Integrated vs. Sequential Approaches for Selecting and Tuning CMA-ES Variants

Diederick Vermetten, Hao Wang, Carola Doerr, Thomas Back

Integrated vs. Sequential Approaches for Selecting and Tuning CMA-ES Variants

Diederick Vermetten
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Hao Wang
Sorbonne Université, CNRS, LIP6
Paris, France

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

Thomas Bäck
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

ABSTRACT

When faced with a specific optimization problem, deciding which algorithm to apply is always a difficult task. Not only is there a vast variety of algorithms to select from, but these algorithms are often controlled by many hyperparameters, which need to be suitably tuned in order to achieve peak performance. Usually, the problem of selecting and configuring the optimization algorithm is addressed sequentially, by first selecting a suitable algorithm and then tuning it for the application at hand. Integrated approaches, commonly known as Combined Algorithm Selection and Hyperparameter (CASH) solvers, have shown promise in several applications.

In this work we compare sequential and integrated approaches for selecting and tuning the best out of the 4,608 variants of the modular Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We show that the ranking of these variants depends to a large extent on the quality of the hyperparameters. Sequential approaches are therefore likely to recommend sub-optimal choices. Integrated approaches, in contrast, manage to provide competitive results at much smaller computational cost. We also highlight important differences in the search behavior of two CASH approaches, which build on racing (irace) and on model-based optimization (MIP-EGO), respectively.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; *Bio-inspired optimization*; Algorithm design techniques.

KEYWORDS

Algorithm Selection, Algorithm Control, Iterative Optimization Heuristics, Black-Box Optimization

1 INTRODUCTION

In computer science, optimization has become an important field of study over the past decades. Because of its rising popularity and its high practical relevance, many different techniques have been introduced to solve particular types of optimization problems. As these methods are developed further, small modifications might lead the algorithm to behave better on specific problem types. However, it has long been known that no single algorithm variant can outperform all others on all functions, an observation that is formalized in the so-called *no-free-lunch theorems* [49]. This fact leads a new set of challenges for practitioners and researchers alike: How to choose which algorithm to use for which problem?

Even when limiting the scope to a small class of algorithms, the choice of which variant to choose can be daunting, leading practitioners to resort to a few standard versions of the algorithms, which might not be particularly well suited to their problem.

The problem of selecting an algorithm (variant) from a large set is commonly referred to as the *algorithm selection* problem [37]. However, the algorithm variant is not the only factor which has an impact on performance. The setting of the variable hyperparameters can also play a very important role [27, 29]. Choosing the right hyperparameter setting for a specific algorithm is known as the *algorithm configuration* problem, or – in the context of evolutionary computation – as the *parameter setting* problem [12].

In this work, we investigate the impact of both hyperparameter tuning and algorithm selection in the context of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). The CMA-ES family [19] is an important collection of heuristic optimization techniques for numeric optimization. In a nutshell, the CMA-ES is an iterative search procedure, which updates after each iteration the covariance matrix of the multivariate normal distribution that is used to generate the samples during the search, effectively learning second-order information about the objective function. Important contributions to the class of CMA-ES have been made over the years, which all reveal different strengths and weaknesses in different optimization contexts [5, 16, 18].

While most of the suggested modifications have been proposed in isolation, [41] suggested a framework which modularizes eleven popular CMA-ES modifications such that they can be combined to create a total number of 4,608 different CMA-ES variants. It was shown in [42] that some of the so-created CMA-ES variants improve significantly over commonly used CMA-ES algorithms. The framework provides a convenient way to study the impact of the different modules on different optimization problems [41]. In all previous works on the modular CMA-ES, the hyperparameters were set to some default values. To the best of our knowledge, it has not been investigated how sensitive the performance of the different variants is with respect to these hyperparameter settings.

The problems of selecting and configuring an appropriate algorithm for a given problem are, of course, highly interlinked. Therefore, it seems natural to tackle both problems at the same time. In practice, however, algorithm selection is typically based solely on

GECCO '20, July 8–12, 2020, Cancún, Mexico
2020. ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3389831

users' previous experience and preferences, whereas the tuning step is often entirely neglected: users mainly resort to default configurations that have previously been suggested in the literature or that happen to be the defaults in the implementation of the algorithm framework they use.

Data-driven techniques for algorithm selection exist (see [23] for a recent survey), but require previous training and are therefore not yet widely applied. Automated algorithm configuration [11, 25], in contrast, is (finally) gaining popularity – both in academic and in industrial applications. Several software frameworks such as, for example, SPOT [6], irace [30], SMAC [20], hyperband [28], MIP-EGO [47], or the more recent BOHB [13] are readily available to support the user in identifying suitable parameter settings.

Note, though, that even when data-driven parameter tuning is used, we still address algorithm selection and configuration sequentially, and not as a *Combined Algorithm Selection and Hyperparameter optimization* (CASH) problem, as the integrated problem was termed in [38]. First automated approaches for the CASH problems were studied in the Machine Learning community, focused on solving classification problems [14, 26]. The CASH-problem has also been studied in the context of creating SAT-solvers [50] or solving Mixed-Integer Programming problems [51].

In this work, we apply CASH to the problem of selecting and configuring variants of the CMA-ES.

The modularity of the used CMA-ES-framework also allows us to integrate *algorithm selection* and *configuration* into a single mixed-integer search space, where we can optimize both the algorithm variant and the corresponding hyperparameters at the same time. We show that such an integrated approach is competitive with sequential approaches based on complete enumeration of the algorithm space, while requiring significantly less computational effort. We also investigate the differences between two algorithm configuration tools, irace [30] and MIP-EGO [47] (see Section 2.4 for short descriptions). While the overall performance of these two approaches is comparable, the balance between algorithm selection and algorithm configuration shows significant differences, with irace focusing much more on the configuration task, and evaluating only a few different CMA-ES variants. MIP-EGO, in turn, shows a broader exploration behavior, at the cost of less accurate performance estimates.

2 EXPERIMENTAL SETUP

We summarize the algorithmic framework, the benchmark suite, the performance measures, and the two configuration tools, irace and MIP-EGO, which we employ for the tuning of the hyperparameters.

2.1 The Modular CMA-ES

Table 1 summarizes the eleven modules of the modular CMA-ES from [41]. Out of these, nine modules are binary and two are ternary, allowing for a total number of 4,608 different possible CMA-ES variants. The modular CMA-ES is available at [39].

So far, all studies on the modular CMA-ES framework have used default hyperparameter values [40, 41, 44]. However, it has been shown that substantial performance gains are possible by tuning these hyperparameters [1, 7], raising the question how much can be gained from combining the tuning of several hyperparameters with the selection of the CMA-ES variant. In accordance with [7], we focus on only a small subset of these hyperparameters, namely c_1, c_c and c_μ, which control the update of the covariance matrix. It is well known, though, that other hyperparameters, in particular the population size [3], have a significant impact on the performance as well, and might be much more critical to configure as the ones chosen in [7]. However, we will see that the performance of the CMA-ES variants is nevertheless strongly influenced by these three hyperparameters. In fact, we show in Section 4 that the ranking of the algorithm variants with default and tuned hyperparameters can differ significantly, indicating that a sequential execution of *algorithm selection* and *algorithm configuration* will not provide optimal results.

#	Module name	0	1	2
1	Active Update [22]	off	on	-
2	Elitism	(μ, λ)	$(\mu+\lambda)$	-
3	Mirrored Sampling [8]	off	on	-
4	Orthogonal Sampling [46]	off	on	-
5	Sequential Selection [8]	off	on	-
6	Threshold Convergence [34]	off	on	-
7	TPA [15]	off	on	-
8	Pairwise Selection [2]	off	on	-
9	Recombination Weights [4]	$r_i(\mu)$	$\frac{1}{\mu}$	-
10	Quasi-Gaussian Sampling	off	Sobol	Halton
11	Increasing Population [3, 16]	off	IPOP	BIPOP

Table 1: Overview of the CMA-ES modules available in the used framework. The entries in row 9 specify the formula for calculating each weight w_i, with $r_i := \log(\mu+\frac{1}{2})-\frac{\log(i)}{\sum_j w_j}$.

2.2 Test-bed: the BBOB Framework

For analyzing the impact of the hyperparameter tuning, we use the Black-Box Optimization Benchmark (BBOB) suite [17], which is a standard environment to test black-box optimization techniques. This testbed contains 24 functions $f : \mathbb{R}^d \rightarrow \mathbb{R}$, of which we use the five-dimensional versions. Each function can be transformed in objective and variable space, resulting in separate instances with similar fitness landscapes. A large part of our analysis is built on data from [44], which uses 5 independent runs on the first 5 instances of each function, for each algorithm variant. This data is available at [43].

2.3 Performance Measures

We next define the performance measures by which we compare the different algorithms. First note that CMA-ES is a stochastic optimization algorithm. The number of function evaluations needed to find a solution of a certain quality is therefore a random variable, which we refer to as the *first hitting time*. More precisely, we denote by $t_i(v, f, \phi)$ the number of function evaluations that the variant v used in the i-th run until it evaluates a solution x satisfying $f(x) \leq \phi$ for the first time. If target ϕ is not hit, we define $t_i(v, f, \phi) = \infty$. To be consistent with previous work, such as [44], we decide to use two estimators of the mean of the hitting time distribution. Both these measures are very commonly applied. A discussion on this can be found in [17].

Definition 2.1. For a set of functions $\mathcal{F} = \{f^{(1)}, \ldots, f^{(K)}\}$, the *average hitting time (AHT)* is defined as:

$$\tilde{T}(v, \mathcal{F}, \phi) = \frac{1}{nK} \sum_{i=1}^{n} \sum_{j=1}^{K} \min\{t_i(v, f^{(j)}, \phi), P\}$$

When a run does not succeed in hitting target ϕ, we have $t_i(v, f, \phi) = \infty$. In this case, a penalty $P \geq B$ (where B is the maximum budget) is applied. Usually, this penalty is set to ∞, in which case this value is called AHT. Otherwise, it is commonly referred to as *penalized* AHT.

In contrast, the *expected running time (ERT)* equals:

$$\mathrm{ERT}(v, \mathcal{F}, \phi) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{K} \min\{t_i(v, f^{(j)}, \phi), B\}}{\sum_{i=1}^{n} \sum_{j=1}^{K} \mathbb{1}\{t_i(v, f^{(j)}, \phi) < \infty\}}.$$

Previous work has shown ERT, as opposed to AHT, to be a consistent, unbiased estimator of the mean of the distribution hitting times [3]. However, it is good to note that ERT and AHT are equivalent when all runs of variant v manage to hit target ϕ.

In the context of the modular CMA-ES, the CASH problem is adopted as follows. Given a set of CMA-ES variants V, a common hyperparameter space H, a set of function instances $\mathcal{F}$, and a target value ϕ, the CASH problem aims to find the combined algorithm and hyperparameter setting that solves the problem below:

$$v^*, \mathbf{h}^* = \operatorname*{argmin}_{v \in V, \mathbf{h} \in H} \mathrm{ERT}\left(v_h, \mathcal{F}, \phi\right).$$

Note here that we aim at finding the best (variant,hyperparameter)-pair for each of the 24 BBOB functions individually and we consider as $\mathcal{F}$ the set of the first five instances of each function. We do not aggregate over different functions, since the benchmarks can easily be distinguished by exploratory landscape approaches [7].

2.4 Hyperparameter Tuning

In this work, we compare two different off-the-shelf tools for mixed-integer hyperparameter tuning: irace and MIP-EGO.

Irace [30, 31] is an algorithm designed for hyperparameter optimization, which implements an iterated racing procedure. irace is implemented in R and is freely available at [32]. For our experiments, we use the elitist version of irace with adaptive capping, which we briefly describe in the following.

irace works by first sampling a set of candidate parameter settings, which can be any combination of discrete, continuous, categorical, or ordinal variables. These parameters are empirically evaluated on some problem instances, which are randomly selected from the set of available instances. After running on x instances, a statistical test is performed to determine which parameter settings to discard. The remaining parameter settings are then run on more instances and continuously tested every ℓ iterations until either only a minimal number of candidates remain or until the budget of the current iteration is exhausted. The surviving candidates with the best average hitting times are selected as the elites.

After the racing procedure, new candidate parameter settings are generated by selecting a parent from the set of elites and "mutating" it, as described in detail in [30]. After generating the new candidates, a race is started with these new solutions, combined with the elites. Since we use an elitist version of irace, these elites are not discarded until the competing candidates have been evaluated on the same instances which the elites have already seen. This is done to prevent the discarding of candidates which perform well on the previous race based on only a few instances in the current race.

Apart from using elitism and statistical tests to determine when to discard candidate solutions, we also use another recently developed extension of irace, the so-called *adaptive capping* [9] procedure. Adaptive capping helps to reduce the number of evaluations spent on candidates which will not manage to beat the current best. Adaptive capping enables irace to stop evaluating a candidate once it reaches a mean hitting time which is worse than the median of the elites, indicating that this candidate is unlikely to be better than the current best parameter settings.

Mixed-Integer Parallel Efficient Global Optimization (MIP-EGO) [47, 48] is a variant of Efficient Global Optimization (EGO, a sequential model-based optimization technique), which can deal with mixed-integer search-spaces. Because EGO is designed to deal with expensive function evaluations, and this variant has the ability to deal with continuous, discrete, and categorical parameters, it is also well suited to the hyperparameter tuning task. It uses a much different approach as irace, as we will describe in the following.

EGO works by initially sampling a set of solution candidates from some specified probability distribution, specifically a Latin hypercube sampling in MIP-EGO. Based on the evaluation of these initial points, a meta-model is constructed. Originally, this was done using Gaussian process regression, but MIP-EGO uses random forests to be able to deal with mixed-integer search spaces. Based on this model, a new point (or a set of points) is proposed according to some metric, called the *acquisition function.* This can be as simple as selecting the point with the largest *probability of improvement* (PI) or the largest *expected improvement* (EI). More recently, acquisition functions based on moment-generating function of the improvement have also been introduced [48]. For this paper, we use the basic EI acquisition function, which is maximized using a simple evolution strategy. After selecting the point(s) to evaluate, the meta-model is updated according to the quality of the solutions. The process is repeated until a termination criterion (budget constraint in our case) is met.

3 SENSITIVITY OF CMA-ES VARIANTS TO ITS HYPERPARAMETERS

As mentioned previously, it is well known that the performance of CMA-ES can be highly dependent on hyperparameter settings. In this work however, we don't just focus on a single CMA-ES algorithm, but a large set of different variants. For such variants of CMA-ES, no study has yet looked at the interplay between hyperparameters and module settings. If these interactions would be relatively minor, it would mean that selecting the algorithm variant and tuning it can be seen as two separate steps which can be executed sequentially. However, if the different algorithm variants require different hyperparameters in order to achieve optimal performance, this might result in some variants having default hyperparameters which are already close to optimal for certain problems, while others have very poor hyperparameter settings.

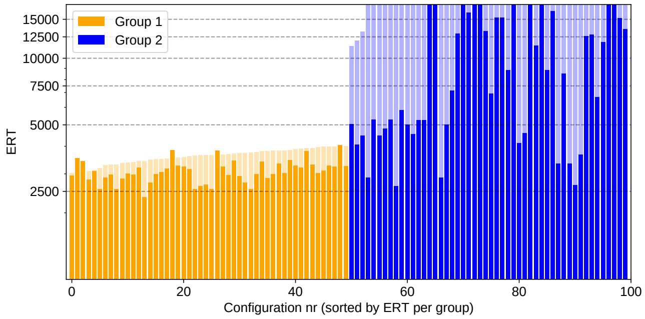

Figure 1: ERT for two groups of CMA-ES variants: The top 50 according to ERT and a group of 50 randomly selected variants. The ERT before tuning is shown in light color (based on 25 runs), while the ERT after tuning and verification is shown in a darker shade.

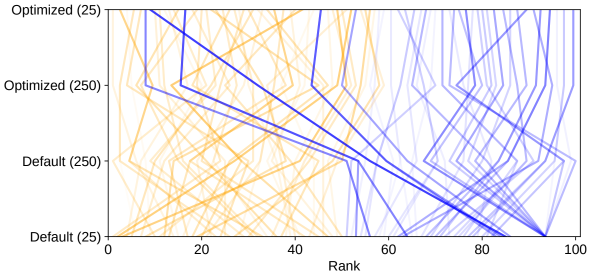

Figure 2: Evolution of ERT-based ranking (lower rank is better) of 100 algorithm variants for 5 and 50 runs on each of the first five instances of F12, respectively. Default refers to the ERT using the default hyperparameters while optimized is the best ERT using the tuned hyperparameters as found by MIP-EGO. Darker lines correspond to larger changes in ranking. Colors correspond to the grouping of variants as in Figure 1.

Hyperparameter tuning might then lead to some variants that perform relatively poorly with default hyperparameters, to outperform all others when the hyperparameters have been sufficiently optimized. In such a scenario, the problem of selecting the algorithm variant and tuning it becomes much more intertwined, and requires much more effort to solve it effectively.

To illustrate the impact of hyperparameter tuning on the different algorithm variants, we present a short use-case in which we zoom into the performance of the modular CMA-ES variants on a single problem, the function F12 from the BBOB framework. After running all variants on the first five instances of this function for 5 runs each, we sort the variants by ERT values, and select the 50 top-performing ones for further investigation. We also create a set of 50 randomly chosen variants. Then, for each of these variants, the hyperparameters are tuned using MIP-EGO. The resulting ERTs are shown in Figure 1. From this figure, it is clear that the relative improvements are indeed much larger for the group of random variants. There are even some variants which start with very poor ERT and which, after the tuning process, become competitive with the variants from the first group.

We also rank these CMA-ES variants based on their ERTs, both with the default and tuned hyperparameters, and both for the 25 runs seen before and during the tuning and for the 250 verification runs. The resulting differences in ranking are shown in Figure 2. We observe two main effects here. One is the instability of the 25-run ERT values (the ranking changes when we move to the 250-run data), and the other effect are the large differences between the rankings of the algorithms with default versus with tuned hyperparameters. This clearly indicates that different variants have different amount of potential to gain from hyperparameter tuning. The choice of algorithm variant and hyperparameters is therefore indeed highly interlinked, potentially making it challenging for sequential CASH methods to find optimal combinations. As we discuss in Section 4.2.2, large dispersions between the improvements of different algorithm variants indicate that these findings apply more broadly to all 24 BBOB-functions.

4 CASH-BASELINE: SEQUENTIAL METHODS

Even though sequential execution of algorithm selection and configuration might not be optimal, it is still a useful baseline to set. Since the ERT for all CMA-ES variants on all benchmark functions is available, a complete enumeration technique would be the simplest form of algorithm selection. The simplest way to do algorithm configuration would then be to tune the best algorithm according to the enumeration. However, this ignores the potential of other algorithms overtaking it when their hyperparameters are tuned. In an attempt to mitigate this, we can choose to tune a set of algorithms instead of a single one. More specifically, we define two sequential methods as follows:

- **Naïve sequential**: Perform hyperparameter tuning (using MIP-EGO) on the one CMA-ES variant with the lowest ERT
- **Standard sequential**: Perform hyperparameter tuning (using MIP-EGO) on a set of 30 variants. We have chosen to consider the following set of variants to have a wide representation of module settings, and to be able to fairly compare the impact of hyperparameter tuning across functions:
  - The 10 variants with lowest ERT.
  - The 10 variants ranked 200-210 according to ERT.
  - The 10 'common' variants, i.e., CMA-ES variants previously studied in the literature (see Table V in [41]).

For both of these methods, the execution of MIP-EGO has a budget of 200 ERT-evaluations, each of which is based on 25 runs of the underlying CMA-ES variant (i.e., 5 runs per each of the five instances). Since the observed hitting times show high variance, we validate the ERT values by performing 250 additional runs (50 runs per each instance). All results shown will be ERT from these verification runs, unless stated otherwise. The variant selection and hyperparameter tuning is done separately for each function.

4.1 First Results

While the two sequential methods introduced are quite similar, it is obvious that the naïve one will always perform at most equally well as the standard version, since the algorithm variant tuned in the naïve approach is always included in the set of variants tuned by the standard method (the same tuned data is used for both methods to exclude impact of randomness). In general, the standard sequential

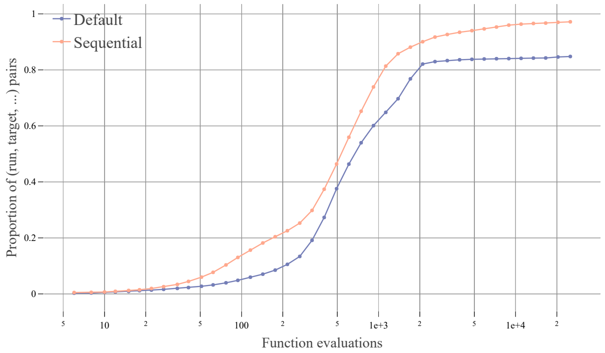

Figure 3: ECDF curve over all benchmark functions of both the standard sequential method as well as the default CMA-ES. Figure generated using IOHprofiler [10].

method achieves ERTs which are on average around 20% lower than the naïve approach.

To better judge the quality of these sequential methods, we compare their performance to the default variant of the CMA-ES, which is the variant in which all modules are set to 0. This can be done based on ERT, for each function, but that does not always show the complete picture of the performance. Instead, the differences between the performances of the sequential method and the default CMA-ES are shown in a *Empirical Cumulative Distribution Function (ECDF)*, which aggregates all runs on all functions and shows the fraction of runs and targets which were hit within a certain amount of function evaluations. This is shown in Figure 3 (targets used available at [45]). From this, we see that the sequential approach completely dominates the default variant. When considering only the ERT, this improvement is on average 73%.

As well as comparing performance against the default CMA-ES, it can also be compared against the best modular variant with default hyperparameters, i.e. the result of pure algorithm selection. For this, the standard sequential approach manages to achieve a 24.7% improvement in terms of average ERT, as opposed to 6.3% for the naïve version. Of note for the naïve version is that not all comparisons against the pure algorithms selection are positive, i.e. for some (5) functions it achieves a larger ERT. This might seem counter-intuitive, as one would expect hyperparameter tuning to only improve the performance of an algorithm. However, this is where the inherent variance of evolution strategies has a large impact. In short, because ERTs seen by MIP-EGO are based on only 25 runs, it may happen that a sub-optimal hyperparameter setting will be selected. This is explained in more detail in the following sections.

4.2 Pitfalls

The sequential methods described here have the advantage of being based on algorithm selection by complete enumeration. In theory, this would be the perfect way of selecting an algorithm variant. However, since CMA-ES are inherently stochastic, variance has a large effect on the ERT, and thus on the algorithm selection. This might not be an issue if one assumes that hyperparameter tuning has an equal impact on all CMA-ES variants. Unfortunately, as shown in Section 3, this is not the case in practice.

4.2.1 Curse of High Variance. The inherent variance present in the ERT-measurements does not only cause potential issues for the algorithm selection, it also plays a large role in the hyperparameter configuration. As previously mentioned, the ERT after running MIP-EGO can be larger than the ERT with the default hyperparameters, even though the default hyperparameters are always included in the initial solution set explored by MIP-EGO. Since this might seem counter-intuitive, a small-scale experiment can be designed to show this phenomenon in more detail.

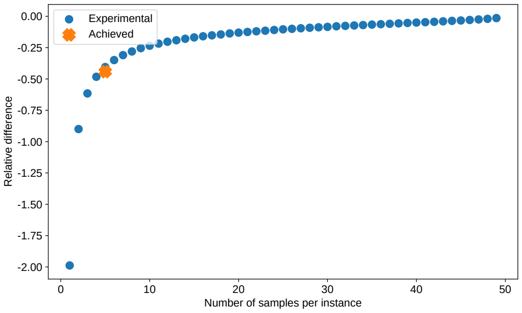

Figure 4: Average improvement of ERT achieved from 250 runs vs the value obtained after running MIP-EGO (25 runs) in orange, vs experimental improvement. Experimental improvement obtained over 100 repetitions of selecting k samples per instance for each variant and calculating their respective ERT.

This experiment is set up by first taking the set of 50 hitting times for each instance as encountered in the verification runs. Then, sample x runs per instance from these hitting times and calculate the resulting ERT. Repeat this 10 times, and take the minimal ERT. Then we can compare the original ERT to this new value. This is similar to the internal data seen by MIP-EGO, if we assume that 5% of the variants it evaluated have a similar hitting time distribution. When preforming this experiment on a set of 100 algorithm variants on F21, we obtain the results as seen in Figure 4, which shows that the actual differences between ERTs given by MIP-EGO and those achieved in the verification runs matches the difference we would expect based on this experiment.

4.2.2 Differences in Improvements. The differences in improvement between CMA-ES variants as seen for F12 in Section 3 are also highly dependent on the underlying test function. When executing the sequential approach mentioned previously, 30 variants are tuned for each function, and the ERTs are verified using 250 runs. The resulting data can then give some insight into the difference in terms of relative improvement possible per function, as is visualized in Figure 5. This shows that, on average, a relatively large performance improvement is possible for the selected variants. However, the distributions have large variance, and differ greatly per function. This highlight the previous findings of different variants receiving much greater benefits from tuned hyperparameters than others,

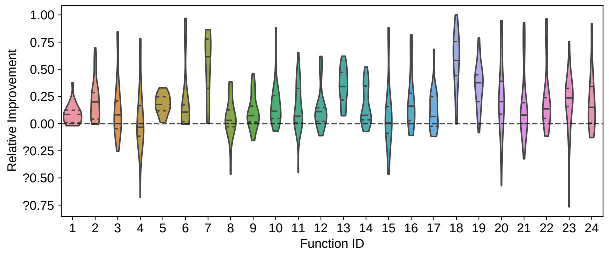

Figure 5: Distribution of relative improvement in ERT between the default and tuned hyperparameters across 30 modular CMA-ES variants for each of the 24 BBOB functions.. These variants were chosen in 3 groups: the 10 best according to ERT, those ranked 200-210 according to ERT and 10 commonly used variants [41]. Note that the variants are not necessarily identical for the different functions. The tuned hyperparameters were computed with MIP-EGO. All results are based on 250 independent runs for each (variant, hyperparameters)-pair.

thus confirming that results from Figure 2 can be generalized to the other BBOB-functions.

4.2.3 Scalability. The final, and most important, issue with the sequential methods lies in their scalability. Because these methods rely on complete enumeration of all variants based on their ERT, the required number of function evaluations grows as the algorithm space increases. If just a single new binary module is added, the size of this space doubles. This exponential growth is unsustainable for the sequential methods, especially if the testbed will also be expanded to include higher-dimensional functions (requiring more budget for the individual runs of the CMA-ES).

5 INTEGRATED METHODS

To tackle the issue of scalability, we propose a different way of combining algorithm selection and hyperparameter tuning. This is achieved by viewing the variant as part of the hyperparameter space, which is easily achieved by considering the module activations as hyperparameters. This leads to a mixed-integer search space, which both MIP-EGO and irace can easily adapt to. Thus, we will use two integrated approaches: MIP-EGO and irace. Both will get a total budget of 25,000 runs, which irace allocates dynamically while MIP-EGO allocates 25 runs to calculate ERTs for its solution candidates.

5.1 Case Study: F12

The viability of this integrated approach can be established by looking at a single function and comparing the results from the integrated approach to the previously established baselines. This is done for F12, since for this function, data for the top 50 variants is available, as shown in Figure 1. We run irace 4 times on instance 1 of this function, and compare the result to those achieved by the best tuned variants. This is done in Figure 6. From this figure, it can be seen that two of the runs from irace are very competitive with the best tuned variants, while the other two still manage to outperform most variants with default hyperparameters. This shows that this integrated approach is quite promising, and worth to study in more detail.

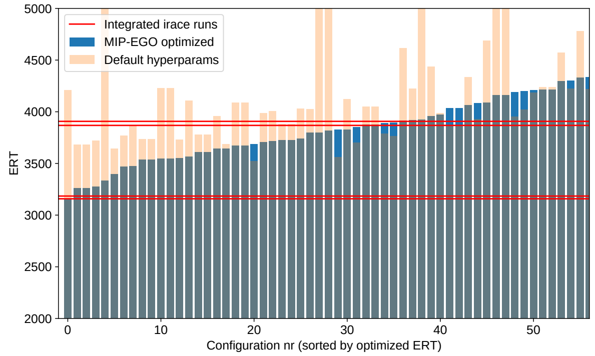

Figure 6: Comparison of ERTs achieved by the integrated approach using irace (straight horizontal red lines) and the ERTS of a set of 56 variants, both with default and tuned hyperparameters (using MIP-EGO). All ERTs are the result of 50 runs on instance 1 of F12.

5.2 Results

The results from running the integrated and sequential approaches on all 24 benchmark functions are shown in Figure 7. This figure shows that, in general, the ERTs achieved by irace and MIP-EGO are comparable. Irace has a slight advantage, beating MIP-EGO on 14 out of 24 functions. However, both methods still manage to outperform the naïve sequential approach while using significantly fewer runs, and are only slightly worse than the more robust version of the sequential approach. As expected, all methods manage to outperform pure algorithm selection quite significantly.

5.3 Comparison of MIP-EGO and Irace

From the results presented in Figure 7 it can be seen that the performance of the two integrated methods, MIP-EGO and irace, is quite similar. However, when introducing these methods, it was clear that their working principles differ significantly. To gain more understanding about how these results are achieved, three separate principles were studied: prediction error, balance between exploration and exploitation, and stability.

5.3.1 Prediction Error. The bars in Figure 7 seem to indicate that the prediction error for irace is smaller than the one for MIP-EGO. This is indeed the case: the average prediction error is 10.6% for irace, compared to 17.4% for MIP-EGO, suggesting that the AHT values reported by irace are more robust than the ERTs given by MIP-EGO. However, we also note that there exist some outliers, for which the prediction error of irace is relatively large (up to 35% for function 4). This happens because irace reports penalized AHT instead of ERT during the prediction-phase (see Definition 2.1). However, these prediction errors for irace can be positive (i.e. overestimating the real ERT), whereas MIP-EGO always underestimates the actual ERT.

5.3.2 Exploration-Exploitation Balance. While the prediction error is an important distinguishing factor between the two integrated methods, a much more important question to ask is how their search behaviour differs. This is best characterized by looking at the balance between exploration and exploitation, which we analyze

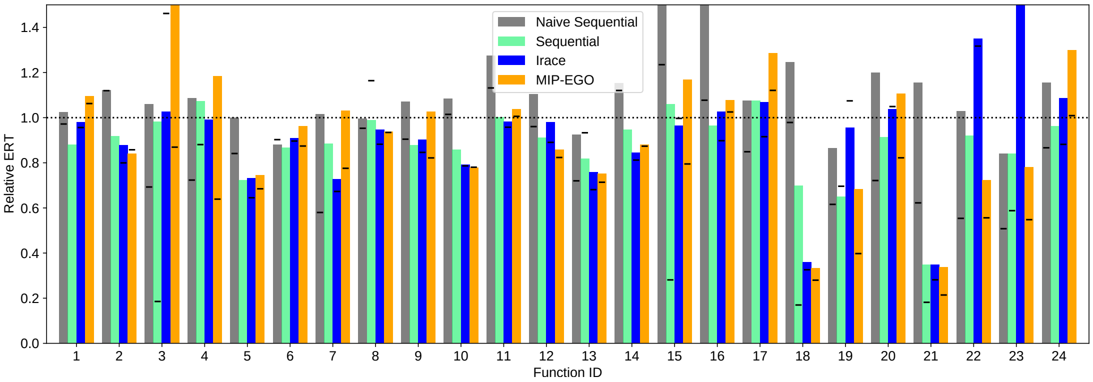

Figure 7: Relative ERTs against the best algorithm variant with default hyperparameters (targets chosen as in [44]) from running MIP-EGO and irace on the integrated selection and configuration space, as well as from the two sequential approaches. The 'predicted' relative ERT (based on 25 runs, with the exception of irace) is shown as a small black bar, whereas all other shown ERTs are based on 250 verification runs. y-axis cut at 1.5 (full data set available at [45]).

by looking at the complete set of evaluated candidate (variant, hyperparameter)-pairs, and noting how many unique variants were explored after the initialization phase. For MIP-EGO, this number is on average 565, while for irace it is only 112. This leads us to conclude that MIP-EGO is very exploitative in the algorithm space, while irace is more focused on exploitation of the hyperparameters.

On average, across all 24 functions, 78.6% of all candidates evaluated by irace differ only in terms of the continuous hyperparameters, whereas only 2.6% of the evaluated (variant, hyperparameter) pairs contain unique variants. Even when including the initial random population, this value only increases to 9.7%, while MIP-EGO achieves an average fraction of 77.8% unique variants evaluated.

This difference in exploration-exploitation balance is expected to lead to a difference in variants found by irace and MIP-EGO, specifically in how these variants would rank with default hyperparameters. This is visualized in Figure 8. From this figure, the differences between irace and MIP-EGO are quite clear. While MIP-EGO usually has better ranked variants, the median ranking is only 108, as opposed to 428 for irace. This confirms the findings of Section 3, where we saw there can be quite large differences in ranking before and after hyperparameter tuning. However, we still find that a larger focus on exploration yields a selection of variants which are ranked better on average.

5.3.3 Stability. Finally, we study the variance in performance of the algorithm variants found by the two configurators. Since MIP-EGO is more exploitative, it might be more prone to variance than irace and thus less stable over multiple runs. To investigate this assumption, we select two benchmark functions and run both integrated methods 15 times. The resulting (variant, hyperparameters)-pairs are then rerun 250 times, the runtime distributions of which are show in Figure 9. For F20, there is a relatively small difference between irace and MIP-EGO, slightly favoring irace. This indicates that the exploitation done by irace is indeed beneficial, leading to slightly lower hitting times. For F1, this effect is much larger, since for F1 most variants behave quite similarly, so the more benefit can be gained by tuning the continuous hyperparameters relative to exploring the algorithm space.

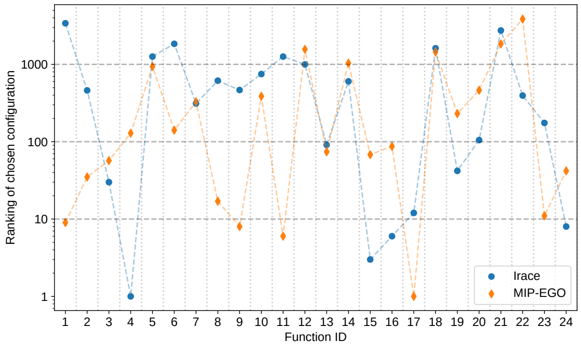

Figure 8: Original ranking (default hyperparameters) of the algorithm variant found by the integrated approaches (MIP-EGO and irace).

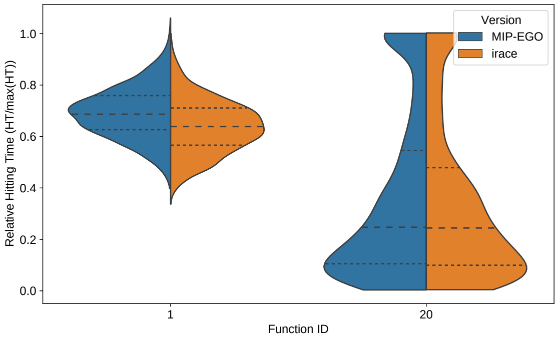

Figure 9: Distributions of relative hitting times (achieved hitting times divided by the maximal hitting time observed for the function) of 15 (variant, hyperparameters)-pairs, resulting from 15 independent runs of the integrated approaches, each of which is run 250 times on the corresponding benchmark function.

	Naïve Seq.	**Seq.**	**MIP-EGO**	**irace**
Best on # functions	0	9	9	6
Avg. Impr. over best modular CMA-ES	6.3%	24.7%	20.2%	20.7%
Avg. Impr. over default CMA-ES	67.4%	73.0%	72.9%	72.5%
Avg. Prediction Error	23.2%	18.8%	17.4%	10.6%
Budget (# function evaluations) / 1,000	~ 120	150	25	25
% Unique CMA-ES variants explored	95.8%	76.8%	77.8%	9.7%

Table 2: Comparison of the four methods for determining (variant, hyperparameter)-pairs used in this paper. Seq.=sequential. Improvement over best modular CMA-ES refers to the relative improvement in ERT over the single best variant with default hyperparameters.

5.3.4 Summary. A summary of the differences between the four methods studied in this paper can be seen in Table 2. From this, we can see that the differences in terms of performance between the integrated and sequential methods are minimal, while the integrated ones require a significantly lower budget. This budget value is in no way optimized, so an even lower budget than the one used in our study might achieve similar results. This might especially be true for irace, since it uses most of its budget to evaluate very small changes in hyperparameter values.

6 CONCLUSIONS AND FUTURE WORK

We have shown that the performance of CMA-ES variants is highly dependent on hyperparameter settings. Furthermore, we have shown that the impact of tuning these hyperparameter setting for different algorithm variants can differ significantly, highlighting the need for methods which combine *algorithm selection* and *algorithm configuration* of modular CMA-ES variants into a single integrated approach. We have shown that a sequential execution of brute-force algorithm selection and hyperparameter is sub-optimal because the large variance present in the observed ERTs. In addition, the sequential approaches require a large number of function evaluations, and quickly becomes prohibitive when new modules are added to the modEA framework. This clearly illustrates a need for efficient and robust combined algorithm selection and configuration methods.

We have shown that both irace and MIP-EGO manage to solve the CASH problem for the modular CMA-ES. They outperform the results from the naïve sequential approach and show comparable performance to the more robust sequential method, and this at much smaller cost (up to a factor of 6 in terms of function evaluations).

We have also observed that, for the integrated approach, MIP-EGO has a heavy focus on exploring the algorithm space, while irace spends most of its budget on tuning the continuous hyperparameters of a single variant. These differences were shown to lead to a slight benefit for irace on the sphere-function, but in general the difference in performance was minimal across the benchmark functions. This indicates that there is still room for improvement by combining the best parts from both methods into a single approach. This could take advantage of the dynamic allocation of runs to instances and adaptive capping which irace uses, as well as the efficient generation of new candidate solutions using the working principles of efficient global optimization, as done in MIP-EGO.

Another way to improve the viability of the integrated approaches studied in this paper would be to tune their maximum budget, as this was set arbitrarily in our experiments, and might be reduced significantly without leading to a large loss in performance.

We have focused in this work on the 3 hyperparameters selected in [7]. A straightforward extension of our work would be to consider the configuration of additional hyperparameters – global ones that are present in all variants (e.g., the population size), but also conditional ones that appear only in some variants but not in others (i.e. the threshold value when the 'threshold convergence' module is turned on). While irace can deal with such conditional parameter spaces, MIP-EGO would have to be revised for this purpose.

Our long-term goal is to develop methods which adjust variant selection and configuration *online*, i.e., while optimizing the problem. This could be achieved by building on exploratory landscape analysis [33] and using a landscape-aware selection mechanism. Relevant features could be local landscape features such as provided by flacco [24] (this is the approach taken in [7]), but also the state of the CMA-ES-parameters themselves, and approach suggested in [35]. We have analyzed the potential impact of such an online selection in [44]. Some initial work in determining how landscape features change during the search has been proposed in [21], but it was shown in [36] that some of the local features provided by flacco are not very robust, so that a suitable selection of features is needed for the use in a landscape-aware algorithm design.

Finally, we are interested in generalizing the integrated algorithm selection and configuration approach studied in this work to more general search spaces, and in particular to possibly much more unstructured algorithm selection problems. For example, one could consider to extend the CASH approach to the whole set of algorithms available in the BBOB repository (some of these are summarized in [18] but many more algorithms have been added in the last ten years since the writing of [18]). Note that it is an open question, though, how well the here-studied configuration tools irace and MIP-EGO would perform on such an unstructured, categorical algorithm selection space. Note also that here again we need to take care of conditional parameter spaces, since the algorithms in the BBOB data set have many different parameters that need to be set.

Acknowledgements. This work has been supported by the Paris Ile-de-France region.

REFERENCES

[1] Martin Andersson, Sunith Bandaru, Amos HC Ng, and Anna Syberfeldt. 2015. Parameter tuned CMA-ES on the CEC'15 expensive problems. In *Proc. of Congress on Evolutionary Computation (CEC'15)*. IEEE, 1950–1957.

[2] Anne Auger, Dimo Brockhoff, and Nikolaus Hansen. 2011. Mirrored Sampling in Evolution Strategies with Weighted Recombination. In *Proc. of Genetic and Evolutionary Computation (GECCO'11)*. ACM, 861–868. https://doi.org/10.1145/2001576.2001694

[3] Anne Auger and Nikolaus Hansen. 2005. A restart CMA evolution strategy with increasing population size. In *Proc. of Congress on Evolutionary Computation (CEC'05)*. 1769–1776. https://doi.org/10.1109/CEC.2005.1554902

[4] Anne Auger, Mohamed Jebalia, and Olivier Teytaud. 2005. Algorithms (X, sigma, eta): Quasi-random Mutations for Evolution Strategies. In *Artificial Evolution.*

Springer, 296–307. https://doi.org/10.1007/11740698_26

[5] Thomas Bäck, Christophe Foussette, and Peter Krause. 2013. *Contemporary Evolution Strategies.* Springer. https://doi.org/10.1007/978-3-642-40137-4_4

[6] Thomas Bartz-Beielstein. 2010. SPOT: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization. *CoRR* abs/1006.4645 (2010). arXiv:1006.4645 http://arxiv.org/abs/1006.4645

[7] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per instance algorithm configuration of CMA-ES with limited budget. In *Proc. of Genetic and Evolutionary Computation (GECCO’17).* ACM, 681–688. https://doi.org/10.1145/3071178.3071343

[8] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V. Arnold, and Tim Hohm. 2010. Mirrored Sampling and Sequential Selection for Evolution Strategies. In *PPSN.* Springer, 11–21. https://doi.org/10.1007/978-3-642-15844-5_2

[9] Leslie Pérez Cáceres, Manuel López-Ibáñez, Holger Hoos, and Thomas Stützle. 2017. An Experimental Study of Adaptive Capping in irace. In *LION*, Roberto Battiti, Dmitri E. Kvasov, and Yaroslav D. Sergeyev (Eds.). 235–250.

[10] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2018. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281* (Oct. 2018). arXiv:1810.05281 https://arxiv.org/abs/1810.05281

[11] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. 2019. Pitfalls and Best Practices in Algorithm Configuration. *J. Artif. Intell. Res.* 64 (2019), 861–893. https://doi.org/10.1613/jair.1.11420

[12] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3 (1999), 124–141.

[13] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proc. of International Conference on Machine Learning (ICML’18).* 1436–1445.

[14] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems.* 2962–2970.

[15] Nikolaus Hansen. 2008. CMA-ES with Two-Point Step-Size Adaptation. *arXiv:0805.0231 [cs]* (May 2008). http://arxiv.org/abs/0805.0231 arXiv: 0805.0231.

[16] Nikolaus Hansen. 2009. Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed. In *Proc. of Genetic and Evolutionary Computation (GECCO’09).* ACM, 2389–2396. https://doi.org/10.1145/1570256.1570333

[17] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. 2016. COCO: Performance Assessment. *arXiv:1605.03560 [cs]* (May 2016). http://arxiv.org/abs/1605.03560 arXiv: 1605.03560.

[18] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proc. of Genetic and Evolutionary Computation (GECCO’10).* ACM, 1689–1696.

[19] Nikolaus Hansen and Andreas Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *CEC.* 312–317. https://doi.org/10.1109/ICEC.1996.542381

[20] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *LION.* Springer, 507–523.

[21] Anja Janković and Carola Doerr. 2019. Adaptive landscape analysis. In *Proc. of Genetic and Evolutionary Computation (GECCO’19).* ACM, 2032–2035. https://doi.org/10.1145/3319619.3326905

[22] Grahame A. Jastrebski and Dirk V. Arnold. 2006. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *Proc. of Congress on Evolutionary Computation (CEC’06).* 2814–2821. https://doi.org/10.1109/CEC.2006.1688662

[23] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (2019), 3–45. https://doi.org/10.1162/evco_a_00242

[24] Pascal Kerschke and Heike Trautmann. 2016. The R-Package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In *Proc. of Congress on Evolutionary Computation (CEC’16).* IEEE, 5262–5269. https://doi.org/10.1109/CEC.2016.7748359

[25] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon R. Graham. 2019. Procrastinating with Confidence: Near-Optimal, Anytime, Adaptive Algorithm Configuration. In *Proc. of Neural Information Processing Systems (NeurIPS’19).* 8881–8891.

[26] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *The Journal of Machine Learning Research* 18, 1 (2017), 826–830.

[27] Johannes Lengler. 2018. A General Dichotomy of Evolutionary Algorithms on Monotone Functions. In *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*, Vol. 11102. Springer, 3–15. https://doi.org/10.1007/978-3-319-99259-4_1

[28] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560* (2016).

[29] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz (Eds.). 2007. *Parameter Setting in Evolutionary Algorithms.* Studies in Computational Intelligence, Vol. 54. Springer.

[30] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43 – 58. https://doi.org/10.1016/j.orp.2016.09.002

[31] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. 2011. *The irace package, Iterated Race for Automatic Algorithm Configuration.* Technical Report TR/IRIDIA/2011-004. IRIDIA, Université Libre de Bruxelles, Belgium. http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

[32] Manuel López-Ibáñez and Leslie Pérez Cáceres. [n. d.]. The irace Package: Iterated Race for Automatic Algorithm Configuration. http://iridia.ulb.ac.be/irace/..

[33] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory landscape analysis. In *Proc. of Genetic and Evolutionary Computation (GECCO’11).* ACM, 829–836.

[34] Alejandro Piad-Morffis, Suilan Estévez-Velarde, Antonio Bolufé-Röhler, James Montgomery, and Stephen Chen. 2015. Evolution strategies with thresheld convergence. In *Proc. of Congress on Evolutionary Computation (CEC’15).* 2097–2104. https://doi.org/10.1109/CEC.2015.7257143

[35] Zbynek Pitra, Jakub Repický, and Martin Holena. 2019. Landscape analysis of gaussian process surrogates for the covariance matrix adaptation evolution strategy. In *Proc. of Genetic and Evolutionary Computation (GECCO’19).* ACM, 691–699. https://doi.org/10.1145/3321707.3321861

[36] Quentin Renau, Johann Dreo, Carola Doerr, and Benjamin Doerr. 2019. Expressiveness and Robustness of Landscape Features. In *Proc. of Genetic and Evolutionary Computation (GECCO’19).* ACM, 2048–2051. https://doi.org/10.1145/3319619.3326913

[37] John R. Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15 (1976), 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[38] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD.* ACM, 847–855.

[39] Sander van Rijn. 2018. Modular CMA-ES framework from [41], v0.3.0. https://github.com/sjvrijn/ModEA. Pypi package: https://pypi.org/project/ModEA/0.3.0/.

[40] Sander van Rijn, Carola Doerr, and Thomas Bäck. 2018. Towards an Adaptive CMA-ES Configurator. In *PPSN (Lecture Notes in Computer Science)*, Vol. 11101. Springer, 54–65. https://doi.org/10.1007/978-3-319-99253-2_5

[41] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. 2016. Evolving the structure of Evolution Strategies. In *SSCI.* 1–8. https://doi.org/10.1109/SSCI.2016.7850138

[42] Sander van Rijn, Hao Wang, Bas van Stein, and Thomas Bäck. 2017. Algorithm Configuration Data Mining for CMA Evolution Strategies. In *Proc. of Genetic and Evolutionary Computation (GECCO’17).* ACM, 737–744. https://doi.org/10.1145/3071178.3071205

[43] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Github repository for Online Selection of CMA-ES Variants. (2019). https://github.com/Dvermetten/Online_CMA-ES_Selection.

[44] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Online selection of CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation (GECCO’19).* ACM, 951–959. https://doi.org/10.1145/3321707.3321803

[45] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. 2020. Github repository with the data used in this paper. https://github.com/Dvermetten/CMA_ES_hyperparam.

[46] Hao Wang, Michael Emmerich, and Thomas Bäck. 2014. Mirrored Orthogonal Sampling with Pairwise Selection in Evolution Strategies. In *SAC.* ACM, 154–156. https://doi.org/10.1145/2554850.2555089

[47] Hao Wang, Michael Emmerich, and Thomas Bäck. 2018. Cooling Strategies for the Moment-Generating Function in Bayesian Global Optimization. In *Proc. of Congress on Evolutionary Computation (CEC’18).* https://doi.org/10.1109/CEC.2018.8477956

[48] Hao Wang, Bas van Stein, Michael Emmerich, and Thomas Bäck. 2017. A new acquisition function for Bayesian optimization based on the moment-generating function. In *SMC.* IEEE, 507–512.

[49] David H. Wolpert and William G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation* 1, 1 (1997), 67–82. https://doi.org/10.1109/4235.585893

[50] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth AAAI Conference on Artificial Intelligence.*

[51] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI).*