Analyzing the Impact of Undersampling on the Benchmarking and Configuration of Evolutionary Algorithms

Diederick Vermetten, Hao Wang, Manuel López-Ibañez, Carola Doerr, Thomas Bäck

Analyzing the Impact of Undersampling on the Benchmarking and Configuration of Evolutionary Algorithms

Diederick Vermetten
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands
d.l.vermetten@liacs.leidenuniv.nl

Hao Wang
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands
h.wang@liacs.leidenuniv.nl

Manuel López-Ibañez
School of Computer Science and Engineering, University of Málaga
Málaga, Spain
manuel.lopez-ibanez@uma.es

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France
Carola.Doerr@lip6.fr

Thomas Bäck
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands
t.h.w.back@liacs.leidenuniv.nl

ABSTRACT

The stochastic nature of iterative optimization heuristics leads to inherently noisy performance measurements. Since these measurements are often gathered once and then used repeatedly, the number of collected samples will have a significant impact on the reliability of algorithm comparisons. We show that care should be taken when making decisions based on limited data. Particularly, we show that the number of runs used in many benchmarking studies, e.g., the default value of 15 suggested by the COCO environment, can be insufficient to reliably rank algorithms on well-known numerical optimization benchmarks.

Additionally, methods for automated algorithm configuration are sensitive to insufficient sample sizes. This may result in the configurator choosing a "lucky" but poor-performing configuration despite exploring better ones. We show that relying on mean performance values, as many configurators do, can require a large number of runs to provide accurate comparisons between the considered configurations. Common statistical tests can greatly improve the situation in most cases but not always. We show examples of performance losses of more than 20%, even when using statistical races to dynamically adjust the number of runs, as done by irace. Our results underline the importance of appropriately considering the statistical distribution of performance values.

CCS CONCEPTS

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

KEYWORDS

Parameter tuning, algorithm configuration, performance measures, evolution strategies

1 INTRODUCTION

The study of iterative optimization heuristics is a continuously developing area within computer science. With the ever expanding number of newly developed algorithms, the importance of proper benchmarking has been gaining more traction [1]. Standardized benchmarking environments have been proposed for a wide variety of problem types [5, 8, 18] and are widely used to compare the performance of different optimizers. However, since most state-of-the-art optimizers are stochastic in nature, assessing their performance is not necessarily a straightforward task, as any selected measure for performance is inherently the result of a limited sampling of some underlying probability distribution.

In order to get reliable estimates for the distributions of these performance measures, benchmarking pipelines generally recommend to perform multiple optimization runs on each problem. For example, the popular COCO environment [8] recommends measuring the performance of 15 independent runs. When comparing algorithms, these individual performance measures are aggregated into a single number per function, using a variety of different measures ranging from taking the mean of hitting times, to Expected Running Time (ERT) or anytime performance metrics such as the area under the Empirical Cumulative Distribution Function (ECDF).

Aggregated performance values calculated from a limited number of runs may poorly estimate the true expected performance of an algorithm, especially when the distribution of individual performance values is non-normal or shows a large variance. It is clear that measuring only one run will lead to many mistakes when comparing a set of algorithms in an algorithm selection or configuration context, but it is not clear whether 15, 50 or more runs are sufficient to alleviate this issue.

In addition to comparing aggregated values such as the mean, statistical tests are often used to decide whether one algorithm outperforms another. Widely-used examples are the parametric t-test, when distributions are assumed to be somewhat normal, or the non-parametric Wilcoxon rank-sum test, when this assumption cannot be made. Although the hypothesis tested by the Wilcoxon rank-sum test refers to the relative ranking of independent samples and cannot be used to conclude anything about mean values, this distinction is often ignored in the literature when drawing conclusions from it.

The problem of comparing algorithms based on a potentially small number of samples is not limited to benchmarking studies,

Please cite as: Diederick Vermetten, Hao Wang, Manuel López-Ibañez, Carola Doerr, and Thomas Bäck. 2022. Analyzing the Impact of Undersampling on the Benchmarking and Configuration of Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference (GECCO '22)*, July 9–13, 2022, Boston, MA, USA. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528799.

but also applies to meta-optimization, such as algorithm configuration [11], also known as automated tuning, or algorithm selection [13]. When faced with an algorithm configuration problem, the configurator has to make a tradeoff between collecting more samples from promising configurations to gain confidence in their performance and exploring a larger variety of configurations. Most commonly, these samples are then compared based on their mean or using statistical tests to determine which configurations are returned as the best-found.

Previous work [21] has shown that configuration methods are susceptible to large performance variance, which may lead to a potential loss of performance with respect to the true best configurations explored.

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

2 PRELIMINARIES

2.1 Performance Measures

We consider the minimization of functions of the form:

$$f : X \subset \mathbb{R}^d \to \mathbb{R}$$

where X is the *search space* and d denotes its dimensionality. We focus on iterative optimization heuristics (IOHs), and in particular randomized IOHs. IOHs optimize f by a sequential process of generating solution candidates $x^1, \ldots, x^\lambda \in X$, evaluating their quality $f(x^1), \ldots, f(x^\lambda)$, and adjusting their strategy to generate the next candidates. For randomized IOHs, both the *number* of candidates that are generated in a given iteration and the *strategy* to generate them can be stochastic.

While there are a large number of metrics that could be considered to measure the performance of these algorithms, we limit ourselves to *Area Under the ECDF Curve (AUC)*. AUC is an anytime performance measure defined as follows.

Definition 2.1 (Area Under the ECDF Curve, AUC). For a given optimization algorithm A with a budget of B function evaluations for minimizing a function $f : X \to \mathbb{R}$ and a given finite set of targets $\mathcal{V} \subset \mathbb{R}$, the AUC value of A on f is approximated by

$$\mathrm{AUC}(A, f, \mathcal{V}) = \int_1^B \widehat{F}(t; A, f, \mathcal{V}) \mathrm{d}t \ , \tag{1}$$

where

$$\widehat{F}(t; A, f, \mathcal{V}) = \frac{1}{N|\mathcal{V}|} \sum_{\phi \in \mathcal{V}} \sum_{i=1}^{N} \mathbb{1}(t_i(A, f, \phi) \leq t) \ , \tag{2}$$

and N is the number of (ideally independent) runs for which we have performance logs, and $\mathbb{1}(t_i(A, f, \phi) \leq t)$ is an indicator function that returns 1 if the first hitting time of target ϕ in run i of A is not larger than t. If the target ϕ is not hit in this run, the indicator always returns 0.

AUC is a measure that should be maximized, however, most automated algorithm configuration methods are designed for minimization. Thus, we use the *Area Over the Curve (AOC)* instead of AUC. Since the values $\widehat{F}(t; A, f, \mathcal{V})$ are normalized between 0 and 1, the AUC value is a real number between 0 and B. We can hence define $\mathrm{AOC}(A, f, \mathcal{V}) = B - \mathrm{AUC}(A, f, \mathcal{V})$. For our experiments, we set the set of targets $\mathcal{V}$ to be the default as used in the COCO environment (51 targets, logarithmically spaced between 10^2 and 10^{-8}).

2.2 Benchmark Functions

For this study, we make use of the single-objective, noiseless functions from COCO's BBOB suite [8], which is considered to be one of the most popular sets of benchmark functions for benchmarking continuous derivative-free black-box optimization algorithms. For our experiments, we use the 5-dimensional versions of these functions.

Benchmarking studies using BBOB commonly perform each individual run on a different "instance" of each function, where each instance is generated by applying transformations in both the domain and objective space, in such a way that the core properties of the function are preserved [9]. In this work we aim to avoid the additional variance caused by multiple instances of each function, and thus focus on a single instance (instance ID 1) of each of the 24 BBOB functions.

2.3 Algorithm Configuration: Irace

As part of this work, we investigate the impact of performance variability on algorithm configuration. To perform algorithm configuration, we use irace [16], which is based on an iterated racing procedure. Irace starts out by randomly generating a set of parameter configurations using uniform sampling in the defined parameter space. Then, the first race starts, where for each of these configurations, a number of runs (*FirstTest*) are performed, after which a statistical test (either t-test or Friedman test) determines which configurations to continue with. The surviving configurations are run again a certain number of times (*EachTest*, set to 1 in this paper). Then the test is performed again to potentially eliminate additional configurations, and runs keep being added in this way until no more than a given number (5 in this paper) of configurations remain (elites)

or the budget assigned to this race is exhausted. The elite configurations are then used as the basis for generating new candidates to be evaluated in the next race, until the overall budget for irace is fully used. This procedure eventually leads to 5 or fewer surviving elite configurations, from which we select the one with the lowest mean as the final recommendation.

This racing approach is however not the only technique used by algorithm configurators to select the best configurations from a larger set. Some algorithm configuration methods, such as Hyperband [14], use a successive halving (SHA) approach [12]. In the first step of SHA, *FirstTest* runs are performed for each of the n initial configurations. Then, given a reduction factor R, the $\lceil n/R \rceil$ configurations with the lowest mean survive, and the others are discarded.

For the surviving configurations, an additional $2 \cdot \mathit{FirstTest}$ runs are performed. This process of keeping the best $1/R$ fraction based on mean, and doubling the number of additional runs, is repeated until only one configuration survives.

2.4 Modular CMA-ES

To study the impact of performance variability on algorithm configuration, we make use of the Modular CMA-ES (modCMA) framework [4, 19]. This framework provides an implementation of the popular CMA-ES [10] algorithm, with a wide variety of different modifications that can be activated independently from each other. In this work, we consider a hyper-parameter space consisting of 10 discrete modules and 4 continuous hyper-parameters. This search space corresponds to the baseline used to analyze the modular CMA-ES in [4], and we thus base our initial analysis on the same data [3] to avoid needlessly re-running irace. The data we use consists of irace runs on each of the 24 BBOB functions. Here, we use 1 of these irace runs for each function. Then, for each of the configurations sampled during these irace runs, we collect 200 new independent runs, which we refer to as *verification runs*. Since the irace runs we use typically generated over 200 configurations each, this means we have collected at least $24 \cdot 200 \cdot 200 = 960\,000$ runs that we analyze in this work. We also use this data to simulate independent races.

Throughout this paper, we will refer to two sets of configurations: the 33 configurations generated during the first race of irace are sampled uniformly at random in the configuration space, and are thus referred to as **random modCMA configurations**. Since we often want to consider the impact of the later parts of the tuning, we also consider the 33 last generated configurations. This set of configurations will be referred to as **high-quality modCMA configurations**.

3 WHY 15 RUNS ARE NOT ENOUGH

While it is clear that any aggregated performance measure used to compare randomized algorithms is an empirical estimation of their true performance, the variance of this estimation is not necessarily equal for all algorithms on all functions. However, for a practical benchmarking setup, this nuance is often ignored in favour of simpler guidelines, such as aggregating a fixed number of *samples* (i.e., individual performance values from independent runs) for each algorithm on each function. The usual recommendation of 15 samples [8] is often enough to make clear decisions on simple uni-modal functions, but the situation is much less clear on more challenging optimization problems.

We illustrate the significant variation in performance between runs by showing in Figure 1 the distribution of 15 independent AOC-values for a wide variety of algorithms from the BBOB-repository (https://numbbo.github.io/data-archive/bbob/) on F21 in 5D. This figure also shows that the normality assumption, commonly taken for granted in benchmarking studies, is not well supported by the apparent distribution of the 15 performance values shown for each algorithm.

For some algorithms, the performance distributions even appear to show signs of bi-modality. As such, any analyses made based on this set of samples should be treated with care. While this large amount of variance is very pronounced in F21, it is not limited to this function, as other functions display similar effects but to a slightly lesser extent.[1]

The impact of performance variability can potentially be even larger when considering the task of algorithm configuration. It has previously been observed that the performance of an algorithm configuration on

verification runs can differ significantly from the runs performed during the configuration task [4].

We illustrate this effect by showing in Figure 2 the changes in the ranking of the 33 **high-quality modCMA configurations** described in Section 2.4 when calculating mean performance using a small sample size (15) and a larger number of verification runs (200) on F21.

While this might be considered a rather extreme case, it is by no means the only scenario in which behaviours like this can occur. Since algorithm configuration often generates similarly performing configurations near the end of a configuration run (while exploiting promising regions), making decisions about which configuration to select might become very noisy when using relatively low sample sizes. This phenomenon is exemplified in Figure 3, where we show the evolution of the mean AOC of 3 selected **high-quality modCMA configurations** relative to an incremental number of AOC values. Each horizontal line of the same color corresponds to the cumulative mean of a sequence of values sampled with replacement from the same 200 AUC values. Despite sampling from the same 200 AUC values, the variance of the means of 15 and 25 samples is quite large and those means often poorly estimate the true mean performance.

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

In Figure 4, we show the distribution of AOC values for each configuration sampled during a run of irace on each of the 24 BBOB functions. The performance of each configuration is based on the mean of 200 verification runs, and the plot shows the relative performance loss to the best of these means. The (up to five) elite configurations returned by irace are marked with a red triangle. The lowest of these elites corresponds to the level of performance loss achieved by irace compared to the best-performing configuration sampled during the configuration process. From this figure, it can be seen that for some of the more complex functions, a 10% performance loss or more can occur, clearly demonstrating that the variability of performance can severely hinder the outcome of the configuration efforts.

4 IMPACT ON BENCHMARKING

To simulate a common algorithm comparison scenario, we make use of the set of 33 **high-quality modCMA configurations** from

[1] Plots equivalent to Figure 1 for other functions and performance measures are available on our figshare repository [23].

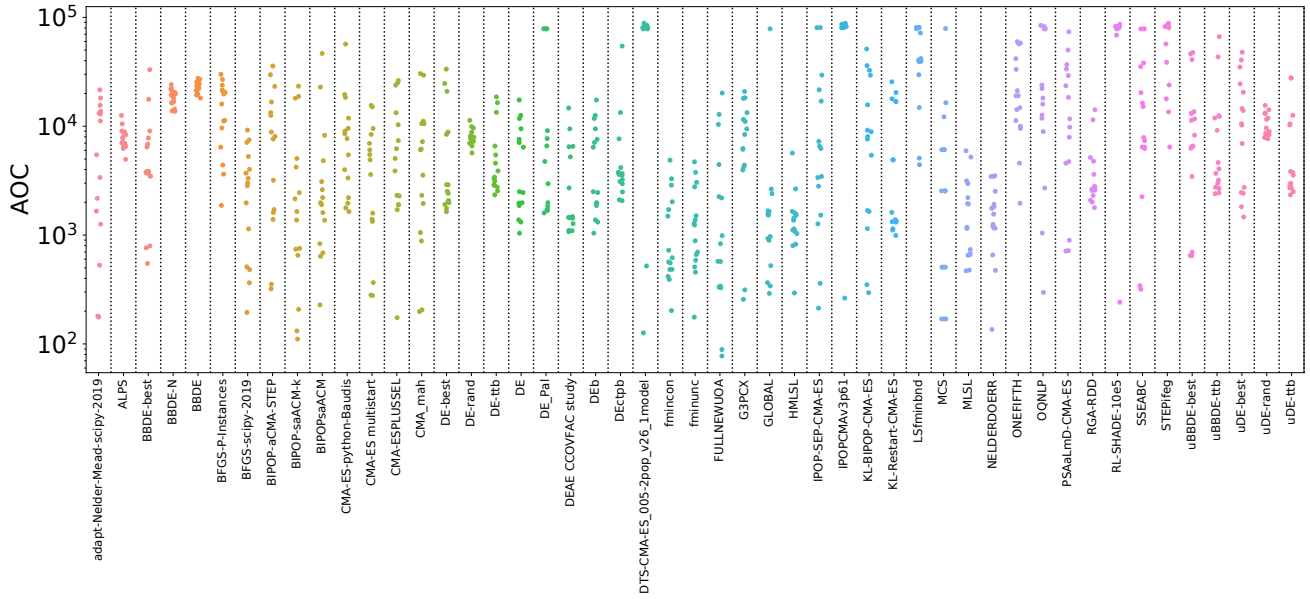


Figure 1: Distribution of the AOC values of 15 independent runs of available BBOB algorithms on F21 in 5D.

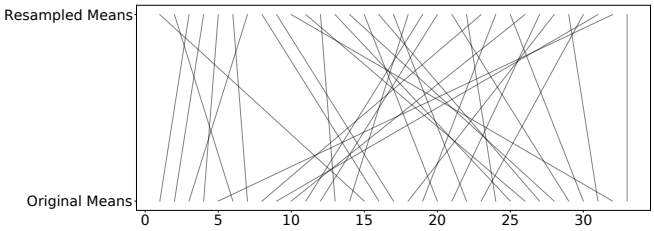


Figure 2: Changes in ranking of 33 random modCMA configurations based on calculating the mean over a sample of 15 AOC values (*Resampled Means*) versus the 200 verification run samples (*Original Means*), on F21.

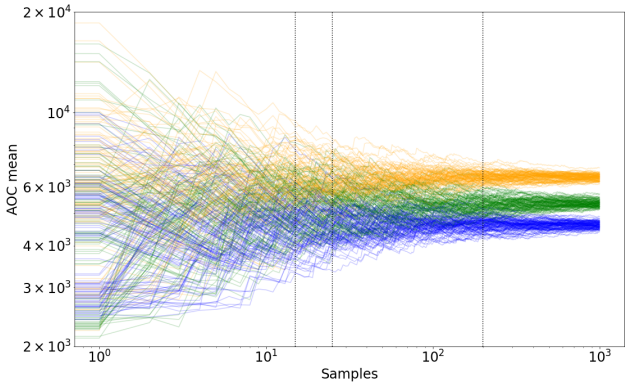


Figure 3: Evolution of the cumulative mean over sample sizes of 3 selected high-quality modCMA configurations on F18. The vertical lines indicate sample sizes 15, 25 and 200 respectively. Means are based on sampling with replacement from the original 200 samples of each configuration.

Section 2.4 and simulate the benchmarking procedure by randomly re-sampling with replacement AOC values, for sample sizes 2, 5, 10, 15, 25 and 50 from the set of 200. Then, we select the configuration with the best mean for each particular sample size as the winner, and compare its true performance (i.e., over 200 runs) to that of the actual best configuration to get an estimate for the performance loss. This process is repeated 5000 times for each sample size and each function, and the resulting performance loss per function is shown in Figure 5. We conclude that using means to determine the best-performing algorithm is not always reliable, and can lead to selecting configurations that are clearly sub-optimal. Many benchmarking studies use non-parametric statistical tests to assess significant differences without assuming normality, yet they still rely on comparison of means to rank the algorithms. While we can see that increasing the used sample size is always beneficial, even using as many as 50 samples can still see performance losses of 10% and more on some functions.

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

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

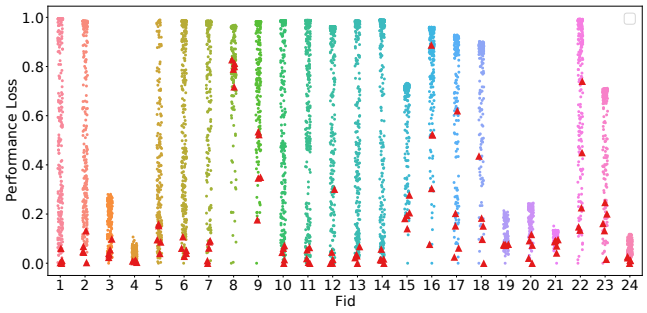


Figure 4: Performance losses between all modCMA configurations explored during the execution of one irace run on each function, and the best one from this set of configurations. The final elites of each irace run are marked in larger red triangles. All datapoints shown are based on 200 independent samples.

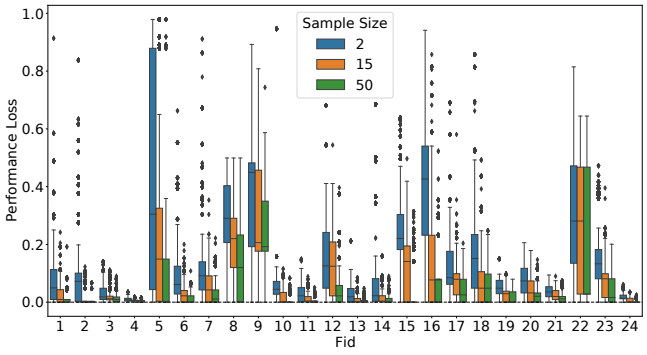


Figure 5: Performance loss, relative to the configuration with the best mean calculated over 200 samples, when comparing 33 high-quality modCMA configurations based on mean calculated from different number of samples. Each bar represents 5000 repetitions of the experiment.

better, than the true mean. We plot in Figure 6 the underestimation error for **high-quality modCMA configurations**.

We observe large underestimation error in almost all functions. In some functions, such as F8, the underestimation error is large even for a sample size of 50. We also notice that large underestimation errors in Figure 6 often coincide with a large performance loss seen in Figure 5. This observation can be explained by looking in more detail at the performance distribution of the used configurations on a particular function, as is done in Figure 7 for F8.

We see in this figure that all configurations have a fraction of runs where the AOC value is very large, indicating that these were very poorly performing runs. When calculating the mean value of a configuration from a limited number of samples, if none of these poor runs appears in the samples, then the mean of the configuration will be lower than its true mean, leading to the large underestimation seen in Figure 6.

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

Another common way in which the mean is used in benchmarking is in the basic pairwise comparison scenario, where two algorithms are directly compared to each other. To investigate this scenario, we simulate pairwise comparisons based on a limited sample size, and correlate the decisions made by the pairwise comparison to the difference in true means between the selected configurations. To achieve this simulation, we use the full set of modCMA configurations generated during an irace run, which totals over 200 configurations on each function. From this set of configurations, we take 10 000 pairs, drawn uniformly at random, to perform the pairwise comparison. Then, for each pair of configurations, we sample a number of AOC values from the 200 values available, calculate the sample means and compare them to decide which configuration is the best. The comparison is *correct* if it gives the same conclusion as comparing the true means. We repeat the sampling and comparison step 500 times to calculate the fraction of times that the comparison is correct. The results of this experiment on F9, F15, and F22, with sample size 15, are displayed in Figure 8.

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

5 STATISTICAL TESTING

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

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

In Figure 9, we show the fraction of configuration pairs where the amount of incorrect tests exceeds the used level of statistical significance ($\alpha = 0.05$). From this figure we see that, while the t-test seems to work well enough for most functions, it is not ideal on all

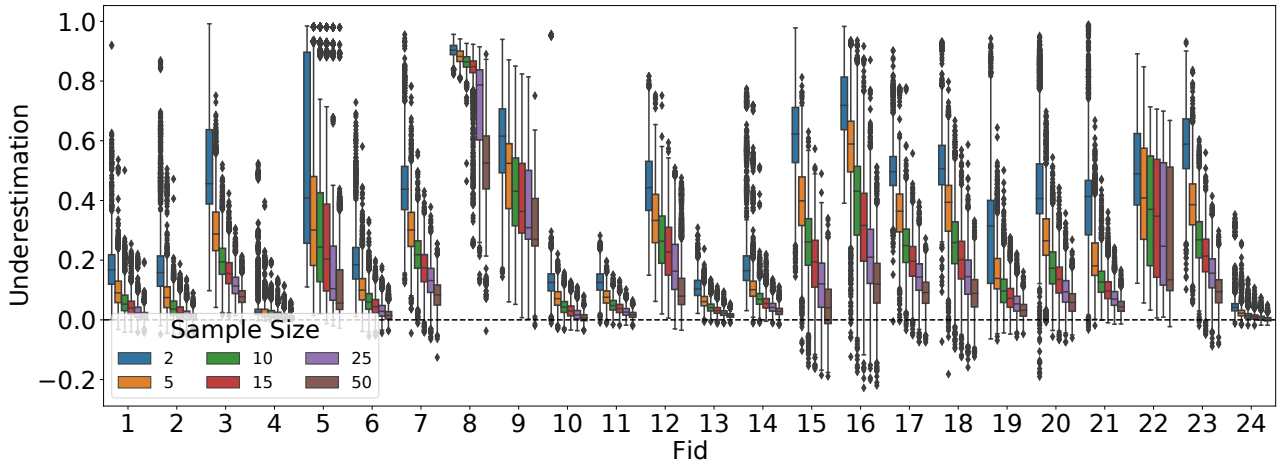


Figure 6: Underestimation when comparing 33 high-quality modCMA configurations based on mean calculated from different number of samples. Each bar represents 5000 repetitions of the experiment.

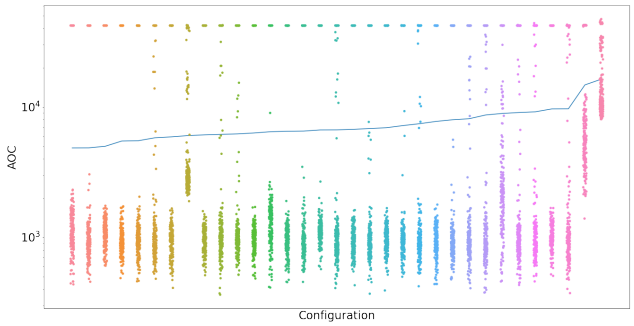


Figure 7: Distribution of AOC values of 200 individual runs of high-quality modCMA configurations on F8. The line indicates the mean AOC value of each configuration, and is the basis for the sorting on the x-axis.

functions, which seemingly indicates that the normality assumption is not met.

We zoom in on function F9 in Figure 10, and look at the difference between making decisions based on means, t-test and Wilcoxon rank-sum test. We note that both statistical tests show an error rate that is larger than α for pairs of configurations with a difference in means up to 60%. We also note that even though the t-test is less frequently incorrect, it is also more frequently inconclusive compared to the Wilcoxon rank-sum test, even for configuration whose means differ significantly.

Inconclusiveness is not a factor when comparing based on means, but that comes with the cost of making more incorrect decisions as well. While the number of incorrect decisions decreases when adding more samples, the overall observations for the three comparison procedures remain similar [23].

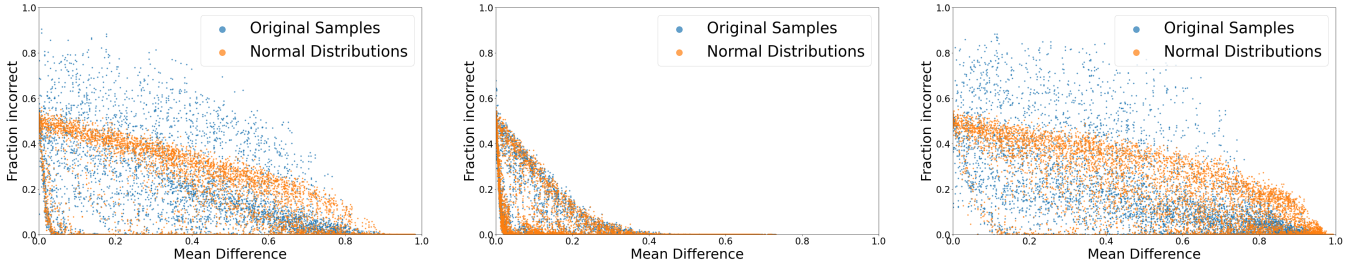
6 RACING

To investigate the impact of performance variability on algorithm configuration, we focus on the racing procedures used by irace, which we simulate using the **high-quality modCMA configurations** from Section 2.4. In particular, we consider two variants of the racing procedure [17]using either the t-test or the Friedman-test. In addition to these racing variants, we also consider two variants of Successive HAlving (SHA) [12] with reduction factors 2 and 3, respectively. For the races using statistical tests, we loosen the total budget restriction, which is usually used as stopping criteria [16], (e.g., in irace) to 10 000 total samples, which means we continue the race until 5 or fewer configurations remain, or until we exceed 10 000 sampled runs ('target runs' in irace terminology). We simulate this race 1000 times for each function and several values of *FirstTest*, and show the resulting performance loss for F9, F15, and F22 in Figure 11. In this figure, performance loss is defined as the difference in true mean of the best elite (configuration with the best sampled mean during the race) against the best configuration which was present in the race.

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

When comparing the different approaches, we note that there is not a clear winner across all functions and values of *FirstTest*. Interestingly, for some of the functions where Figure 4 shows the largest performance losses of irace elites, the races using the Friedman test seem to perform relatively poorly. This might indicate that for these functions, we could regain some of the lost performance, if it can be detected during the algorithm configuration that a different testing strategy would be required.

From Figure 11, we can clearly see that any any variant of racing or SHA is much more reliable than the *sampling-only* approach. However, racing uses more total samples, since it add runs when needed, while the sampling-only approach uses a fixed number



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

Figure 8: Fraction of incorrect decisions when using the sample mean to compare pairs of modCMA configurations. Each subplot contains 10 000 points. Each point compares two configurations selected uniformly at random from the available configurations. The x-axis indicates the normalized difference between their true means (based on the 200 AOC values per configuration). The y-axis indicates the fraction of incorrect decisions based on 500 independent samplings of 15 AUC values for each of the two selected configuration. *Original samples* refers to sampling with replacement from the 200 AOC values available, while *Normal distributions* refers to sampling values from a normal distribution with the same mean and standard deviation as the 200 values of the corresponding configuration.

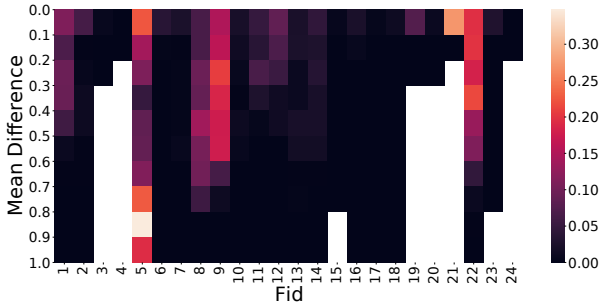


Figure 9: Fraction of configuration-pairs where the t-test gives an incorrect conclusion in more than $\alpha = 0.05$ of cases, on each of the 24 BBOB functions when considering 10 000 random pairs of modCMA configurations and a sample size of 15.

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

In order to account for the differences in total budget, we summarize cumulative performance loss curves, such as those in Figure 11, using their corresponding AUC values, and plot these AUC values against the total samples used in Figure 12.

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

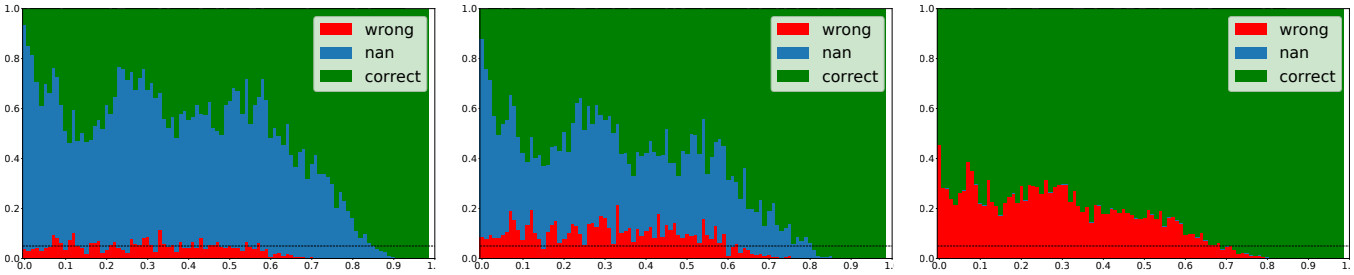
7 CONCLUSIONS AND FUTURE WORK

In this work, we have highlighted that commonly used performance metrics can have various non-normal distributions with large amounts of variance. While this is inherent to the field of iterative optimization heuristics, the impact of this stochasiticity is often overlooked in empirical work, where rankings between algorithms are made purely based on aggregations of limited runs, which can lead to incorrect decisions between algorithms.

While using statistical test largely alleviates these problems, they are not a silver bullet. Since data is often shared to compare new algorithms to the state-of-the-art, continuous re-use of the same few data points has the potential to lead to bias, especially when the underlying distribution has a high variance. A similar effect can be clearly seen in algorithm configuration, where choices between similarly performing algorithms are made, and even if each individual choice is valid, the overall result is likely to significantly underestimate the true performance of the chosen configuration. This can lead to a selection of sub-optimal configurations, which can be considerably worse than others that participated in the same race.

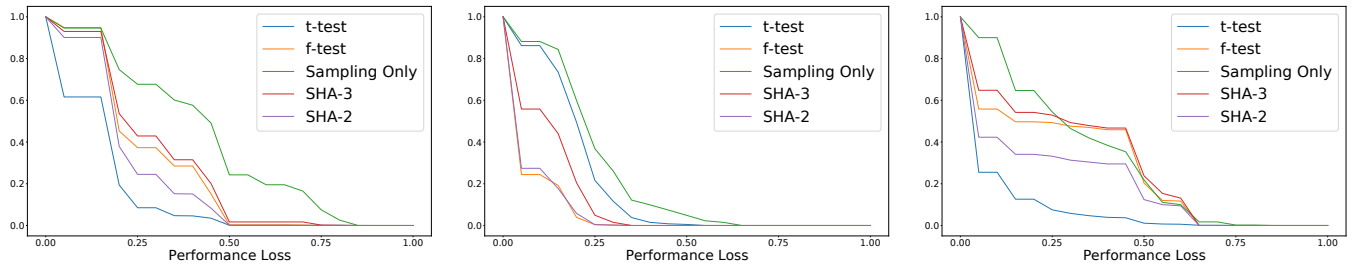
In an ideal case, each time a comparison is done, the used samples from both algorithms should be newly generated. However, this is obviously infeasible, as computation time is a limited resource. As such, making a larger set of samples available would be a more practical solution. The recommended number of samples would differ on a per-function basis, as some functions inherently cause algorithms run on them to have a higher variance of performance. Power-analysis studies [2, 6] based on algorithms for which performance data is already available might help us find better defaults, but since we can not know what kind of distributions the collected samples will be compared against in future, this might not fully solve the problem.

In addition, a more robust statistical analysis of the commonly used performance measures would be highly beneficial to gain more insight into the reasons for the observed errors. To aid with reproducibility, moving from standard hypothesis testing to the safe variant [7], where samples can be added continuously, would allow us to add more samples when this is deemed necessary. If this is combined with better guidelines for code availability and standards for data sharing (including formatting guidelines to ease



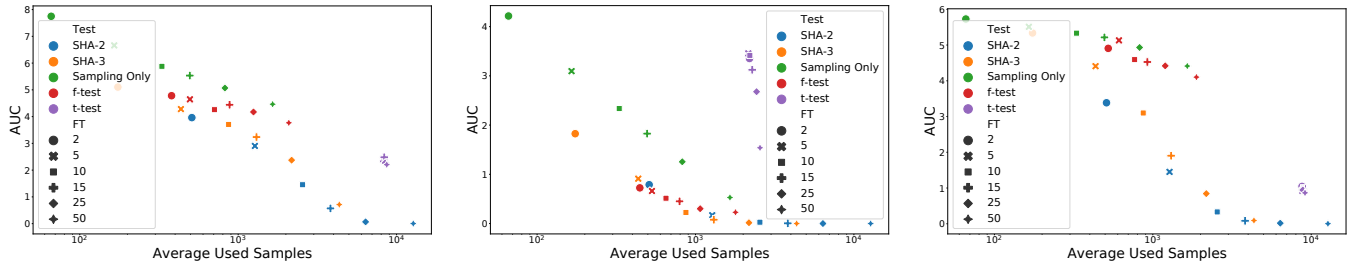
(a) t-test (b) Wilcoxon ranked-sum test (c) Comparison means

Figure 10: Correctness of decisions made in pairwise comparisons between modular CMA-ES configurations on F9, using different procedures. The x-axis shows the relative difference in true mean between the selected configurations. The y-axis shows the fraction of comparisons, out of 500 repetitions, that the decision was correct, incorrect or inconclusive (nan) when comparing configurations with this difference. Each repetition samples 15 values out of the 200 available for each configuration compared. These figures are available for all 24 functions and multiple sample sizes in our figshare repository [23].



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

Figure 11: Cumulative performance loss of 5 variants of the racing procedure using *FirstTest* = 2: t-test, Friedman-test, sampling and selecting based on mean, and successive halving with reduction factors 2 and 3.



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

Figure 12: Comparison of AUC value of Cumulated performance loss (Figure 11), relative to the average amount of samples used by each process.

interoperability), it will allow any researcher to gather new samples from existing algorithms to expand algorithm comparisons where needed, without hurting the statistical rigour of the comparison.

Reproducibility. To ensure that the work shown in this paper is reproducible [15], all data and code used is made available on Zenodo [22]. This includes figure generation code for figures that have not been included here because of the limited space available. For ease of viewing, these additional figures have also been uploaded on figshare [23]. In particular, Figure 1 for all BBOB functions and additional performance metrics, Figure 7 for all functions and more combinations of configurations and Figures 8, 10, 11 and 12 for all functions and different sample sizes.

ACKNOWLEDGMENTS

M. López-Ibáñez is a "Beatriz Galindo" Senior Distinguished Researcher (BEAGAL 18/00053) funded by the Spanish Ministry of Science and Innovation (MICINN). This work is supported by the Paris Ile-de-France region, via the DIM RFSI AlgoSelect project.

REFERENCES

[1] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel López-Ibáñez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. 2020. Benchmarking in Optimization: Best Practice and Open Issues. *Arxiv preprint arXiv:2007.03488 [cs.NE]* (2020). https://arxiv.org/abs/2007.03488

[2] Felipe Campelo and Elizabeth F. Wanner. 2020. Sample size calculations for the experimental comparison of multiple algorithms on multiple problem instances. *Journal of Heuristics* 26, 6 (2020), 851–883. https://doi.org/10.1007/s10732-020-09454-w

[3] 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

[4] 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 *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO Companion 2021*, Francisco Chicano and Krzysztof Krawiec (Eds.). ACM Press, New York, NY, 1375–1384. https://doi.org/10.1145/3449726.3463167

[5] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2018. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *Arxiv preprint arXiv:1806.07555* (Oct. 2018). https://doi.org/10.48550/arXiv.1810.05281

[6] Paul D Ellis. 2010. *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results.* Cambridge university press.

[7] Peter Grünwald, Rianne de Heide, and Wouter M Koolen. 2020. Safe testing. In *2020 Information Theory and Applications Workshop (ITA)*. IEEE, 1–54.

[8] 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* 36, 1 (2020), 1–31. https://doi.org/10.1080/10556788.2020.1808977

[9] 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, France. Updated February 2010.

[10] Nikolaus Hansen and Andreas Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, Thomas Bäck, T. Fukuda, and Zbigniew Michalewicz (Eds.). IEEE Press, Piscataway, NJ, 312–317. https://doi.org/10.1109/ICEC.1996.542381

[11] Changwu Huang, Yuanxiang Li, and Xin Yao. 2020. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Transactions on Evolutionary Computation* 24, 2 (2020), 201–216. https://doi.org/10.1109/TEVC.2019.2921598

[12] Zohar Karnin, Tomer Koren, and Oren Somekh. 2013. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, Sanjoy Dasgupta and David McAllester (Eds.), Vol. 28. 1238–1246. http://jmlr.org/proceedings/papers/v28/

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

[14] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52.

[15] Manuel López-Ibáñez, Jürgen Branke, and Luís Paquete. 2021. Reproducibility in Evolutionary Computation. *ACM Transactions on Evolutionary Learning and Optimization* 1, 4 (2021), 1–21. https://doi.org/10.1145/3466624

[16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. 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

[17] O. Maron and A. W. Moore. 1997. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Research* 11, 1–5 (1997), 193–225.

[18] J. Rapin and O. Teytaud. 2018. Nevergrad: A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.

[19] Sander van Rijn. 2018. Modular CMA-ES framework from [20], v0.3.0. https://github.com/sjvrijn/ModEA. Available also as pypi package at https://pypi.org/project/ModEA/0.3.0/.

[20] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. 2016. Evolving the structure of Evolution Strategies. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, Xuewen Chen and Andreas Stafylopatis (Eds.). 1–8. https://doi.org/10.1109/SSCI.2016.7850138

[21] Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2020. Integrated vs. Sequential Approaches for Selecting and Tuning CMA-ES Variants. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020*, Carlos A. Coello Coello (Ed.). ACM Press, New York, NY. https://doi.org/10.1145/3377930.3389831

[22] Diederick Vermetten, Hao Wang, Manuel López-Ibáñez, Carola Doerr, and Thomas Bäck. 2022. *Analyzing the Impact of Undersampling on the Benchmarking and Configuration of Evolutionary Algorithms – Dataset.* https://doi.org/10.5281/zenodo.5925410

[23] Diederick Vermetten, Hao Wang, Manuel López-Ibáñez, Carola Doerr, and Thomas Bäck. 2022. Performance Variability - Figures. https://doi.org/10.6084/m9.figshare.18857486.v1