the function itself, such as being censored, having multiple objectives or temporal shifts. In principle, the target functions used to evaluate objective function capabilities can be combined with all functions designed to assess configuration space capabilities. Notably, the *configuration space* is also one of the key dimensions in the problem view classification scheme proposed by Schede et al. [36], whereas the dimensions *target algorithm objective type*, *configuration adjustment*, and *external runtime setting* are included in the objective function capabilities. We provide an overview of the included capabilities in Table 1 with references to relevant papers that motivate the evaluation of most of them. Further detailed discussions are provided in Sections 5 and 6.

Benchmark. We introduce `SynthACticBench` to offer greater control and flexibility over key dimensions of algorithm configuration problems. Each benchmark within `SynthACticBench` is designed to evaluate a specific capability, defined by one or multiple synthetic target functions $f : \boldsymbol{\Theta} \times \mathcal{I} \to \mathbb{R}$ to be minimized. We adhere to the standard AC notation, with the cost of a specific parameter configuration given by the function value of that configuration: $c(\theta, i) = f_i(\theta)$. Thus, the synthetic target function f corresponds to the cost metric, assessing an algorithm configuration for a problem instance, and its input parameters $\boldsymbol{\Theta}$ correspond to the configuration parameters, respectively. Consequently, the algorithm configurator is concerned with minimizing the target function over a set of problem instances, which corresponds to identifying the optimal parameter configuration.

Table 1: Capabilities that SynthACticBench evaluates.

	Benchmark Name	Capability
Config. Space	C1: Relevant Parameters [12, 19]	Identify and handle relevant parameters.
	C2: Parameter Interactions [31, 33]	Detect and exploit parameter interactions.
	C3: Mixed-Type [4, 24]	Tune mixed-type configuration spaces.
	C4: Activation Structures [22, 40]	Support activation structures.
	C5: Hierarchical Structures [42]	Manage hierarchical configuration structures.
	C6: Shifting Domains	Adapt to shifts in parameter domains.
	C7: Crashing Parameters [42]	Handle crashing parameter evaluations.
Objective Func.	O1: Deterministic Objective	Avoid redundant evaluations.
	O2: Noisy Objective [36]	Adapt sampling behaviour to noise level.
	O3: Multiple Objectives [36]	Optimize multiple objectives simultaneously.
	O4: Time-Dependent	Manage time-dependent distributional drift.
	O5: Right-Censored [36]	Handle censored information (e.g. timeouts).
	O6: Multi-Modal Landscapes	Escape local optima.
	O7: Single-Peak Landscapes	Search with uninformative feedback.

Many of the synthetic target functions in SynthACticBench are based on the function $f(\theta_1, \ldots, \theta_n) = \sum_{i=1}^{n} q_i(\theta_i)$ with quadratic components $q_i(\theta_i) = u \cdot \theta_i^2 + v \cdot \theta_i + w$, where the parameters u, v and w are sampled from a uniform distribution $\mathcal{U}[a, b]$. We will refer to this function as *quadratic target base function*. We choose quadratic functions as the base due to their versatility and simplicity. They offer enough flexibility to model a wide range of function behaviors through adjustable curvature and complexity by setting the parameters u, v, and w. At the same time, their relative simplicity ensures that the main focus of the benchmark lies in optimizing the constructed target function rather than the base function itself. Further, the simplicity maintains the ease of implementation, which is crucial for the practicality of our benchmark. For notational convenience, we denote a configuration $\theta = (\theta_1, \ldots, \theta_n)$ as the values of parameter setting θ_i in the following.

Instance generation. For each benchmark, we generate multiple homogeneous instances by varying the output space of the corresponding target function. For each instance, we sample an offset from a standard normal distribution $\delta_i \sim \mathcal{N}(\mu, \sigma^2)$ and add it to the function's output: $f_i = f_i(\theta) + \delta_i$. The AC problem is given by:

$$\theta^* \in \arg\min_{\theta \in \mathbf{\Theta}} \mathbb{E}_{\delta_i \sim \mathcal{N}(\mu, \sigma^2)} \left[f_i(\theta) + \delta_i \right].$$

For example, consider optimizing the target function $f(\theta) = 5 \cdot \theta^2 - 2 \cdot \theta + 0.3$ with domain $\Theta = [-10, 20] \subset \mathbb{R}$. Here, the offset value sampled from $P = \mathcal{N}(0, 1)$ is 0.3, and the only parameter to be optimized is θ. An algorithm configurator would aim to find the optimal parameter $\hat{\theta} = 0.2$, which is the global minimum, and hence, yields the minimal cost. Across different instances, the coefficients (5 and -2 in this case) remain constant, hence also the optimal parameter $\hat{\theta}$ is the same. However, the offset value varies, with larger values leading to more challenging instances.

5 Configuration Space Capabilities

Configuration space capabilities evaluate how well algorithm configurators handle various structural challenges in the configuration space. This includes, e.g., identifying relevant parameters, handling mixed-type parameters, and many more. For each capability, we follow a consistent structure: we first provide the motivation behind testing this capability, then describe the capability being evaluated, followed by the scenario in which it is tested, and finally, detail the specific benchmark implementation.

5.1 C1: Relevant Parameters

Algorithms may expose hundreds of parameters [23]. However, in many cases, only a handful of those parameters substantially affect the algorithm's performance [12, 19].

Capability. Identify effective (relevant) parameters and handle noisy (irrelevant) parameters.

Scenario. The configurator is presented with a set of parameters, but only a subset of these are relevant to the configuration process and influence the function value.

Synthetic Benchmark. To evaluate this capability, we specify the target function $f : [-100, 100]^n \to \mathbb{R}$ as the sum of quadratic and noisy functions $f(\theta) = \sum_{i=1}^{p} q_i(\theta_i) + \sum_{j=1}^{d} n_j$ where $p + d = n$. The quadratic components are denoted by $q_i(\theta_i)$, while the noisy samples are drawn from the distribution $n_j \sim \mathcal{N}(0, 100)$, hence being independent of the parameters θ_j. Thus, the minimum of the target function occurs when $(\theta_1, \ldots, \theta_p)$ correspond exactly to the minima of their respective quadratic subfunctions q_i, while the parameters $(\theta_{p+1}, \ldots, \theta_{p+d})$ can take arbitrary values.

5.2 C2: Parameter Interactions

In algorithm configuration and related fields, such as hyperparameter optimization, a common assumption refers to the interdependence of parameters and that they need to be tuned together to account for interaction effects [31, 33].

Capability. Detect and effectively exploit parameter interactions by optimizing them jointly.

Scenario. The configurator is presented with a set of parameters that exhibit interaction effects. For example, a change in one parameter influences the behavior of others, making independent optimization insufficient to achieve the best outcome.

Synthetic Benchmark. We implement the well-known *Rosenbrock function* [34] and the *Ackley function* [1]. The *Rosenbrock function* is defined over the domain $[-5, 10]^n$ and features a narrow, curved valley leading to its global minimum at $(1, \ldots, 1)$. The *Ackley function* is characterized by a nearly flat outer region, numerous local minima, and a single global minimum at $(0, \ldots, 0)$. It is defined over the domain $[-32.768, 32.768]^n$.

5.3 C3: Mixed-Type Configuration Spaces

A general challenge of algorithm configuration is that configurators must handle mixed-type configuration spaces [4, 36].

Capability. Effectively tune configuration spaces consisting of different types of parameters.

Scenario. The configurator is presented with a set of mixed-type parameters, including continuous, categorical, boolean, and integer.

Synthetic Benchmark. This benchmark is implemented as a composition of functions with categorical ($\mathcal{K}$), boolean ($\mathbb{B}$), integer ($\mathbb{Z}$), and continuous ($\mathbb{R}$) input domains: $\Theta^n = \mathcal{K}^{n_1} \times \mathbb{B}^{n_2} \times \mathbb{Z}^{n_3} \times \mathbb{R}^{n_4}$. To each parameter domain, we assign a pre-specified proportion of the total number of dimensions, i.e. $n = n_1 + n_2 + n_3 + n_4$. We bound each categorical space $\mathcal{K}$ to a random number of categories ranging between 3 and 20. We also bound each integer space $\mathbb{Z}$ to the interval of $[-100, 100]$ and each $\mathbb{R}$ to the interval of real-values $[-100, 100]$, respectively. For the final target function, we use the above defined *quadratic target base function*. However, we transform the categorical and boolean variables into continuous

variables using uniform binning over their respective ranges before querying the target function, while the integer and float parameters are used directly in the optimization process.

5.4 C4: Activation Structures

Complex algorithm systems are typically based on submodules that also expose parameters that need to be tuned. When different submodules of the same type or serving the same sub-task can be toggled, the effectiveness of tuning their exposed parameters depends on the choice of the submodule and thus changes depending on the corresponding parameter's value [22, 40].

Capability. Handle activation structures when parameters become active or inactive depending on the value of another parameter.

Scenario. The configurator is presented with a set of parameters that can be grouped into distinct subsets each of which is active if and only if a categorical parameter takes a certain value.

Synthetic Benchmark. This benchmark builds on the *quadratic target base function* with domain $\Theta = [-100, 100]^{n-1} \times \mathcal{K}$. The $n-1$ non-categorical parameters are partitioned into g groups, and the categorical parameter can take values $k \in \{1, \ldots, g\}$. Depending on k, only the parameters inside the activated group are evaluated by the corresponding subfunctions. For example, for a given input vector $\theta = (\theta_1, \ldots, \theta_{n-1}, k)$, assume that $\theta_3, \theta_4, \theta_5$ are grouped into the group $k = 2$. If k then takes the value 2, the target function evaluates only the activated parameters: $f(\theta) = \sum_{i=3}^{5} q_i(\theta_i)$.

5.5 C5: Hierarchical Configuration Spaces

Submodules themselves may include additional submodules that need to be chosen and tuned [42]. For instance, when configuring machine learning methods for multi-label classification, the configuration process can involve up to several layers of hierarchy. These layers may include ensembling methods for multi-label classifiers, multi-label classifiers, ensembling methods for single-label classifiers, and single-label classifiers.

Capability. Handle hierarchical configuration space structures with deeply nested submodules.

Scenario. The configurator is presented with a set of parameters that exhibit a hierarchical structure and expose parameters to be tuned on different levels.

Synthetic Benchmark. We again implement the *quadratic base target function* with the domain given by $\Theta = [-100, 100]^{n-2} \times \mathcal{K}^2$. The $n-2$ non-categorical parameters are partitioned into g groups, each further partitioned into h subgroups. The categorical parameters can take values $k_1 \in \{1, \ldots, g\}$ and $k_2 \in \{1, \ldots, h\}$, respectively. When c_1 and c_2 take certain values, only the parameters inside the activated group and subgroup are evaluated by the corresponding subfunctions. This benchmark essentially extends benchmark C4, with one additional level introduced.

5.6 C6: Shifting Domains

Sometimes, the choice of some parameter affects the domains of other parameters. For example, when configuring a decision tree, setting a low maximum depth can restrict the valid ranges of parameters like the minimum number of samples per leaf or the maximum number of features considered for a split.

Capability. Handle shifts in the parameter domains for still effectively optimizing with changed parameter domains.

Scenario. The configurator is presented with a set of parameters, where the domain is shifted depending on a certain parameter.

Synthetic Benchmark. In this benchmark, the domains of parameters depend on each other, so changing one parameter's value results in a change of another parameter's domain. The target function f is given by a sum of n quadratic functions. However, we implement two instantiations that differ in their bounded parameter domain. The original function f_{orig} is bounded by $[-100, 100]^n$, and the shifted function f_{shift} by $[-100, 100] \times [0, 200]^{n-1}$. The target function evaluates f_{orig} if the first parameter is negative, otherwise f_{shift}.

5.7 C7: Crashing Parameterizations

Parameterizations can also result in a crash of the algorithm due to e.g., a parameterization that does not fit into memory [42].

Capability. Handle configuration spaces with evaluation holes, i.e., subspaces that do not yield any performance evaluation.

Scenario. The configurator is presented with a set of parameters spanning a configuration space but which contains certain subspaces that result in a crash signal instead of a valid objective function evaluation.

Synthetic Benchmark. The target function is again given by $f(\theta) = \sum_{i=1}^{n} q_i(\theta_i)$. H owever, the domain is modified. Specifically, we randomly sample a hypercube with a given side length and exclude it from the domain. If a parameter configuration falls within this hypercube, the target function raises an error and does not return a value.

6 Objective Function Capabilities

Objective function capabilities assess how well algorithm configurators handle characteristics of the function itself, such as deterministic evaluations, multiple objectives, or temporal dynamics. The test functions designed to evaluate objective function capabilities can, in principle, be combined with those assessing configuration space capabilities. We follow the same structure for describing each capability as for the configuration space capabilities.

6.1 O1: Deterministic Objective Function

When dealing with deterministic algorithms, the relationship between the input parameters and the output is entirely predictable. This means that for a given set of parameter values, the algorithm will always produce the same result, and the output is not influenced by any form of randomness or noise.

Capability. The configurator can notice that an objective function is deterministic and thus does not waste its budget on multiple evaluations of the same solution candidate.

Scenario. The configurator is provided with a deterministic function to optimize, where the output is entirely determined by the specified parameter inputs. For any given set of parameter inputs, the function consistently produces the same result.

Synthetic Benchmark. The objective function is a composition of deterministic functions, which may include any aforementioned noise-free target functions, that are used to test other capabilities. As a result, it also inherits their domain.

6.2 O2: Noisy Objective Function

In real-world algorithm configuration problems, the evaluation of solution candidates is often noisy either because the algorithms themselves are non-deterministic or because the measurement of the corresponding performance criterion is noisy [36]. This is, for example, the case when considering runtime as a performance measure. Depending on the load of the execution environment and tasks scheduled by the operating system, runtime measurements may vary for the same parameterization.

Capability. The configurator can detect the noisiness of the objective function and adapt its sampling behavior to the noise level in the evaluation. Note that we assume homoscedastic noise, i.e., the noise is independent of any parameter values.

Scenario. The configurator is presented with a noisy objective function, where evaluations of the same configuration can yield different results due to the additional noise term.

Synthetic Benchmark. The target function is given by the *quadratic base target function*; however, for each function evaluation, an additional noise value independent of the parameter values is added to the function's output. This noise value is drawn from a distribution, which can be normal, uniform, or exponential. For example, assuming that the uniform distribution is chosen, the target function is given by $f(\theta) = \sum_{i=1}^{n} q_i(\theta_i) + \epsilon$ with $\epsilon \sim \mathcal{U}(-1, 1)$.

6.3 O3: Multiple Objectives

Objective functions for algorithms can be quite diverse and measure solution quality, runtime, memory occupation, fairness, interpretability, etc. Often, several objectives are required to make informative decisions between suitable compromises [36].

Capability. The configurator can handle the optimization of multiple objectives simultaneously and returns a set of non-dominated (ideally Pareto optimal) solutions.

Scenario. The configurator is given a multi-objective problem to optimize two or more conflicting objectives.

Synthetic Benchmark. To evaluate the capability of multi-objective optimization, we use the *ZDT1* and *ZDT3* functions [5], which were introduced by Zitzler et al. [44]. These functions are widely recognized test functions for multi-objective optimization problems featuring a set of Pareto-optimal solutions. Both functions are defined on $\mathbf{\Theta} = [0, 1]^n$.

6.4 O4: Time-Dependent Objective Functions

Environments may change over time, as do objective functions operating in these environments. For example, a computing system may degrade over time and slow down, e.g., through inefficient cache infrastructures, slower tasks scheduled by the operating system, etc. Consequently, performance measures, particularly for runtime or algorithm execution limited by runtime thresholds, may vary.

Capability. The configurator can efficiently optimize objective functions that exhibit a distributional drift over time.

Scenario. The configurator is tasked with optimizing an objective function that is influenced not only by the parameters but also by an external factor, such as time.

Synthetic Benchmark. We implement one target function that changes according to a linear drift f_{drift} and one, that changes according to an oscillation f_{oscil}, both defined on $\mathbf{\Theta} = [-100, 100]^n$. The former is given by $f_{\text{drift-op}}(\theta) = \sum_{i=1}^{n} q_i(\theta_i) + a + b \cdot t$ and the latter by $f_{\text{oscil-op}}(\theta) = \sum_{i=1}^{n} q_i(\theta_i) + \sin(a + b \cdot t)$ with $a = 1, b = 0.005$ and t the current time step.

6.5 O5: Right-Censored Objective Functions

Right-censored objective functions are quite common in the domain of algorithm configuration where typically runtime is optimized and the time for evaluating a single configuration is limited by the user to make the algorithm configuration problem decidable and to ensure sufficient exploration of the configuration space [36]. Moreover, this avoids stalling in certain subspaces where evaluation takes a long time, and evaluation resources are blocked when evaluating these expensive-to-evaluate candidates.

Capability. The configurator can effectively handle censored information, as a lower bound on the objective function is typically still conceivable. In the example of optimizing runtime with timeouts, we know that the true runtime exceeds the set timeout.

Scenario. The configurator is faced with objective functions that are censored due to timeouts or other limitations.

Synthetic Benchmark. This benchmark wraps any other target function in `SynthACticBench` and hence inherits its domain $\mathbf{\Theta}$, but cuts off the objective function at a certain level $\kappa > 0$. This cutoff κ has to be specified beforehand. During the evaluation, the function value of a configuration is only reported if it is not more than the cutoff away from $f_{min} = f(\theta^*)$, i.e., if $|f_{min} - f| < |f_{min}| \cdot \kappa$. Otherwise the function raises an error and does not return a function value. This benchmark is a special case of the crashing parameterization benchmark C7.

6.6 O6: Multi-Modal Landscapes

The landscape of an objective function can also be multi-modal, exhibiting multiple local optima that can also be quite close to the global optimum.

Capability. The configurator can handle multi-modal landscapes of the objective function and avoids getting stuck in local optima.

Scenario. The configurator is presented with an objective function that exhibits multiple local optima.

Synthetic Benchmark. The objective function is constructed s.t. it has multiple local optima in different regions of the configuration space. We implement the *Ackley function* [1] and the *Griewank function* [13]. The latter features a complex, multimodal landscape with many regularly distributed local minima, and the global minimum is located at $(0, \ldots, 0)$. We restrict the domain to $[-600, 600]^n$.

6.7 O7: Single-Peak Landscapes

A very difficult setting for optimizing an algorithm is when the landscape of the objective function is comparably flat and only exhibits a single peaked optimum. Informally speaking, the configurator is then tasked to find the needle in the haystack. Systematically searching the configuration space, improving over sampling simply at random, is particularly challenging.

Capability. The configurator can search with relatively uninformative feedback by the objective function and eventually identify a single peak in the landscape.

Scenario. The configurator is given a function that is constant almost everywhere, except for a small peak for the global minimum.

Synthetic Benchmark. The objective function is constructed so that it only hits a peak in a single place, and the remaining landscape is flat. This benchmark is defined over a bounded domain $[-100, 100]^n$. A specific subspace, characterized as a hyper-cube of a predefined width, yields the global minimum value of $f(\theta^*) = 0.0$. All regions outside this hyper-cube evaluate to 1.

7 Experiments

We conduct an empirical study comparing the algorithm configurators irace [28], SMAC [18, 26], and random search as a baseline across all benchmark problems specified in `SynthACticBench`. For the multi-objective benchmark O3, we exclude irace from the evaluation since, to the best of our knowledge, it cannot deal with multi-objective scenarios.

7.1 Experimental Setup

For each benchmark, we define the synthetic target function that evaluates the corresponding capability in a 10-dimensional space. If the target function is based on the *quadratic target base function*, we sample the parameters u, v, and w from the uniform distribution $\mathcal{U}[-10, 10]$. Further, we sample offsets from the normal distribution $\mathcal{N}(0, 2)$ to generate 500 problem instances for each target function. This number is inspired by the average number of instances in the AClib benchmark suite. Regarding the specific benchmark choices, we make the following adjustments. In C1, we select three relevant parameters and seven noisy ones. In C4, the non-categorical parameters are divided into two groups, with only one group being activated by the remaining categorical parameter. In C5, we define three groups, each further split into two subgroups. For the objective function benchmarks, we modify O1 by wrapping the noisy benchmark O2 while modifying the noise sampling distribution to always return zero. In O5, we set the cutoff κ to 0.8. In the benchmarks C7 and O7, we set the cube side length and peak width to 0.5 times the length of the first dimension, respectively. At first glance, this choice may seem overly generous. However, due to the curse of dimensionality, the relative size of the cube shrinks rapidly as the number of dimensions increases.

Each configurator executes 5,000 trials per benchmark, and we repeat each experiment with 20 independent random seeds. All experiments were conducted using 2 CPU cores and 8 GiB of RAM per run. The computations were carried out on HPC nodes equipped with two AMD Milan 7763 processors and 256 GiB of main memory.

7.2 Algorithm Configurators

We consider two state-of-the-art algorithm configurators and aim to benchmark them with respect to their capabilities drafted in `SynthACticBench`: SMAC [18, 26] and irace [30]. While we initially planned to include the Gender-based Genetic Algorithm (GGA) [2] and the Golden Parameter Search (GPS) [32] in our benchmark, implementation challenges prevented its inclusion, and we leave it for future work.

SMAC. SMAC [18, 26] is short for sequential model-based algorithm configuration and represents a Bayesian optimization framework that uses surrogate models (typically random forests) to efficiently explore and exploit the configuration space of the target algorithm. It is particularly suited for scenarios with expensive evaluations, allowing for robust tuning of machine learning models and combinatorial optimization algorithms.

irace. irace [30] is an iterated racing-based approach that sequentially eliminates poorly performing configurations based on statistical tests. As for genetic algorithms in general, offspring configurations are sampled from the fittest individuals via an individual-specific probability distribution over the configuration space, narrowing the probability distributions over time and thereby fostering exploitation in the vicinity of the stronger configurations. It is efficient for scenarios where evaluations are noisy or expensive, focusing computational effort on the most promising configurations.

7.3 Evaluation Method

We evaluate the performance of each configurator as follows. For every benchmark function, we compute the final regret, which is defined as the absolute difference between the function value of the final configuration found by the configurator, $f(\theta_{\text{conf}})$, and the known global optimum, $f(\theta^*)$. The regret of this configuration is hence given by:

$$\mathcal{R}_{\text{conf}} = \left| f(\theta_{\text{conf}}) - f(\theta^*) \right|.$$

For the multi-objective benchmark O3, we use the difference between the true Pareto front's hypervolume and the hypervolume of the Pareto front of the final configurations identified by the configurator. The regrets are averaged over 20 seeds. To present the regrets in a unified plot, we account for differences in scale across different benchmarks by normalizing them. Let $\mathcal{R}_{\text{max}}$ denote the regret of the final configuration of the worst configurator for a given benchmark. We then define the normalized regret as follows:

$$\mathcal{R}^{\text{norm}}_{\text{conf}} = \frac{\mathcal{R}_{\text{conf}}}{\mathcal{R}_{\text{max}}}.$$

Thus, it holds that $\mathcal{R}^{\text{norm}}_{\text{conf}} \in [0, 1]$ and $\mathcal{R}^{\text{norm}}_{\text{conf}} = 1$ precisely if the configurator exhibits the worst performance among those compared. Notably, the regret can be exactly 0, as the target functions are synthetic and their exact minima are known. Furthermore, to visualize the results in the bar plots shown in Figure 2, where a higher bar corresponds to better performance, we apply the transformation: $g(\mathcal{R}^{\text{norm}}_{\text{conf}}) = 1 - \mathcal{R}^{\text{norm}}_{\text{conf}}$. The only exception is irace on the O3 benchmark, which we were unable to execute. For this case, we set the value in the bar plot to a value below 0 to indicate that it is not applicable.

7.4 Results

The bar plots in Figure 2 provide a comparative analysis of the performance of SMAC, irace, and random search across the various capabilities defined in `SynthACticBench`. For each capability, we display a bar per configurator, representing the transformed normalized regret, higher values hence indicate better performance.

Several key trends can be observed: SMAC and irace exhibit noticeably different performance profiles, reflecting their distinct strengths. SMAC and irace both perform well on benchmark C6, which involves shifting domains; benchmark C1, which focuses on identifying relevant parameters; and benchmark O6, which features multimodal objective functions. SMAC demonstrates superior

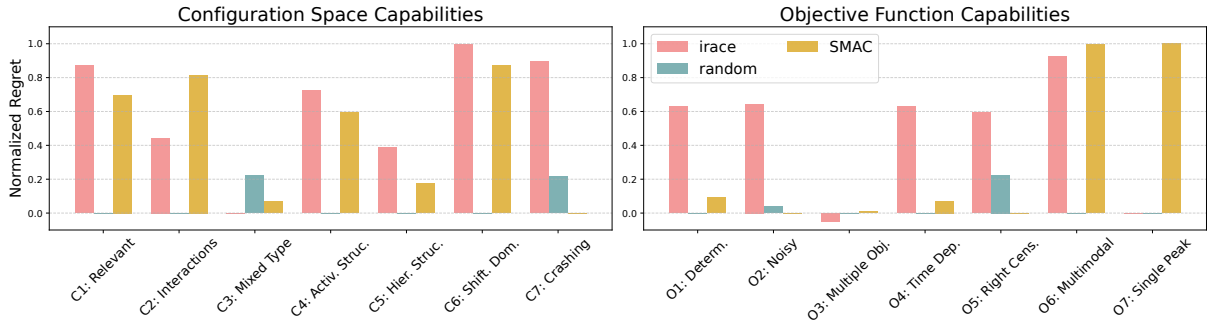

Figure 2: Evaluated configuration space capabilities (left) and objective function capabilities (right) of the algorithm configurators irace and SMAC and random search. A larger covered area indicates better performance.

performance on benchmark C2, which evaluates the ability to identify interactions between parameters, and benchmark O7, which involves a single peak in the objective function. This can be attributed to SMAC's use of a surrogate model to approximate the target function and, hence, the ability to exploit parameter interactions effectively. On the other hand, irace excels in benchmarks that involve crashing parameterizations (C7), time-dependent behaviors (O4), and activation structures (C4). This is likely due to irace's ability to adaptively focus on the most promising configurations, allowing it to adapt the population over time.

We note several details: (i) crashing parameterizations are not handled in the same way by the current SMAC implementation as in the original one [17], limiting the performance on C7; (ii) using surrogate-models with an i.i.d. assumption to guide the search is a drawback in view of O4; and (iii) it is somewhat surprising that SMAC underperforms w.r.t activation structures, since it is a common belief that SMAC's random-forest-based surrogate models should handle them well. In some cases, the differences between SMAC and irace appear marginal in the plots. This is due to the normalization, which is based on the regret of the worst-performing configurator. For instance, if random search performs particularly poorly on a given benchmark while both irace and SMAC perform well, their normalized regret values may appear similar. However, on benchmark O6, for example, their absolute regrets differ by a factor of 10, showing the benefits of a surrogate-guided global search. When comparing regrets across different benchmarks, we observe substantial variations in difficulty. For instance, SMAC achieves a regret of 0 on benchmark O7, indicating that it has found the true optimum. Similarly, SMAC and irace perform very well on benchmark O6 and almost reach the optimum. However, some benchmarks remain highly challenging despite their relatively low dimensionality (only ten dimensions), particularly those that involve a mix of different parameter types, such as continuous, categorical, and integer variables.

8 Conclusion and Future Work

In this paper, we presented capabilities relevant to evaluating algorithm configurators, including those related to the configuration space (e.g., parameter interactions, mixed-type parameters, or conditional domains) and the objective function (e.g., censorship or multiple objectives). Building on these, we proposed the synthetic, capability-based `SynthACticBench` benchmark to offer a structured, interpretable approach for comparing algorithm configurators. This framework facilitates the identification of specific strengths and weaknesses of different algorithm configurators. We further evaluated two state-of-the-art algorithm configurators, irace, and SMAC, on `SynthACticBench`. Our evaluation revealed that both configurators possess very different capabilities. SMAC excels in handling parameter interactions and single-peak objective functions, likely due to its use of surrogate models that allow it to explore the parameter space efficiently. In contrast, irace performs better on benchmarks involving time-dependent behaviors and activation structures, which is attributed to its iterative racing approach and adaptive refinement of the search space. Looking ahead, we aim to expand our evaluation to include additional algorithm configurators, such as GGA [2] and GPS [32]. Moreover, we plan to extend `SynthACticBench` to support both online and per-instance algorithm configuration, which would allow us to explore additional dimensions of algorithm configurator problems identified by Schede et al. [36], such as *training settings*, *configuration adjustment*, or *configuration scope*. Many of the capabilities incorporated in `SynthACticBench` - such as managing mixed parameter types (C3), hierarchical structures (C5), or noisy objectives (O2) - are also highly relevant to hyperparameter optimization [18, 39]. Therefore, the broader HPO and Bayesian optimization communities could also benefit from our benchmark. By extending `SynthACticBench` to include benchmarks tailored explicitly to HPO, we aim to facilitate systematic comparisons across different optimization methods and promote cross-domain advancements.

Acknowledgments

Marcel Wever and Marius Lindauer acknowledge funding by the European Union (ERC, "ixAutoML", grant no.101041029). Anna Lappe acknowledges funding by the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research. Eyke Hüllermeier has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Sklodowska-Curie grant agreement No 101073307.

References

[1] David H. Ackley. 1987. *A Connectionist Machine for Genetic Hillclimbing*. Ph. D. Dissertation.

[2] C. Ansótegui, Y. Malitsky, M. Sellmann, and K. Tierney. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, Q. Yang and M. Wooldridge (Eds.). 733–739.

[3] J. Bergstra and Y. Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.

[4] Mauro Birattari, Marco Chiarandini, Marco Saerens, and Thomas Stützle. 2011. *Learning Graphical Models for Parameter Tuning*. Technical Report. Citeseer.

[5] J. Blank and K. Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[6] A. Blot, H. Hoos, L. Jourdan, M. Kessaci-Marmion, and H. Trautmann. 2016. MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework. In *Learning and Intelligent Optimization - 10th International Conference, LION (Lecture Notes in Computer Science, Vol. 10079)*, P. Festa, M. Sellmann, and J. Vanschoren (Eds.). Springer, 32–47.

[7] D. Doblas, A. Nebro, M. López-Ibáñez, J. García-Nieto, and C. Coello. 2022. Automatic Design of Multi-objective Particle Swarm Optimizers. In *Swarm Intelligence - 13th International Conference, ANTS (Lecture Notes in Computer Science, Vol. 13491)*, M. Dorigo, H. Hamann, M. López-Ibáñez, J. García-Nieto, A. Engelbrecht, C. Pinciroli, V. Strobel, and C. Camacho-Villalón (Eds.). Springer, 28–40.

[8] Carola Doerr, Johann Dréo, and Pascal Kerschke. 2019. Making a case for (Hyper-)parameter tuning as benchmark problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM.

[9] C. Doerr, F. Ye, N. Horesh, H. Wang, O. Shir, and T. Bäck. 2020. Benchmarking discrete optimization heuristics with IOHprofiler. In *Applied Soft Computing*, Mario Köppen (Ed.), Vol. 88. 106027.

[10] K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown. 2015. Efficient Benchmarking of Hyperparameter Optimizers via Surrogates. In *Proceedings of the Twenty-ninth AAAI Conference on Artificial Intelligence (AAAI'15)*, B. Bonet and S. Koenig (Eds.). AAAI Press, 1114–1120.

[11] T. Elsken, J. Metzen, and F. Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20, 55 (2019), 1–21.

[12] C. Fawcett and H. Hoos. 2016. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics* 22, 4 (2016), 431–458.

[13] Andreas O. Griewank. 1981. Generalized descent for global optimization. *Journal of Optimization Theory and Applications* (1981).

[14] N. Hansen, A. Auger, S. Finck, and R. Ros. 2010. *Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup*. Research Report [inria-00462481]. Inria Saclay Ile de France.

[15] N. Hansen, A. Auger, R. Ros, O. Mersman, T. Tušar, and D. Brockhoff. 2020. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* (2020).

[16] José Hernández-Orallo. 2017. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement. *Artif. Intell. Rev.* (2017).

[17] F. Hutter, H. Hoos, and K. Leyton-Brown. 2011. Bayesian Optimization With Censored Response Data. In *NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'11)*, N. de Freitas, R. Garnett, F. Hutter, and M. Osborne (Eds.).

[18] F. Hutter, H. Hoos, and K. Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11) (Lecture Notes in Computer Science, Vol. 6683)*, C. Coello (Ed.). Springer, 507–523.

[19] F. Hutter, H. Hoos, and K. Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, E. Xing and T. Jebara (Eds.). Omnipress, 754–762.

[20] F. Hutter, H. Hoos, K. Leyton-Brown, and K. Murphy. 2009. An Experimental Investigation of Model-Based Parameter Optimisation: SPO and Beyond. In *Proceedings of the 11th Genetic and Evolutionary Computation Conference (GECCO'09)*, G. Raidl et al (Ed.). Morgan Kaufmann Publishers, 271–278.

[21] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (2009), 267–306.

[22] F. Hutter, H. Hoos, and T. Stützle. 2007. Automatic Algorithm Configuration based on Local Search. In *Proceedings of the Twenty-second AAAI Conference on Artificial Intelligence (AAAI'07)*, R. Holte and A. Howe (Eds.). AAAI Press, 1152–1157.

[23] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, and K. Leyton-Brown. 2017. The Configurable SAT Solver Challenge (CSSC). *Artificial Intelligence* 243 (2017), 1–25.

[24] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Stützle. 2014. AClib: a Benchmark Library for Algorithm Configuration. In *Proceedings of the Eighth International Conference on Learning and Intelligent Optimization (LION'14) (Lecture Notes in Computer Science)*, P. Pardalos and M. Resende (Eds.). Springer, 36–40.

[25] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. 2010. ISAC - Instance-Specific Algorithm Configuration. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, H. Coelho, R. Studer, and M. Wooldridge (Eds.). IOS Press, 751–756.

[26] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23, 54 (2022), 1–9.

[27] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub. 2015. AutoFolio: Algorithm Configuration for Algorithm Selection. In *Proceedings of the Workshops at Twenty-ninth AAAI Conference on Artificial Intelligence (AAAI'15)*. Association for the Advancement of Artificial Intelligence.

[28] M. López-Ibáñez, J. Dubois-Lacoste, L. Perez Caceres, M. Birattari, and T. Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[29] M. López-Ibáñez and T. Stützle. 2010. Automatic Configuration of Multi-Objective ACO Algorithms. In *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS'10) (Lecture Notes in Computer Science)*, M. Dorigo, M-Birattari, G. Di Caro, R. Doursat, A. Petrus Engelbrecht, D. Floreano, L. Gambardella, R. Groß, E. Sahin, H. Sayama, and T. Stützle (Eds.). Springer, 95–106.

[30] M. López-Ibánez and T. Stützle. 2014. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research* 235, 3 (2014), 569–582.

[31] Y. Pushak and H. Hoos. 2018. Algorithm Configuration Landscapes: - More Benign Than Expected?. In *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN'18)*, A. Auger, C. Fonseca, N. Lourenço, P. Machado, L. Paquete, and L. Darrell Whitley (Eds.). 271–283.

[32] Y. Pushak and H. Hoos. 2020. Golden parameter search: exploiting structure to quickly configure parameters in parallel. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*, J. Ceberio (Ed.). ACM Press, 245–253.

[33] Y. Pushak and H. Hoos. 2022. AutoML Loss Landscapes. *ACM Transactions on Evolutionary Learning and Optimization* 2, 3 (2022), 1–30.

[34] H. H. Rosenbrock. 1960. An Automatic Method for Finding the Greatest or Least Value of a Function. *Comput. J.* (1960).

[35] Maria Laura Santoni, Elena Raponi, Renato De Leone, and Carola Doerr. 2024. Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB. *ACM Trans. Evol. Learn. Optim.* (2024).

[36] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. 2022. A Survey of Methods for Automated Algorithm Configuration. *Journal of Artificial Intelligence Research* 75 (2022), 425–487.

[37] M. Schneider and H. Hoos. 2012. Quantifying Homogeneity of Instance Sets for Algorithm Configuration. In *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12) (Lecture Notes in Computer Science, Vol. 7219)*, Y. Hamadi and M. Schoenauer (Eds.). Springer, 190–204.

[38] Kate Smith-Miles and Leo Lopes. 2012. Measuring instance difficulty for combinatorial optimization problems. *Comput. Oper. Res.* (2012), 875–889.

[39] J. Snoek, H. Larochelle, and R. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Eds.). Curran Associates, 2960–2968.

[40] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. 2013. Auto-WEKA: combined selection and Hyperparameter Optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy (Eds.). ACM Press, 847–855.

[41] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. 2020. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*. ACM, 654–662.

[42] Marcel Wever, Alexander Tornede, Felix Mohr, and Eyke Hüllermeier. 2021. AutoML for Multi-Label Classification: Overview and Empirical Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[43] L. Xu, H. Hoos, and K. Leyton-Brown. 2010. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence (AAAI'10)*, M. Fox and D. Poole (Eds.). AAAI Press, 210–216.

[44] E. Zitzler, K. Deb, and L. Thiele. 2000. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation* 8, 2 (2000), 173–195.