Tuning as a Means of Assessing the Benefits of New Ideas in Interplay with Existing Algorithmic Modules

Jacob de Nobel
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

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

Hao Wang
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

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

ABSTRACT

Introducing new algorithmic ideas is a key part of the continuous improvement of existing optimization algorithms. However, when introducing a new component into an existing algorithm, assessing its potential benefits is a challenging task. Often, the component is added to a default implementation of the underlying algorithm and compared against a limited set of other variants. This assessment ignores any potential interplay with other algorithmic ideas that share the same base algorithm, which is critical in understanding the exact contributions being made. We explore a more extensive procedure, which uses hyperparameter tuning as a means of assessing the benefits of new algorithmic components. This allows for a more robust analysis by not only focusing on the impact on performance, but also by investigating how this performance is achieved. We implement our suggestion in the context of the Modular CMA-ES framework, which was redesigned and extended to include some new modules and several new options for existing modules, mostly focused on the step-size adaptation method. Our analysis highlights the differences between these new modules, and identifies the situations in which they have the largest contribution.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; **Bio-inspired optimization**.

ACM Reference Format:
Jacob de Nobel, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2021. Tuning as a Means of Assessing the Benefits of New Ideas in Interplay with Existing Algorithmic Modules. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3449726.3463167

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463167

1 INTRODUCTION

With the continuous increase in interest for the field of optimization, many new algorithms get introduced every year. A large number of these algorithms are not completely novel, but instead add new algorithmic ideas to existing methods. Originally referring to one particular algorithm, CMA-ES has developed into a whole family of algorithms that are built around the core design of the original CMA-ES algorithm from [22]. While this growth of the algorithm set helps to keep improving the state-of-the-art performance, it also raises a simple question: "How to assess the benefits of new algorithmic ideas?"

The naive way of performing such an assessment is to implement the algorithmic idea into a bare-bones version of the base algorithm, and to benchmark it against the default (and maybe some other variants). While this technique does manage to give an indication of the usefulness of the newly introduced component, the results are not always practical and hide important information, since they only consider the idea in isolation. Often, there tends to be an important interplay between algorithmic components, which is completely missed when doing the type of assessment described above.

We aim to provide in this work a roadmap for assessing these algorithmic ideas in a way which takes component interactions into account. This is achieved by considering the different algorithmic ideas as modules in a modular framework. Several of these types of frameworks have been developed over the years [10, 33, 35, 36, 43]. In this work, we present a use case for this roadmap by using the Modular CMA-ES (ModCMA), which is extended from the existing ModEA framework [36], by both adding new modules and new options for existing modules (see Section 2 for details). With this modular framework, we show in this work how hyperparameter tuning can be used to assess the contributions of the newly implemented components.

We illustrate how this approach gives a detailed perspective on the benefits of new algorithmic ideas, by not only looking at pure performance metrics, but also considering the interplay with existing modules. We show, among other things, that the introduction of new Step-Size Adaptation (SSA) methods can be beneficial, but that it requires careful consideration of the interactions with other modules, such as the choice of recombination weights. We also discuss the limitations of this approach, and how to best use it to gain the most understanding about these new algorithmic ideas.

2 REDESIGNING MODEA TO A MODULAR CMA-ES FRAMEWORK

Our work relies heavily on the Modular Evolutionary Algorithms (ModEA) framework introduced in [36]. Since this framework hasn't undergone any active development in recent time, we decided to redesign the framework to our specifications. The modifications we made rendered the name of the framework no longer befitting, as only CMA-ES variants can now be created using the framework, whereas the original framework also supported the design of other evolutionary algorithms. The new framework was dubbed the Modular CMA-ES (ModCMA) and is available as an open source Python package within the IOHprofiler [14] environment[1]. It is integrated with the IOHexperimenter, giving access to a broad set of benchmark problems, including a C++ implementation of the BBOB functions [21] from the COCO environment [20]. In addition, this allows for easy data logging, which can be used directly with the interactive performance analysis and visualization from IOHanalyzer [41].

Motivation. The primary goal behind redesigning the framework was to reduce its complexity, and to only include functionality compatible to the CMA-ES and its variants. The reasoning behind this is the fact that the framework mostly revolves around the CMA-ES. Other EAs are available in the framework, but are quite underdeveloped w.r.t. the CMA-ES. Moreover, introducing working interactions between the CMA-ES and operators from other EAs overly complicates the framework's structure. For example, ModEA contains a range of different methods for performing recombination. However, the canonical CMA-ES does not explicitly perform recombination. Instead, it updates its mean m by taking a weighted average of the individuals in its current population, which it then uses to sample new individuals from a normal distribution. In other EAs, recombination occurs in a much more pronounced sense, for example by crossover. In order to make the modular algorithm of the CMA-ES function with these other forms of recombination, its original method for "recombination" had to be adapted. The CMA-ES however, is still only able to properly function with one of these recombination methods, the canonical one. As this pattern could be observed in other parts of the framework as well (i.e., mutation, selection), it was decided to remove these other methods all together and to focus solely on the CMA-ES.

2.1 The Modular CMA-ES

To design the Modular CMA-ES, we use the implementation from the popular CMA-ES tutorial [19] as a starting point. This work provides a detailed description of the CMA-ES algorithm, including a practical guide to its implementation. From this basic design, we separate the CMA-ES in a number of functionally related blocks, in order to allow a customization of a specific part of the algorithm. This allows us to implement algorithmic variants of the CMA-ES as functional modules. From a user perspective, any of these modules could then be combined in order create a custom instantiation of the CMA-ES, by selecting an option for each available module.

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

We implemented a new module for performing boundary correction (see Section 2.2), and added five alternative options for performing step size adaptation (see Section 2.3). These two extensions to the framework will be the focus of our analysis through out this work. This set of changes give us the following list of modules for the redesigned Modular CMA-ES:

(1) **Active Update**: Bad candidate solutions are penalized in the covariance matrix update using negative weights [24]. Note that in [19], this is given as the default version, here we consider it to be optional.
(2) **Elitism**: $(\mu + \lambda)$ - selection instead of (μ, λ) - selection.
(3) **Orthogonal Sampling**: All the newly sampled points in the population are orthonormalized using a Gram-Schmidt procedure [39].
(4) **Sequential Selection**: Candidate solution are immediately ranked and compared with the current best solution. If improvement is found, no additional objective function evaluations are performed [11].
(5) **Threshold Convergence**: A method for balancing exploration with exploitation, scaling the mutation vectors to a required length threshold, which decays over time [34].
(6) **Step-Size Adaptation**: Supplementary to the default Cumulative Step size Adaptation (CSA), Two Point step size Adaption (TPA) [17] is implemented. TPA requires two additional objective function evaluations, used for evaluating both a shorter and a longer version of the population's center of mass. The version which shows the higher objective function value determines whether the step size should be increased or decreased. Five newly added mechanism for performing step size adaptation are implemented. They are described in detail in Section 2.3.
(7) **Mirrored Sampling**: For every newly sampled point, its mirror image is added the population, by reversing its sign [3]. With *Pairwise Selection*, only the best point of each mirrored pair is used in recombination.
(8) **Quasi-Gaussian Sampling**: Instead of performing the simple random sampling from the multivariate Gaussian, new solutions can alternatively be drawn from quasi-random sequences (a.k.a. low-discrepancy sequences) [6]. We implemented two options for this module, the Halton and Sobol sequences.
(9) **Recombination Weights**: Three options are implemented; 1) default weights (see [19]), 2) equal weights: $w_i = 1/\mu$, and 3) $w_i = 1/2^i + 1/(\lambda 2^\lambda)$ for $i = 1, 2, \ldots, \lambda$.
(10) **Restart Strategy**: When the optimization process stagnates, the CMA-ES can be restarted using a restart strategy. Two strategies are implemented. IPOP [5] increases the size population after every restart by a constant factor. BIPOP [18] also changes the size of the population, but alternates between larger and smaller population sizes.

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

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

Table 1: The modules available for the Modular CMA-ES. The numeric index for each module corresponds to the index used in the text of Section 2.1. Newly added modules/options are given in bold.

(11) **Boundary Correction**: If candidate solutions are sampled outside the search domain, they can be transformed back into the search domain by applying a boundary correction operation. In Section 2.2, we describe six options for performing boundary correction which have been implemented.

In Table 1, an overview is given of all currently implemented modules and their options in the Modular CMA-ES framework.

2.2 Boundary Correction

In the original framework, a boundary correction function taken from [28] was implemented, and always applied after each mutation. In some cases, however, this operator can degrade the performance of the algorithm quite drastically. We therefore decided to make the boundary correction optional, and to implement it as a module, for it to only be used when beneficial. A number of different boundary correction strategies were implemented, taken from [12]:

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

2.3 Step-Size Adaptation

In this work, we consider a number of alternative step size adaptation mechanisms for new options for the Modular CMA-ES. We take inspiration from [25], which provides a qualitative evaluation of multiple step size adaptation mechanisms used in ES. In addition the CSA and TPA step size adaptation methods, which were already implemented, we implemented the following procedures:

(1) **Median success rule (MSR)** [1]: The MSR mechanism adapts the step-size σ as follows: it firstly computes a success rate by checking the number of current individuals that are better than some user-defined quantile of the function values in the previous population, then accumulates such success rates in every iteration, and finally decides to increase the step-size if the cumulated values is bigger than 1/2 and decrease it otherwise.
(2) **Population success rule (PSR)** [32]: determines the success rate of the current population using a rank-based approach. It firstly sorts all individuals in the current and previous population together, then retrieves the set of ranks of individuals belonging to the current iteration and the one for the previous iteration, and finally calculates the average rank difference between those two sets as the population success rate, which controls the step-size updates.
(3) **xNES step-size adaptation (xNES)**[16, 26, 42]: calculates the length of each standardized mutation vector and subtracts from it the expected length of the standard Gaussian vector. The resulting difference is then scalarized using the same weights used in the recombination, which is finally fed into an exponential function to generate a multiplicative coefficient to modify the step-size.
(4) **mean-xNES step-size adaptation (m-XNES)** [26]: functions similarly to xNES, with the exception that it takes the standardized differential vector between current center of mass and the one in the previous iteration and compares it to the expected length of the standard Gaussian vector.
(5) **xNES with log normal prior Step size adaptation (p-xNES)** [26]: resembles the principle of self-adaptation for step-sizes, where λ trial step-sizes are generated from a log-normal distribution which takes the current step-size as its mean and each trial step-size is used to sample a candidate point. To determine the new step-size, this method calculates the weighted sum of the log-transformed trial step-sizes, where those assigned to their corresponding candidate points in the recombination.

3 INCREMENTAL ASSESSMENT OF MODULE PERFORMANCE

With the introduction of these new module settings, we have a clear use-case for the assessment of algorithmic ideas within the CMA-ES algorithm. Since these options are implemented into a framework with many existing modules, it will not suffice to look at them in isolation. Instead, we should carefully consider the potential interactions with the existing modules and investigate their impact on the empirical performance of ModCMA. Previous work [37] used data from a complete enumeration of all module settings to analyze the contribution of each individual module. However, such an approach becomes intractable when we are confronted with a huge set of algorithmic variants, or more importantly if we aim to obtain the contribution of some new modules implemented incrementally to an existing portfolio of algorithms, which we have investigated previously. In addition, this complete enumeration approach ignores entirely the configuration of continuous strategy parameters,

e.g., c_1, c_μ, and c_c, which have been shown to significantly impact the per-instance performance of the resulting configurations [8].

To properly address the problem of determining the contribution of a single module setting to an existing portfolio of modules, we make use of hyperparameter optimization, which has previously been shown to achieve results comparable to the complete enumeration method, while being much more easily extendable to other hyperparameters [38]. We propose the following roadmap to formalize this procedure, which is designed to be generic, so that it can function with any modular algorithm, hyperparameter tuner, and performance metric:

(1) Select a modular implementation of the base algorithm to which the new module has been added, a hyperparameter optimizer and a performance metric.
(2) Collect a list of the existing modules and relevant hyperparameters (without the new module to assess). This will be the search space for the hyperparameter optimization.
(3) Run the selected hyperparameter optimizer on this search space, ideally for a wide set of relevant benchmark functions. This data will then serve as the baseline performance.
(4) Extend the original search space by including the new module to assess, and run the hyperparameter optimization on this extended search space (using the exact same setup as the baseline).
(5) Compare the data from the baseline to the experiment with the extended search space. This should not only be done from a performance perspective, but also from the resulting configurations themselves. This allows for the analysis of potential interactions between modules.

3.1 Performance Measure

Assuming a set of optimization algorithms $\mathcal{A} = \{A_1, A_2, \ldots\}$, a set of objective functions $\mathcal{F} = \{f_1, f_2, \ldots\}$, a function evaluation budget B, and N repeated runs of each algorithm, we denote by $T(A, f, v, i)$, $i \in [1..N]$, the number of function evaluations (hitting time) consumed by algorithm A to find in its i-th run on function f a solution of solution quality at least v. We consider the target values $\mathcal{V} = \{10^{\frac{10-i}{5}} : i \in [1..51]\} \subset [10^{-8}, 10^2]$, and adopt a performance measure which aggregates the hitting times of each of these targets; the Area Under the ECDF Curve (AUC):

$$\mathrm{AUC}(A, f, \mathcal{V}) = \int_1^B \widehat{F}(t; A, f, \mathcal{V}) \mathrm{d}t,$$

$$\widehat{F}(t; A, f, \mathcal{V}) = \frac{1}{N|\mathcal{V}|} \sum_{v \in \mathcal{V}} \sum_{i=1}^{N} \mathbb{1}(T(A, f, v, i) \leq t),$$

where $\mathbb{1}$ is the characteristic function. We note that most hyperparameter tuning methods are built with minimization in mind. As such, we use the Area Over the Curve (AOC) instead of AUC, since we know $\mathrm{AOC}(A, f, \mathcal{V}) = B - \mathrm{AUC}(A, f, \mathcal{V})$.

In order to collect the AOC measure from the runs of the ModCMA, we integrated it into the IOHprofiler [14], which provides ease-of-use logging functionality required to calculate the AOC of each run.

3.2 Experimental Overview

In this paper, we use the irace [29, 30] hyperparameter optimizer. Irace[2] is based on the principle of iterated racing, in where each race[3] repeatedly executes several configurations until there is a statistically significant reason to discard enough of them to move to the next race (thus inherently allocating more runs to more promising configurations).

Four runs of irace are performed for each of the 24 objective functions in the BBOB single objective noiseless problem suite [20, 21], of which the first function instance is used in 5D. Each run of Irace is given a budget of 1 000 algorithm evaluations, which themselves have a budget of $10\,000 \cdot D$ function evaluations. We use the AOC attained by a run of a given configuration as the objective function value for irace. Irace will designate one or more configurations as elites, which are the best configurations found by irace. We validate the performance of these elite configurations by performing 25 validation runs, with the same random seeds for all configurations. We use the results of these runs to assess the final performance.

Conform our roadmap, we define a baseline by tuning the existing modules from ModCMA, which are shown (plain text) in Table 1. In addition, we tune four continuous hyperparameters c_1, c_μ, c_c, and c_σ, which control the dynamics of the adaption of the covariance matrix (c_1, c_μ, and c_c) and of the step-size (c_σ).

We compare two experiments to our baseline where in addition to the existing modules, 1) a number of new SSA methods (see Section 2.3) are included, and 2) a new boundary correction module (see Section 2.2) is added to the tuned parameters [4]. Both of these experiments are using the same experimental setup as the baseline experiment (excluding the tuned parameters). Note that in the boundary correction experiment, the new SSA methods cannot be selected and vice versa.

3.3 Single Module Performance

Before considering our proposed method, we run a basic benchmarking experiment on each of the individual module options (including the new options). This is similar to the common approach of benchmarking a new module against a set of other algorithm variants. We show the resulting best single-module configurations (a.k.a. the virtual best solver, VBS for short) relative to the default CMA-ES in Table 2. In this table, we see that among the new modules, only two have been selected: MSR for F23 and m-XNES for F5. We can further look at the over-all contributions of the newly introduced step-size settings by plotting the ECDF-curves over all functions, as done in Figure 1. In this figure, we can clearly see that most methods are quite competitive, with the only exception being xNES, which has a overall worse performance than the others. Overall, the MSR method seems to be quite effective, but there is no strict domination over the other settings.

[2] Implemented in R, freely available at [31].
[3] The initial iteration of irace consists of random configurations and the default CMA-ES setting.
[4] All of the code used in these experiments, and the resulting data, is available in [13]

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

Table 2: Table showing the AOC of the best single-module configuration for each function (VBS), compared to that of the default CMA-ES. The name of the solver corresponds to the module which is active, e.g. <module_name>_<option_value>. Note that these values does not include benefits from tuning the continuous hyperparameters, which are set to the default values for all configurations in this table.

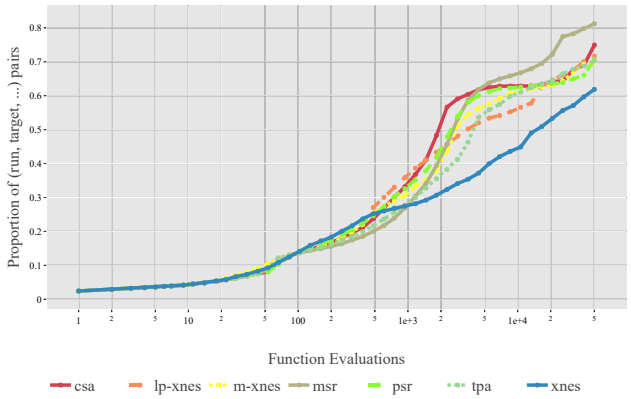

Figure 1: ECDF-curve of all single-module stepsize options. Figure generated using IOHanalyzer [41].

4 ANALYSIS AND RESULTS

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

4.1 Baseline

As mentioned in Section 3.2, we conduct a baseline tuning experiment. Since we run 4 runs of irace for each function, this results in 4 sets of elites (each set has up to 5 configurations), for which we then perform the verification runs. We plot the distribution of

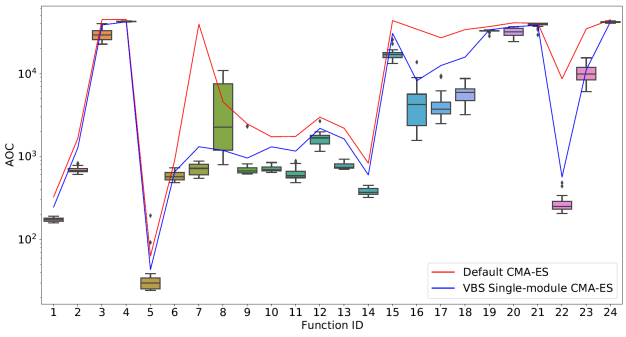

Figure 2: Distribution of the area over the ECDF curve for the final elite configuration of the baseline irace runs. All AOC's are averages of 25 verification runs. The VBS single-module configurations can be seen in Table 2.

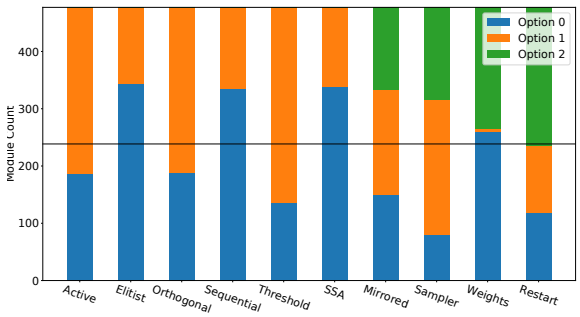

Figure 3: Module counts of all elites found in the baseline-experiment, over all 24 BBOB-functions. The option numbers correspond to those in Table 1

the AOC for each of these configurations in Figure 2, in addition to this, the AOC of the default CMA-ES and the VBS is shown. From this figure, it is clear that the tuning of all parameters at once is much better than simply selecting a single-module variant, as is to be expected. This plot also highlights the variance in performance of the final found configurations. There are two main reasons for this fact: the inherent stochasticity of the CMA-ES itself, and the large impact of the initially generated configurations of irace. We discuss these challenges in detail in Section 5.

From this baseline data, we can also study the resulting configurations themselves. This can be done by aggregating the modules which have been selected in the final elite configurations in the separate irace runs, as is visualized in Figure 3. In this figure, we can see that there is a large variability in the selected module options, which seems to indicate that they are all usable for at least some functions. One notable exception is the weights option "equal", which is chosen in less than 1% of configurations.

4.2 Performance analysis

First, we visualize the distributions of the AOC of the single best configuration found in each run of irace (based on the verification runs) in Figure 4. In this plot, we can see that the effect of introducing the new modules is quite mixed. For some functions,

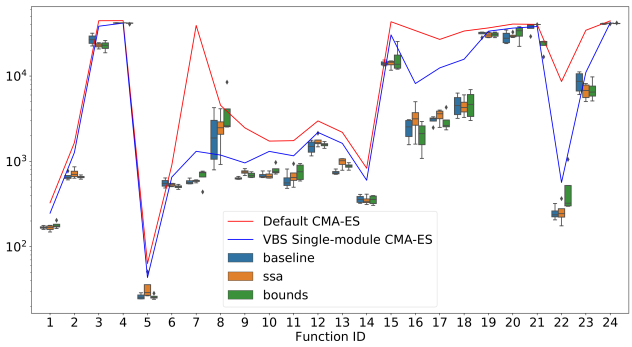

Figure 4: Distribution of the single best elites from the baseline and the tuning with the additional modules. AOC values are the result of averaging over 25 verification runs.

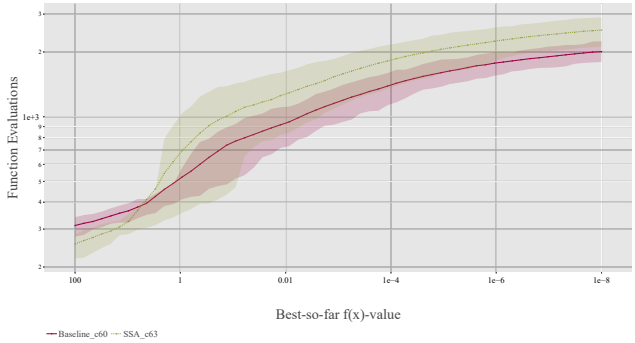

Figure 5: Comparison of the Expected Running Time of the best configurations found on F12 by both the baseline and the SSA experiments. Shaded areas indicate the outer quantiles (20-80).

performance decreases (e.g., on F8) after introducing new modules, while for others we see the desired improvement (e.g. on F23).

In order to better show these differences, we show in Figure 6 the AOC of the single best configurations found in both the SSA and bound-experiments relative to the best configuration from the baseline. Here we observe a generally negative trend, with outliers in both directions. This seems to indicate that these new modules are not always beneficial to the final performance. For example, we can consider F12, where the configuration found by the baseline has an average AOC of 1 159, while the best configuration found when including the new SSA-methods in the search space reaches an average AOC of 1 480. We show the expected running time of these two configurations in Figure 5, where we can clearly observe this difference. However, we can observe a large variance between runs, which can partly explain poor performance. Indeed, if we look at the average AOC as found during the irace run (instead of the later verification runs), the difference between these two configurations is only 7%, even though the distance between them in the verification runs is much larger. This leads to an important observation about the assessment of the new algorithmic modules: when judging results purely from the average performance measures, it is necessary to also consider the overall variability of the experiment, as well as the inherent stochasticity of the base algorithm.

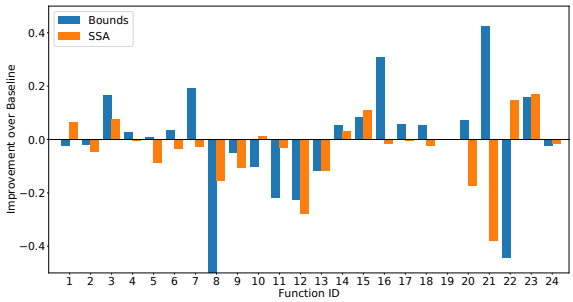

Figure 6: The relative improvements per function of the best configuration found by irace relative to the baseline experiment's best configuration.

We perform the same procedure for the boundary correction methods. The impact of this module is expected to be smaller, since for most of the "easier" functions, the boundary condition is rarely violated. For some of the more challenging functions however, the penalty value given by BBOB function itself might not be sufficient to "guide" the algorithm back in bounds, but an explicit boundary correction could be beneficial in these cases. We can see that this seems to indeed be the case in Figure 6, where on the more complex functions, e.g., F21, the performance is improved when the boundary correction module is tuned.

In Figure 6, we also see that the inclusion of the new SSA methods manages to improve the overall performance for some functions. As an example, on F23 we saw an improvement of 17.1% over the best baseline configuration. If we consider all four elite configurations and compare the average performance differences, the average improvement is even higher, at 22.3%. The stability of this improvement is promising, but in order to fully grasp how the inclusion of the new SSA mechanisms leads to this improvement, we need to analyze the selected modules across these different experiments.

4.3 Module Analysis

We have seen that the performance of the elite configuration found on F23 improves when we include the new SSA modules in the search space. In order to identify what this performance can tell us about the new modules themselves, we should study the configurations in more detail. The obvious way to see the difference is by looking at how often the new module options have been selected in the final elite configurations. Over 20 elites, the PSR update was selected 14 times, MSR once, and CSA five times. This shows that these new modules are indeed used in the successful configurations. To see how the inclusion of these module options changes the interactions with the other modules, we look at the combined module activation plot, which is shown in Figure 7. From this figure, we can see that there are some interesting differences between the two sets of configurations: the options for the restart and mirrored module are not as uniform when using the new SSA methods, and the weights option is changed completely. These observations show that there is a clear interplay between these modules.

In addition to module activation, we can look into the distributions of configured continuous hyperparameters. To illustrate this, we study F3, and show the pairwise relations between the four continuous hyperparameters and the final AOC value in Figure 11.

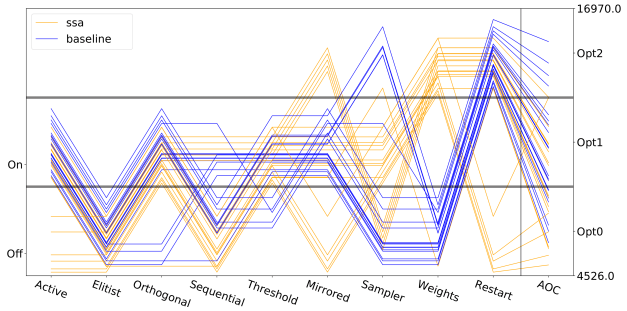

Figure 7: Combined module activation plot for the elites found in the baseline and SSA experiments, for function 23. The lower the line, the better its performance, scaled within each band according to the AOC. The option numbers correspond to those in Table 1.

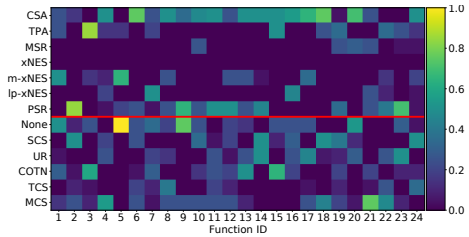

Figure 8: Heatmap showing the fraction of the elite configuration in which each of the options for either SSA (top) or boundary correction (bottom) are active.

From the marginal distribution (shown on the diagonal), we can see that the optimized setting of c_σ differs the most across the SSA, boundary correction, and the baseline experiments. This is a direct result of the introduction of the new SSA methods, each of which prefers slightly different setting for this parameter. This indicates that even though the final performance of the elite configurations is similar between the baseline and the SSA-experiment, the inclusion of new SSA methods clearly changes the selected configurations.

We can extend this module analysis to all functions by aggregating the most important differences found between the baseline and SSA-experiments. First, we can plot how often each new module option is selected in the elites for each function, as is done in Figure 8. We can use the same principle to study the interaction with the other modules. For the binary modules, we can directly capture the module difference by looking at which modules occur more or less often in the final set of elites, as is visualized in Figure 9. While this does not directly generalize to modules with more settings, we can create a similar plot for the other modules by considering the overlap in selected module distributions. This is visualized in Figure 10. From these figures, it becomes clear that the elites on some functions are barely affected by the inclusion of the new modules, while others require completely different module settings to properly exploit the changed search space.

The cost of tuning. We should note that only considering the final elite configurations does not tell the full story of a module's contribution. As noted previously, introducing a new module increases the size and complexity of the search space, which has a large impact on the hyperparameter tuning task. If a module is very dependent on the settings of other hyperparameters, this can lead to deterioration of the final results, since the initially sampled configurations are likely to have worse performance than those in the baseline. This is visualized in Figure 12, where this is clearly seen on function F5. This is a linear slope function, but the BBOB-specification does not include a sufficient penalty for leaving the search space. As a result, an algorithm which quickly leaves the search space will reach the required objective value very quickly.

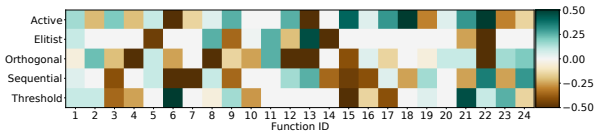

Figure 9: Heatmap showing difference in the fraction of the elite configuration in which each the of the binary modules are active, between the baseline and the SSA experiment. Positive values indicate a module is turned on more often in the SSA experiments.

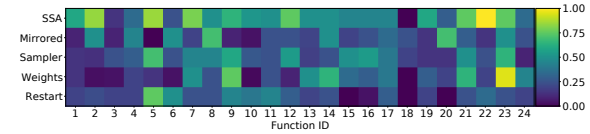

Figure 10: Heatmap showing difference in the distribution of the ternary modules selected in the final elites, between the baseline and the SSA experiment. 0 indicates that the distribution is identical, while 1 indicates that there is no overlap at all.

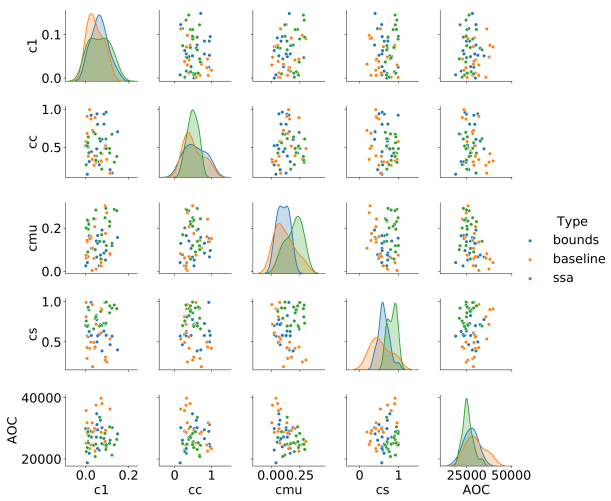

Figure 11: Distribution of the continuous hyperparameters from the elite configurations found in all three experiments.

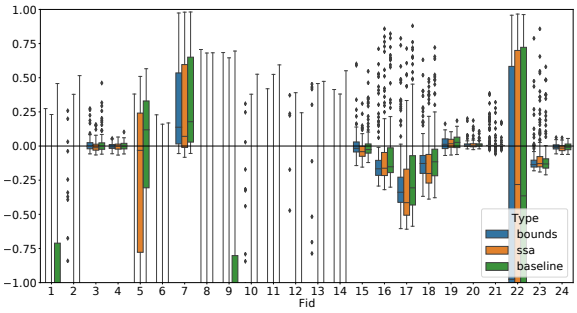

Figure 12: Distribution of the relative AOC values found in the initial race of irace (relative to the default CMA-ES configuration; positive values equate to lower AOC.)

Thus, when adding boundary correction methods, $\frac{5}{6}$ random configurations are not able to abuse this loophole, leading to a worse initial performance. While for F5, the function is simple enough that the good configurations can still be found (and the inclusion of the default CMA-ES settings in the initial population means that there is always at least one good configuration present), the same issue exists to a lesser extent in other functions. Figure 12 also shows that the "tunability" of modules on different functions varies widely. For instance, on functions F16 - F18, the spread of AOC values is significantly larger than those on functions F19 - F21, suggesting that it is relatively more difficult to tune the modules in the latter since the tuner will very likely take a considerably larger budget to identify optimal configurations. Also, while on some functions it is trivial to get improvement (e.g., F7) over the default CMA-ES, it is a lot more challenging on others, for example on functions F16 - F18.

5 CHALLENGES

We discuss three key challenges for the module assessment procedure based on hyperparameter optimization as identified in this work.

Influence and stochasticity of the hyperparameter tuning: While we showed that assessing the impact of an algorithmic component by using a hyperparameter tuning approach provides useful insights, there are several factors which can complicate this approach. Since hyperparameter tuning is a very challenging problem, with many different approaches to solving it, the kind of tuner used will have a large impact on the resulting assessment [38]. In this paper, we used irace, which tends to focus on converging to a single configuration, instead of covering a large set of different solutions. This necessitates running multiple repetitions of the irace procedure itself, as the initialization might otherwise have too much impact on the final configurations. This can quickly become computationally expensive.

Algorithm-inherent stochasticity: As we discussed in the results, we need to take care when drawing conclusions from the performance of the different CMA-ES configurations. Since CMA-ES is inherently stochastic, the amount of variance of the configurations on a certain function has a large impact on the search procedure of irace. Since we end up selecting elites based on the average performance, we are inherently underestimating the AOC of the final configuration. Even though irace largely mitigates this by using statistical testing in the races to decide when to discard configurations, there will always be some degree of underestimation of the performance (the median performance in the verification runs is 3.4% worse than predicted from the irace runs).

Limits of the per-instance analysis: In the current setup, the performance assessment is done on an per-instance basis. While this can be preferred over tuning for large sets of functions/instances [9], it does have some drawbacks. Specifically, if a module is designed to have a good performance over a wide set of functions, but other settings exist for each individual function which outperform it, this new module would not be seen as beneficial. Because of this, we argue that module assessment by hyperparameter tuning should not replace the traditional assessments, but rather complement it for more in-depth, per-instance analysis.

6 DISCUSSION AND FUTURE WORK

We introduced a roadmap for assessing the performance of individual algorithmic ideas, which takes into account the interplay with other existing settings by comparing the results of hyperparameter tuning. Since this approach requires a modular design to function as intended, we use the Modular CMA-ES framework, which we extended with new modules. Our analysis showed that the newly added step size adaptation mechanisms are not always useful, but do provide clear benefits in several functions. The results also showed that SSA is most useful when combined with a different weights option.

The current version of the Modular CMA-ES framework is a good step in the direction of complete modularization of the CMA-ES algorithm, but some further enhancements can still be made. This would allow for even more precise control over each of the individual components, leading to an ideal testbed for new algorithmic ideas, which can then be evaluated using the approach outlined in this paper. However, since this can be computationally intensive, we should aim to share and reuse data as much as possible, by developing and maintaining a well-organized repository for this type of benchmark data. This does not only reduce the amount of computation needed to test new modules, but it also gives rise to the possibility of testing methods to re-use data from other experiments. Ideally, this would allow for the usage of methods from transfer learning to significantly shorten the time needed to assess a module's performance, even within a large modular search space.

Additionally, we note that while the proposed module assessment is inherently dependent on the used hyperparameter tuning method, the overall procedure remains the same no matter which tuner is used. As a result, the analysis of the results should take into account the particularities of the tuner, such as the way configurations are generated. Further research should still be done into different hyperparameter optimization methods (e.g., SMAC [23], MIP-EGO [40], SPOT [7], GGA [2], hyperband [27], etc.). Moreover, one could further investigate a module's contribution to the portfolio by developing a credit assignment scheme, e.g. using Shapley values [15]. Additionally, an analysis pipeline for this type of benchmarking could be designed within existing tools like the IOHanalyzer [41], which would greatly reduce the amount of effort needed to assess new algorithmic ideas.

REFERENCES

[1] Ouassim Ait Elhara, Anne Auger, and Nikolaus Hansen. 2013. A Median Success Rule for Non-Elitist Evolution Strategies: Study of Feasibility. *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference* (07 2013). https://doi.org/10.1145/2463372.2463429

[2] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. 2015. Model-based Genetic Algorithms for Algorithm Configuration. In *Proc. of International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 733–739.

[3] Anne Auger, Dimo Brockhoff, and Nikolaus Hansen. 2011. Mirrored Sampling in Evolution Strategies with Weighted Recombination. In *GECCO*. ACM, 861–868. https://doi.org/10.1145/2001576.2001694

[4] Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Tea Tušar, and Konstantinos Varelas. 2020. Data from BBOB-workshops and competitions on 24 noiseless functions. https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob.

[5] 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

[6] Anne Auger, Mohammed 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

[7] 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

[8] 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

[9] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per instance algorithm configuration of CMA-ES with limited budget. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, Peter A. N. Bosman (Ed.). ACM, 681–688. https://doi.org/10.1145/3071178.3071343

[10] Mauro Birattari, Luis Paquete, and Thomas Stützle. 2003. Classification of Metaheuristics and Design of Experiments for the Analysis of Components. https://www.researchgate.net/publication/2557723_Classification_of_Metaheuristics_and_Design_of_Experiments_for_the_Analysis_of_Components. Technical report.

[11] 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

[12] Fabio Caraffini, Anna V. Kononova, and David Corne. 2019. Infeasibility and structural bias in differential evolution. *Inf. Sci.* 496 (2019), 161–179. https://doi.org/10.1016/j.ins.2019.05.019

[13] Jacob de Nobel, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2021. *Data and Code from: Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules*. https://doi.org/10.5281/zenodo.4524959

[14] 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 The BBOB datasets from [4] are available in the web-based interface of IOHanalyzer at http://iohprofiler.liacs.nl/.

[15] Alexandre Fréchette, Lars Kotthoff, Tomasz P. Michalak, Talal Rahwan, Holger H. Hoos, and Kevin Leyton-Brown. 2016. Using the Shapley Value to Analyze Algorithm Portfolios. In *Proc. of the AAAI Conference on Artificial Intelligence*. AAAI Press, 3397–3403. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12495

[16] Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. 2010. Exponential Natural Evolution Strategies. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (Portland, Oregon, USA) *(GECCO '10)*. Association for Computing Machinery, New York, NY, USA, 393–400. https://doi.org/10.1145/1830483.1830557

[17] 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.

[18] Nikolaus Hansen. 2009. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (Montreal, Québec, Canada) *(GECCO '09)*. Association for Computing Machinery, New York, NY, USA, 2389–2396. https://doi.org/10.1145/1570256.1570333

[19] Nikolaus Hansen. 2016. The CMA Evolution Strategy: A Tutorial. *CoRR* abs/1604.00772 (2016). arXiv:1604.00772 http://arxiv.org/abs/1604.00772

[20] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2020. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software* (2020), 1–31.

[21] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Technical Report RR-6829. INRIA. https://hal.inria.fr/inria-00362633/document

[22] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

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

[24] Grahame A. Jastrebski and Dirk V. Arnold. 2006. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *CEC*. 2814–2821. https://doi.org/10.1109/CEC.2006.1688662

[25] Oswin Krause, Tobias Glasmachers, and Christian Igel. 2017. Qualitative and Quantitative Assessment of Step Size Adaptation Rules. In *Proceedings of the 14th ACM/SIGEVO Conference on Foundations of Genetic Algorithms* (Copenhagen, Denmark) *(FOGA '17)*. Association for Computing Machinery, New York, NY, USA, 139–148. https://doi.org/10.1145/3040718.3040725

[26] Oswin Krause, Tobias Glasmachers, and Christian Igel. 2017. Qualitative and Quantitative Assessment of Step Size Adaptation Rules. In *Proceedings of the 14th ACM/SIGEVO Conference on Foundations of Genetic Algorithms* (Copenhagen, Denmark) *(FOGA '17)*. Association for Computing Machinery, New York, NY, USA, 139–148. https://doi.org/10.1145/3040718.3040725

[27] 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).

[28] Rui Li. 2009. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Theses. Leiden University.

[29] 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

[30] 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

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

[32] Ilya Loshchilov. 2014. A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization. arXiv:1404.5520 [cs.NE]

[33] Nuno Lourenço, Francisco Pereira, and Ernesto Costa. 2012. Evolving Evolutionary Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*. ACM, New York, NY, USA, 51–58. https://doi.org/10.1145/2330784.2330794 bibtex: lourenco_evolving_2012.

[34] Alejandro Piad-Morffis, Suilan Estévez-Velarde, Antonio Bolufé-Röhler, James Montgomery, and Stephen Chen. 2015. Evolution strategies with thresheld convergence. In *CEC*. 2097–2104. https://doi.org/10.1109/CEC.2015.7257143

[35] Jorge Tavares, Penousal Machado, Amílcar Cardoso, Francisco B. Pereira, and Ernesto Costa. 2004. On the Evolution of Evolutionary Algorithms. In *Genetic Programming (Lecture Notes in Computer Science)*, Maarten Keijzer, Una-May O'Reilly, Simon Lucas, Ernesto Costa, and Terence Soule (Eds.). Springer, 389–398. https://doi.org/10.1007/978-3-540-24650-3_37

[36] 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

[37] Sander van Rijn, Hao Wang, Bas van Stein, and Thomas Bäck. 2017. Algorithm Configuration Data Mining for CMA Evolution Strategies. In *GECCO*. ACM, 737–744. https://doi.org/10.1145/3071178.3071205

[38] Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2020. Integrated vs. sequential approaches for selecting and tuning CMA-ES variants. In *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, Carlos Artemio Coello Coello (Ed.). ACM, 903–912. https://doi.org/10.1145/3377930.3389831

[39] 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

[40] Hao Wang, Michael Emmerich, and Thomas Bäck. 2018. Cooling Strategies for the Moment-Generating Function in Bayesian Global Optimization. In *CEC*. 1–8. https://doi.org/10.1109/CEC.2018.8477956

[41] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. 2020. IOHanalyzer: Performance Analysis for Iterative Optimization Heuristic. *CoRR* abs/2007.03953 (2020). https://arxiv.org/abs/2007.03953 IOHanalyzer is available at CRAN, on GitHub, and as web-based GUI, see https://iohprofiler.github.io/IOHanalyzer/ for links.

[42] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. 2008. Natural Evolution Strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. 3381–3387. https://doi.org/10.1109/CEC.2008.4631255

[43] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2012. Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors. In *Proc. of Theory and Applications of Satisfiability Testing (SAT'12) (Lecture Notes in Computer Science, Vol. 7317)*. Springer, 228–241. https://doi.org/10.1007/978-3-642-31612-8_18