Large-scale Benchmarking of Metaphor-based Optimization Heuristics

Diederick Vermetten, Carola Doerr, Hao Wang, Anna Kononova, Thomas Bäck

Large-scale Benchmarking of Metaphor-based Optimization Heuristics

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

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

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

Anna V. Kononova
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands
a.kononova@liacs.leidenuniv.nl

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

ABSTRACT

The number of proposed iterative optimization heuristics is growing steadily, and with this growth, there have been many points of discussion within the wider community. One particular criticism that is raised towards many new algorithms is their focus on metaphors used to present the method, rather than emphasizing their potential algorithmic contributions. Several studies into popular metaphor-based algorithms have highlighted these problems, even showcasing algorithms that are functionally equivalent to older existing methods. Unfortunately, this detailed approach is not scalable to the whole set of metaphor-based algorithms. Because of this, we investigate ways in which benchmarking can shed light on these algorithms. To this end, we run a set of 294 algorithm implementations on the BBOB function suite. We investigate how the choice of the budget, the performance measure, or other aspects of experimental design impact the comparison of these algorithms. Our results emphasize why benchmarking is a key step in expanding our understanding of the algorithm space, and what challenges still need to be overcome to fully gauge the potential improvements to the state-of-the-art hiding behind the metaphors.

CCS CONCEPTS

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

1 INTRODUCTION

When faced with an optimization problem, we have an ever-growing pool of algorithms to choose from. New optimization algorithms are continuously being proposed, which means that the challenge of understanding the state-of-the-art becomes harder by the day. For new approaches to stand out from everything else, researchers often refer back to metaphors as a framing device for their algorithms.

The usage of metaphors to illustrate algorithmic ideas has been around for a long time and some of the most well-established algorithm families in the field made use of metaphors such as Darwin's theory of evolution in Evolutionary Computation [2], the swarming behavior of bird-like objects in the Particle Swarm Optimization algorithm [17] or the foraging behavior of ants in Ant Colony Optimization [13]. Given the successes of these methods, it is natural that many new algorithms follow the same approach. There is, however, an increasingly visible problem, where the metaphor seems to become more important than the algorithm, which hinders the understanding of an algorithm's contribution to the state of the art [27]. In some cases, this leads to duplicated algorithms with different names, which grow to be highly cited and viewed as independent algorithms by practitioners [5, 6].

Even though many optimization algorithms are presented with an emphasis on the metaphor, there might still be interesting ideas and insights to be gained from understanding these algorithms in more detail. Given the widespread usage of these types of algorithms, writing off a method because of how it is presented seems counterproductive.

In this paper, we focus on benchmarking publicly available implementations of a wide variety of optimization algorithms. We specifically do not address the question of whether these algorithms contain any novelty in terms of algorithmic operators and only focus on showcasing their performance within a large benchmarking scenario. We highlight the ways in which benchmarking helps to gain insight into the strengths and weaknesses of optimization algorithms and discuss the potential benefits to be gained from such benchmark setups. In particular, we show that performance between algorithms is highly varied, with some algorithms performing consistently worse than RandomSearch, while others manage to outperform several well-established baselines. By considering two types of performance measures, we highlight the dependence of results on the used benchmarking setup. Finally, we discuss some of the challenges inherent to benchmarking studies, especially when performed on newly proposed algorithms. This includes questions regarding precise algorithm implementation, which causes seemingly the same algorithm to show widely different performance in our benchmark.

2 RELATED WORK

Within the optimization field, metaphor-based optimization algorithms have been receiving quite some criticism in the last decade [32]. One of the key arguments is that the usage of metaphors

throughout an algorithm description does not advance our understanding of the algorithm, but only hides its true design ideas [27, 28]. To gain some idea of the scope of these types of algorithms, a community effort has been made in the form of the evolutionary computation bestiary, which documents the rise of metaphor-based algorithms over time [7].

While detailed analyses of these optimization algorithms are time-consuming, in the last years several highly visible algorithms have been shown to contain no novelty over the previously existing algorithm families [3, 5, 6]. This has led to the community asking for stricter requirements when new algorithms are proposed [1], which are being adopted slowly, such as by the Transactions on Evolutionary Learning and Optimization (TELO) and Evolutionary Computation Journal (ECJ) journals where metaphor-based algorithms are now highly discouraged. On a wider scale, some initial benchmark studies have been performed recently, which highlight that many implementations of these metaphor-based algorithms perform very similarly to each other [11, 22]. Finally, a study into some behavioral properties of these algorithms has shown that many of them are highly biased towards the center of the domain, leading to misleading performance comparisons if benchmarked on functions with similar types of bias [20].

3 EXPERIMENTAL SETUP

While nature-inspired optimization heuristics are common, it is often challenging to find open-source implementations of these algorithms which have been validated by the authors. We thus rely on third parties who have designed libraries containing a variety of algorithm implementations in relatively accessible formats. For this study, we identified four such libraries implemented in Python and used the following number of algorithms from them:

- 14 algorithms from EvoloPy [14], accessible at https://github.com/7ossam81/EvoloPy.
- 53 algorithms from Niapy [37], accessible at https://github.com/NiaOrg/NiaPy.
- 139 algorithms from Mealpy [31], accessible at https://github.com/thieu1995/mealpy.
- 76 algorithms from Opytimizer [10], accessible at https://github.com/gugarosa/opytimizer.

In addition to these libraries, we include some established algorithms as baselines, taken from the Nevergrad toolbox [26]. From Nevergrad, we utilize 8 algorithms: DE, diagonal CMA-ES, multi-BFGS, 1+1 ES, PSO, Powell, Cobyla, and RandomSearch. Finally, we include two configurations from modular algorithm families: CMA-ES and BIPOP-CMA-ES from the modCMA package [8] and DE and L-SHADE from the modDE package [33]. To ease the analysis, we group these last 12 algorithms under the 'Baselines' denominator. All algorithms are used with default parameters settings from their respective libraries.

In total, our portfolio consists of 294 algorithm implementations. Each of these algorithms is benchmarked on the single-objective, noiseless BBOB suite [16], using the IOHexperimenter package [9]. For these experiments, we use all 24 functions contained in the BBOB suite, in dimensionalities $d \in \{2, 5, 10, 20\}$. For each function, we perform 5 independent runs on each of the first 10 instances. In total, this gives us 1 411 200 runs. The budget for each of these runs is set to $B = 10\,000 \cdot d$.

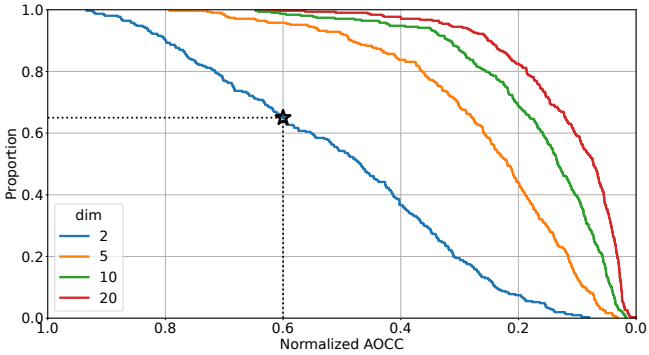


Figure 1: Cumulative Distribution of AOCC (default bounds) worse than x-axis value for all 294 algorithms in the portfolio. For example, in dimension 2, the star indicates that the fraction of algorithms with an AOCC below 0.6 is 0.65. AOCC values shown are aggregated over all 24 BBOB functions.

Throughout this paper, we look at two different performance measures. The first is a standard fixed-budget setting, where we look at the precision (difference between best-so-far and optimal function values) at specific budgets. Since convergence curves are usually logarithmic, when aggregating these precisions we use geometric means unless stated otherwise. The second performance measure we use is an anytime measure: the normalized area over the convergence curve (AOCC), which is defined as follows:

$$AOCC(\vec{y}) = \frac{1}{B}\sum_{i=1}^{B}\left(1 - \frac{\min(\max((y_i), lb), ub) - lb}{ub - lb}\right)$$

where $\vec{y}$ is the vector of best-so-far precision reached during the optimization run, $B = 10\,000 \cdot d$ is the budget, lb and ub are the lower and upper bound of the precision values we consider. To be consistent with existing benchmarking studies on BBOB, when we refer to AOCC we make use of precision bounds $lb = 10^{-8}$ and $ub = 10^2$ with a logarithmic scaling between them. Since for higher dimensionalities $ub = 10^2$ can be a challenging bound on many functions, we also include some results on *AOCC* with a relaxed upper bound of $ub = 10^8$, which we refer to as *AOCCIarge*. For all figures relating to *AOCC* or *AOCCIarge*, an equivalent figure with the other bounds is available on our Figshare repository [34].

Reproducibility. To ensure the reproducibility of our work, our full benchmarking setup, raw and processed data and all scripts used for analysis and visualization presented in this paper are made available on Zenodo [34].

4 ANYTIME PERFORMANCE RESULTS

To investigate the overall performance of the selected algorithm portfolio, we look at the distribution of average AOCC across all functions, separated by problem dimensionality, which is visualized in Figure 1. From this figure, we can see that there is a rather wide spread of performance within the portfolio, especially for the

lower dimensionalities. As dimensionality increases, not only do the functions become more challenging to optimize to the same precision, but the shape of the performance distribution changes as well. With growing dimensionality, there are some well-performing algorithms, after which average performance seems to drop off exponentially. This suggests that relatively few algorithms can scale effectively with regard to dimensionality.

Since Figure 1 can provide only a very highly aggregated view of the underlying performance data, we next look at the distribution of AOCC on a per-function and per-algorithm level. For each dimensionality, we create a heatmap indicating this per-function AOCC, shown in Figure 2. Since our portfolio consists of 294 algorithms, we highlight the baseline algorithm with a red rectangle. In addition, we color-code the libraries as follows: Baselines, Opytimizer, Niapy, Evolopy and Mealpy. This coloring will remain consistent throughout all further figures.

From Figure 2, we observe that, while several baseline algorithms are near the top, the best-performing algorithms come from a combination of different libraries. In general, there is no clear ordering between the libraries in terms of performance. When zooming in on the best algorithms, we also notice some patterns between the BBOB functions. For example, in dimensionality 10, the top-performing algorithm (BIPOP-CMA-ES), achieves relatively poor anytime performance on functions 3 and 4, while the second-best algorithm (JADE) manages those functions rather well. These kinds of differences highlight the potential complementarity between the algorithms in our portfolio.

When looking at the worst performing algorithms in Figure 2, we note that RandomSearch is not the worst-performing algorithm. While the total number of algorithms which are worse on average decreases as the dimensionality grows, the total number of algorithms which fail to beat this baseline is not insignificant. To further analyze this aspect of our portfolio's performance, we identify per function how many algorithms achieve worse AOCC than RandomSearch by at least 10 percent, and show the results in Figure 3a. Since the ability of RandomSearch to hit the upper bound of AOCC decreases as the problem dimensionality grows, there are some functions where it has an AOCC of 0, leading to no algorithms being considered worse. On some other functions, such as F11, it seems that RandomSearch is quite adequate in terms of anytime performance, beating over 30 percent of algorithms. For the remaining functions, there are a rather large portion of algorithms which compare poorly. Figure 3b highlights the 20 algorithms which lose the comparison on the most functions. Given that some of these algorithms are worse on all 2-dimensional problems, it seems likely that their implementation is not fully functional. This confirms the importance of including RandomSearch as an algorithm to compare to in any benchmarking study.

Similar to the poorly performing algorithms, the left side of Figure 4 zooms in on which algorithms achieve good performance on at least one function. Here, we characterize good performance as being ranked in the top 3 algorithms within our portfolio based on AOCC. While there are 96 (function, dimension) combinations, only 45 unique algorithms are in the top 3 for at least 1 function, and 20 of those show up exactly once. To gauge the overall performance of these algorithms, we calculate their total loss (difference in AOCC value to the best algorithm on each function, averaged over all functions), which is shown in the middle column of Figure 4. While the top-performing algorithm in both figures is the same (BIPOP-CMA-ES), it is interesting to note the differences. Specifically, multiBFGS performs in the top for 26 (function, dimension) combinations, but in terms of average loss, it is only ranked 14th. This indicates that multiBFGS is a rather specialized algorithm, which leads to good performance on some types of problems, at the cost of worse performance on other function groups. On the other hand, the JADE algorithm is only in the top 3 for 6 (function, dimension) pairs, but ranks 3rd based on overall loss. Finally, we look at whether these algorithms could improve upon the set of baselines we consider. To achieve this, we consider a portfolio's performance to be the average of the minimum AOCC its component algorithms achieve on each function. By computing this measure for the set of all baselines, and for the set of the baselines with each considered algorithm included, we have a measure of contribution. These are shown on the right side of Figure 4, and highlight that purely considering the number of competitive functions ignores the scale of improvements, as can be seen for example in RRA, which is competitive in only 1 20-dimensional function, but contributes a lot to the 20-dimensional baseline portfolio.

5 FIXED-BUDGET RESULTS

In addition to the anytime performance metric, we can also analyze our data from a fixed-budget viewpoint. By recording the full performance trajectory during the benchmarking, we can compare different final budget values. This allows us to highlight the impact of the available optimization budget on the relative performance of the different optimization algorithms. For this analysis, we choose 7 different budget factors: $b \in \{10, 50, 100, 500, 1\,000, 5\,000, 10\,000\}$. For each dimensionality, the total budget is then set to $b \cdot d$.

Since we observed in Figure 4 that few algorithms were ever ranked in the top 3 on any given function, we start by analyzing the impact of budget on this finding. For a given dimensionality, we consider which algorithm is ranked first based on the average function value reached after the given budget has been used. This is visualized in Figure 5. Since we consider the problem optimized when a precision of 10^{-8} is achieved, ties can occur, especially on the 'easier' problems with large budgets. As such, whenever 2 or more algorithms are tied, we don't make a distinction between them and instead only show how many algorithms are tied for that particular setting (shown in light cells with brown text).

From Figure 5, it is clear that there are indeed many ties occurring, in particular for the sphere (F1) and linear slope (F5), where a large fraction of algorithms manages to reach the same function value after $10\,000 \cdot d$ evaluations. We also observe some interesting patterns in terms of which algorithm ranks first on particular functions. For example, in F17 and F18 at low budgets, Opytimizer's DE performs well, but it is overtaken by BIPOP as the number of function evaluations increases.

To better gauge how much each algorithm contributes to the overall performance of our portfolio, we make use of a Shapley-based analysis. In particular, we consider a fixed budget-factor and problem dimensionality, and with this setting we consider the performance of a set of algorithms to be the sum over all functions of the minimal average precision reached by an algorithm in the

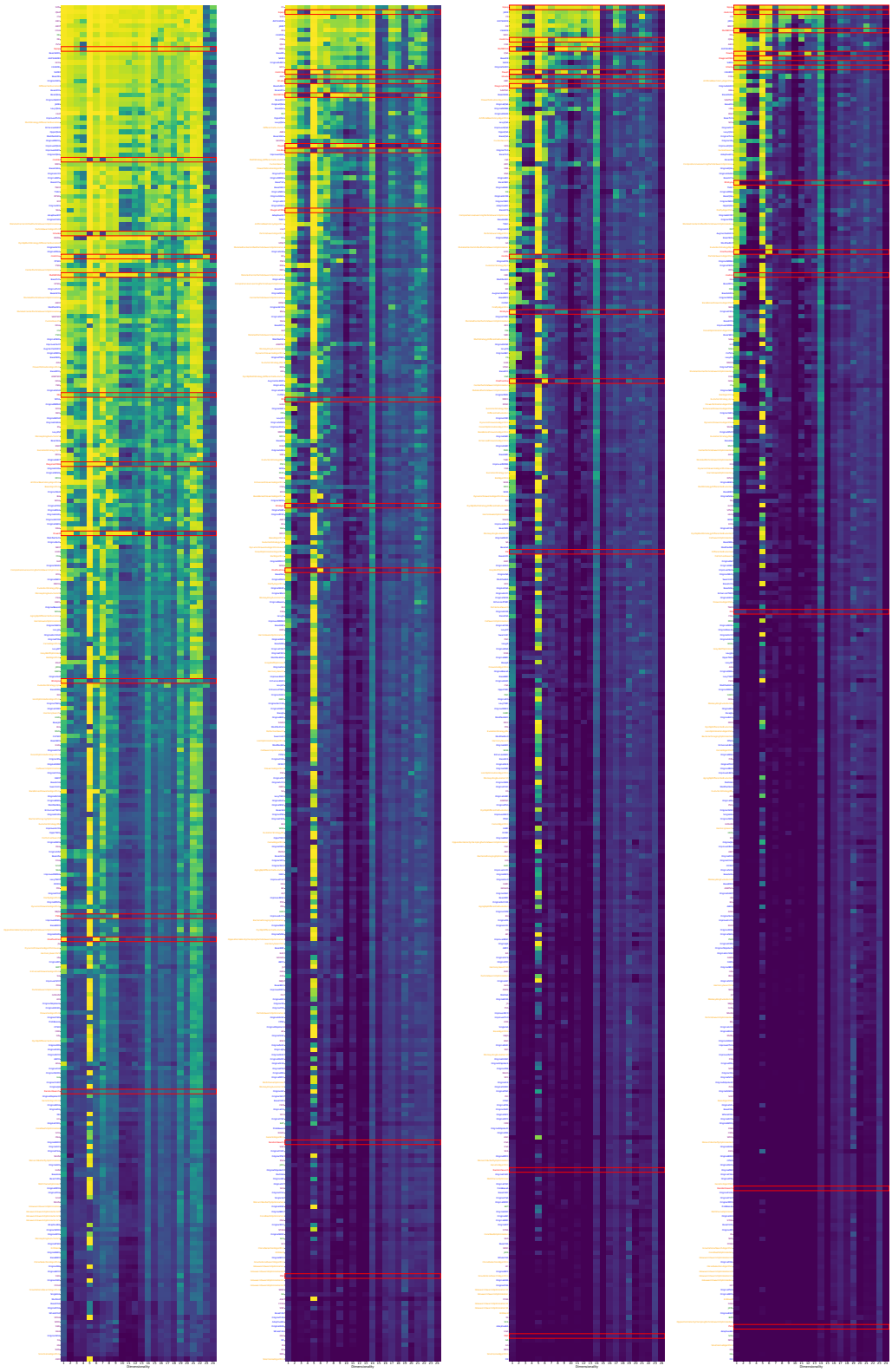
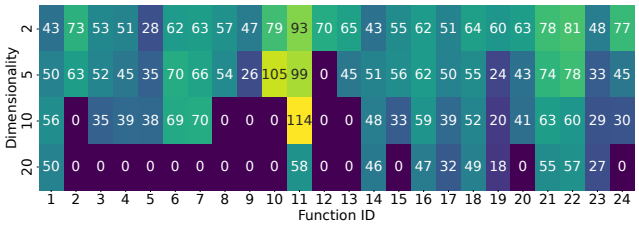
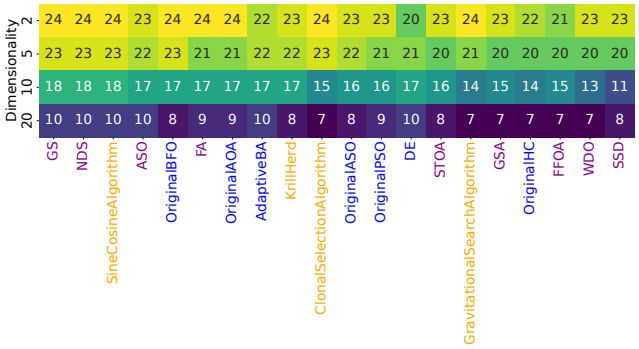


Figure 2: Normalized AOCC values per function for all 294 algorithms, ordered from dimensionality 2 (left) to 5, 10 and 20 (right). Color scales from dark blue=0 (worst) to yellow=1 (best). Larger versions of these figures are available on our Figshare repository [34]. Colors denote the algorithm's library and algorithms are sorted by total AOCC over all functions.



(a) Number of algorithms (out of 294) which are worse than RandomSearch in each (function, dimensionality) combination.



(b) On how many functions (out of 24) the selected algorithms perform worse than RandomSearch in each dimensionality. Algorithms are selected based on the total number of functions on which the algorithm is worse than RandomSearch, with the top 20 of these algorithms included.

Figure 3: Comparisons of algorithms performance to RandomSearch based on AOCC. An algorithm is considered worse on a function if its AOCC is at least 10% less than that of RandomSearch.

set on that function. The marginal contribution of an algorithm to a set is thus the difference in total precision when this algorithm is included and when it is excluded from the given set. By averaging this marginal contribution over 250 sets of sizes between 1 and 20 (for a total of 5 000 sets), keeping the sets consistent across algorithms, we obtain an approximate Shapley value indicating to what extent the given algorithm contributes to the overall portfolio.

In Figure 6, we show the normalized version of these approximate Shapley values for 30 algorithms, for all dimensions and budget factors. These 30 algorithms were selected based on the best average approximate Shapley values over all settings. In this figure, we can observe a clear difference between algorithms which contribute more as budget increases (BIPOP, CMA-ES, several DE versions) and those which perform better at lower budgets, which are in the majority. When looking across dimensionalities, we see that, e.g., the ABC and GCO algorithms are rather effective in low dimensionalities, but fail to scale up effectively.

6 DISCUSSION

Understanding strengths and weaknesses of algorithms. As illustrated throughout this paper, algorithm performance data can be processed in a wide variety of ways. By varying the performance measure,

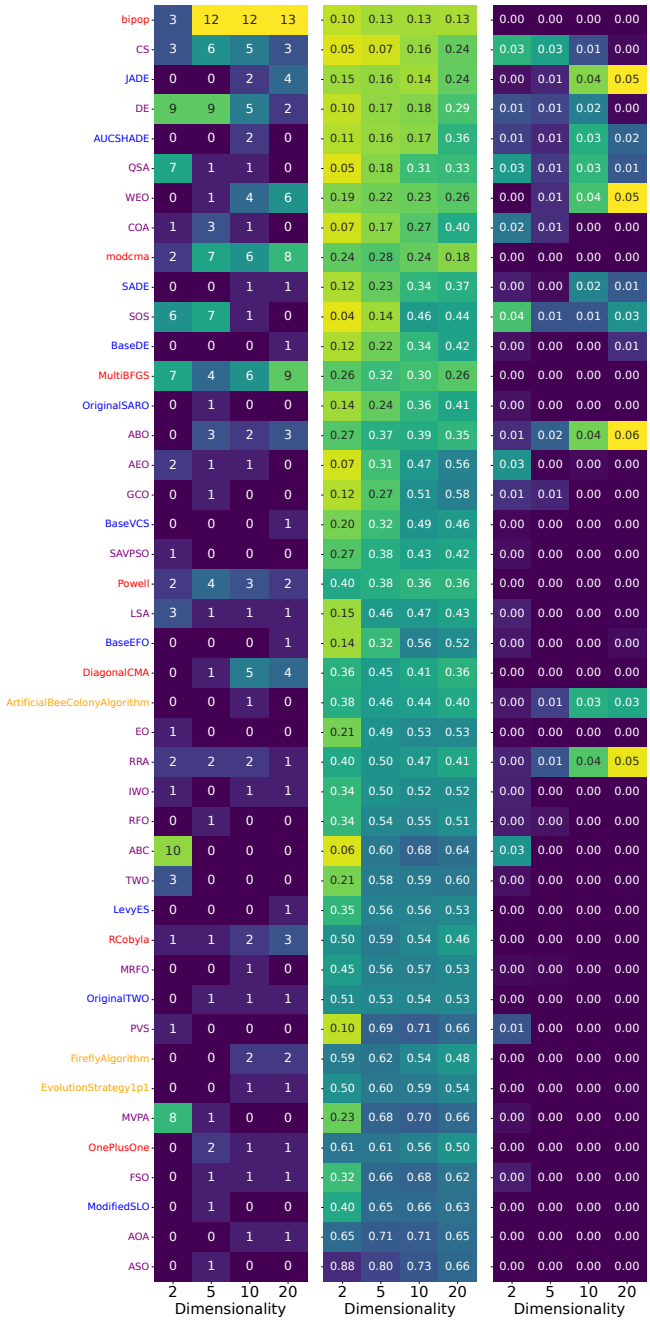


Figure 4: Left: Number of functions (out of 24) on which each algorithm is in the top 3 (on average AOCC). Middle: Average loss (absolute difference to best AOCC per function) over all 24 BBOB functions for each algorithm. Right: Contribution to an algorithm portfolio consisting of all baselines. Only algorithms which are considered competitive on at least one function are included, for a total of 43 unique algorithms.

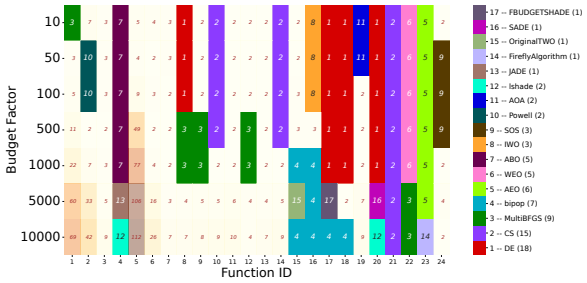


Figure 5: Best algorithm for each (budget, function) combination in dimensionality 10, based on average precision with a cutoff at 10^{-8}. Light cells with brown text indicate a tie, with the number in the cell indicating how many algorithms are tied. The number in brackets after the algorithm name indicates how often it occurs in the figure.

type of comparison or level of aggregation, many different research questions can be investigated. While we are limited to only a few pages of results, providing access to the performance data allows others to re-use and expand on our analysis. For example, our data could be de-aggregated to only compare algorithm performance on a two-dimensional sphere to compare convergence speed between algorithms. The benefits of sharing performance data publically are clear to see when considering COCO's long-standing collection of performance data which comes from hundreds of algorithms, collected over more than a decade. Not only does this allow researchers to compare performance to a variety of known algorithms, but insights gained from looking at benchmark data can often inspire further research, and sometimes even lead to theoretical studies of empirically-found effects [12].

A complementary aspect of understanding an algorithm's strengths is identifying whether it shows some performance characteristics which are not present in established algorithms. This might suggest that an algorithm contains useful new ideas or a way of combining existing ideas in a beneficial manner. This is particularly relevant for the type of nature-inspired optimization algorithms we discussed here, since many of them are introduced based on a metaphor, which obfuscates the underlying algorithmic concepts. In many cases, algorithms can be widely used for years, to then be found to be equivalent to an existing algorithm with modified naming schemes. An example of this is the Cuckoo Search algorithm (CS), which performed especially well in this paper. However, it has been shown that CS contains no novelty, and is simply a reformulation of an existing ES variant [5].

Since these detailed investigations into an algorithm's underlying principles can be time-consuming, benchmarking data offers a way to identify which algorithms might be worthwhile to analyze further. By looking for algorithm implementations which show strengths complementary to a set of baselines, we might discover the most high-potential algorithms from the larger portfolio. Given our benchmark data, we can look for algorithms which perform decently on average but are high-performing on a different set of (function, dimension) combinations than our baselines. To illustrate this approach, we can visualize the performance space using a dimensionality reduction technique, in our case UMAP [23], to reduce the performance representation of the algorithm down to 2 dimensions. In this case, the performance vectors are created by concatenating the per-function AOCC on all four dimensionalities, resulting in a 96-dimensional vector space. In the reduced version of this space, shown in Figure 7, we can see a cluster of well-performing algorithms on the top-right, which include several of the baseline algorithms. To zoom in on the relation between baselines and non-baselines, we can explicitly plot the average performance relative to the distance in performance space to the nearest baseline, as is done in Figure 8. Here, the algorithms in the top-right are of interest, since they both achieve good average performance as well as distinct performance differences from the closest baseline. These algorithms seem to be the most complementary to our baselines and could thus be the first candidates for further analysis.

Data-driven algorithm selection. Since algorithms generally have different strengths and weaknesses, it makes sense to use this knowledge to make informed decisions on which algorithm to use to solve a given problem. Based on benchmark data, we can determine which algorithm performs best for given problem settings, where "*problem settings*" can differ in resources available to solve the problems (budget, possibility to execute evaluations in parallel, etc) and on characteristics of the problems. This can result in recommendation systems based on aggregate performance over a varied set of functions, based only on problem dimensionality, budget, etc. A recent example of this is the NGopt wizard [25]. Alternatively, when considering the problem characteristics in more detail, for example by using part of the budget to compute low-level landscape features [24], we can exploit complementarity in performance on the function-level, or even the instance-level, to achieve automated algorithms selectors [18].

Chaining rules. In addition to problem-level algorithm complementarity which can be exploited with algorithm selection methods, we can also use benchmark data to identify per-function complementarity. If one algorithm performs well at the beginning of the search but fails to converge to the optimum, while another algorithm takes a long time to find the right region but once found converges quickly, a hybrid algorithm which chains these two together could lead to significant improvements in anytime performance [35, 36].

Judging Algorithmic Contributions. Given the many different layers of algorithm complementarity which can be exploited using various learning methodologies, we should carefully consider how the contribution of an algorithm to the state of the art is judged. There is a clear distinction between generalist algorithms, which perform well across many functions, and specialist algorithms, which exploit specific function properties to achieve great performance on some, while losing out on others. Many newly proposed algorithms are studied from the generalist viewpoint, aiming to perform better than existing algorithms across wide suites of functions. This focus might mean missing out on interesting specialists, which could push the field forward by improving performance on a smaller set of functions. In communities like SAT-solving, competitions are now focussed more on contribution to a portfolio, rather than average performance, which has led to very potent solvers that select

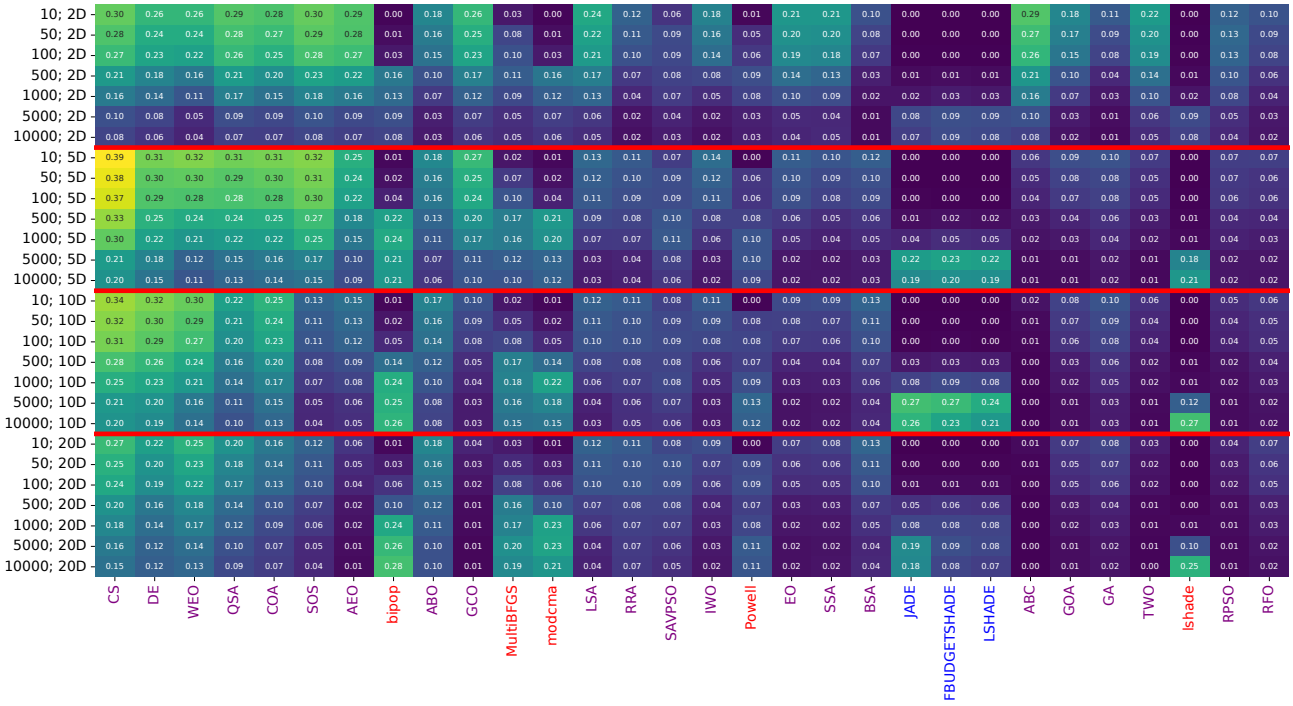


Figure 6: Normalized approximate Shapley values for each shown algorithm to portfolios of size at most 20, for different budget factors and dimensionalities. Algorithms are sorted based on total contribution across all functions, dimensions and budgets. Shapley values are computed based on fixed-budget contribution in log-space, capped at 10^{-8}

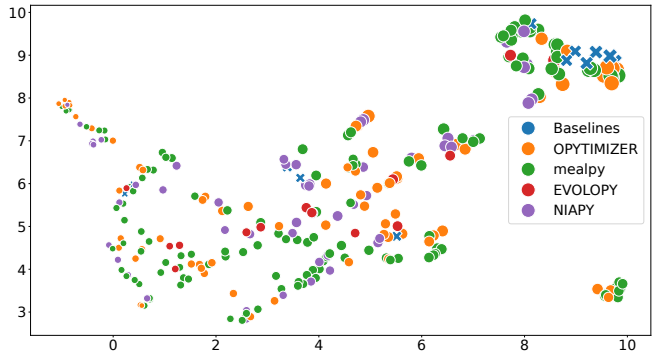


Figure 7: UMAP projection of the 96-dimensional (4 dimensionalities times 24 functions) AOCC vectors for each algorithm. Dots are sized based on average performance, with larger dots performing better.

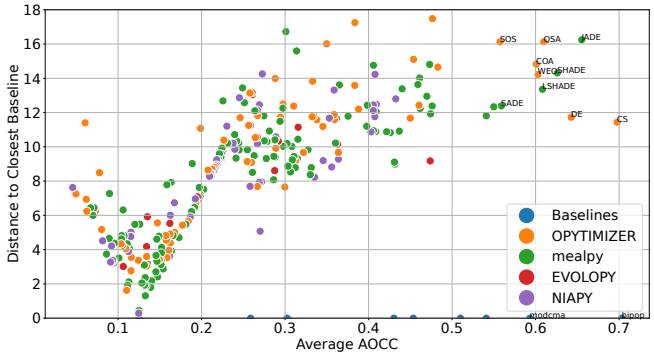


Figure 8: Relation between mean performance (AOCC averaged over all functions, dimensions) and distance to the closest baseline algorithm (manhattan distance). Algorithms with an average AOCC over 0.55 are annotated.

and combine the individual competitors [39]. These approaches could be highly beneficial within optimization as well.

Underspecification of Parameterization / Implementations. The algorithm portfolio we considered in this paper is a combination of algorithms from several different libraries, and as such some overlap between them is likely. We opted not to remove these duplicate algorithms, since the specific implementations might differ significantly. For example, many libraries contain some form of differential evolution (DE), which is a large algorithm family in its own right. An algorithm called DE could vary significantly based on which mutation operator or crossover variant is used, what

adaptation rules and parameter settings are chosen or even how solutions outside the domain are handled [19]. Even for specific versions of DE, such as the L-SHADE algorithm [29], we observe differences between the implementation in modDE and Mealpy, as can be seen in Figure 6. In this case, the Mealpy version performs better at low budgets, while the modDE version becomes much more effective at larger budgets in higher dimensionalities. These differences once again highlight the need for code to be made available, as even detailed specifications like L-SHADE can result in these very different algorithm behaviors.

When code is not available, and papers are the only source of an algorithm's specification, there are often insufficiently detailed or ambiguous descriptions of components, which make it almost impossible to fully recreate the used algorithm, resulting in difficulties in judging the accuracy of reported results. In addition to requiring reproducibility of results and availability of code, a robust way to judge algorithmic contributions could involve more focus on algorithm modularity. When proposing a new algorithm, it can often be framed as a modification of existing algorithms and thus implemented in existing modular frameworks, which are being proposed for many common algorithm families [4, 8, 33]. This way, the algorithm can be fairly compared to the base version of the used algorithm, as well as other modifications made available in the chosen framework, and the relative impact of different modifications can be analyzed rigorously [30].

Algorithm Tunability. While algorithm modularity can be a useful tool, it also raises another important question about the way in which parameter settings should be considered in comparative benchmarking studies. Most algorithms inherently contain a set of parameters which can drastically impact their performance on a chosen benchmark collection. This is especially obvious when comparing the settings achieved by hyperparameter optimization (HPO) on modular algorithms to their default values [8]. If an algorithm is designed with tunability in mind, and its modules and/or parameters are set to perform well on a specific suite of functions, it should be no surprise if it outperforms a second algorithm tuned on a completely different set. As such, the question of which parameterizations were used should be kept in mind when drawing conclusions from benchmark data. In this study, we chose not to apply HPO to any of the considered algorithms, as the required computational effort would be too large. We believe that the data presented here could be used to guide more detailed follow-up work, which could address the question of how significant the results would change if HPO were to be applied equally to all algorithms.

Impact of Benchmarking Setup. In Sections 4 and 5, we looked at two different measures of algorithm performance. Many benchmarking studies, or papers in which new algorithms are introduced, tend to include only one type of analysis. As such, the choice of which performance measure to use is one of the many decisions which have to be made in order to present benchmark data. As can be seen when comparing, e.g., Figures 4 and 6, while some of the rankings between algorithms are consistent, they are by no means identical. Even within a specific performance perspective, other choices in the experimental setup, such as the available budget, all have an impact on the conclusions which can be drawn. One aspect in particular which is often cause for concern when a new algorithm is presented is the set of algorithms it is compared to. In the extreme case, one could present RandomSearch as a well-performing algorithm by comparing only to the implementations at the top of Figure 3, which would be misleading at best.

While there is no right answer to the question of which performance measure, budget, function suite or baseline set is most appropriate for a given study, care should be taken to motivate these design choices, and their setting should be kept in mind when presenting conclusions about an algorithm's performance.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have illustrated the potential benefits that can be gained from robust benchmarking of optimization algorithms, in particular for the large set of metaphor-based optimizers. While many challenges remain to fully assess and fairly compare the contributions made by these algorithms, our analysis highlights that benchmarking can shed light on the relative strengths of algorithms, identifying the most interesting candidates for follow-up studies. We have also identified a surprisingly large number of algorithms that struggle to outperform random sampling on problems that can be considered very easy.

To reduce the burden of setting up a sound benchmarking environment, a number of open-source software tools are being developed to reduce the barrier of entry to rigorous benchmarking. With tools like COCO [15], Nevergrad [26], IOHprofiler [38], and many more, it is now easier than ever to run benchmarking studies and to compare performance and behavioral data to that of hundreds and even thousands of previously evaluated ones. The use of standardized benchmarking practices and data recording practices also facilitates reproducibility and data sharing [21].

While our study focuses on the BBOB problem suite, it should be noted that we are not claiming that all studies should be run on this same benchmark suite. While there is some benefit in terms of comparability with existing data, sticking with a single benchmark suite might risk overfitting to the biases of that suite, which might result in worse generalizability. What matters to us is that algorithms are assessed in a fair manner, on problems that allow to assess strength and weaknesses of the algorithms in different optimization scenarios.

ACKNOWLEDGMENTS

This work was supported by CNRS Sciences informatiques via the AAP project IOHprofiler.

REFERENCES

[1] Claus Aranha, Christian Leonardo Camacho-Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. 2022. Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intell.* 16, 1 (2022), 1–6. https://doi.org/10.1007/S11721-021-00202-9

[2] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. 1997. Handbook of evolutionary computation. *Release* 97, 1 (1997), B1.

[3] Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. 2019. The intelligent water drops algorithm: why it cannot be considered a novel algorithm - A brief discussion on the use of metaphors in optimization. *Swarm Intell.* 13, 3-4 (2019), 173–192. https://doi.org/10.1007/S11721-019-00165-Y

[4] Christian L Camacho-Villalón, Marco Dorigo, and Thomas Stützle. 2021. PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. *IEEE Transactions on Evolutionary Computation* 26, 3 (2021), 402–416.

[5] Christian L Camacho-Villalón, Marco Dorigo, and Thomas Stützle. 2022. An analysis of why cuckoo search does not bring any novel ideas to optimization. *Computers & Operations Research* 142 (2022), 105747.

[6] Christian Leonardo Camacho-Villalón, Thomas Stützle, and Marco Dorigo. 2020. Grey Wolf, Firefly and Bat Algorithms: Three Widespread Algorithms that Do Not Contain Any Novelty. In *Proc. of Swarm Intelligence (ANTS) (LNCS, Vol. 12421)*. Springer, 121–133. https://doi.org/10.1007/978-3-030-60376-2_10

[7] Felipe Campelo and Claus Aranha. 2023. Lessons from the evolutionary computation bestiary. *Artificial Life* 29, 4 (2023), 421–432.

[8] 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO'21)*. ACM, 1375–1384. https://doi.org/10.1145/3449726.3463167

[9] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2023. Iohexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation* (2023), 1–6.

[10] Gustavo H de Rosa, Douglas Rodrigues, and João P Papa. 2019. Opytimizer: A nature-inspired python optimizer. *arXiv preprint arXiv:1912.13002* (2019).

[11] Javier Del Ser, Eneko Osaba, Aritz D Martinez, Miren Nekane Bilbao, Javier Poyatos, Daniel Molina, and Francisco Herrera. 2021. More is not always better: insights from a massive comparison of meta-heuristic algorithms over real-parameter optimization problems. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–7.

[12] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. 2018. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1 + \lambda)$ EA variants on OneMax and LeadingOnes. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 951–958. https://doi.org/10.1145/3205455.3205621

[13] Marco Dorigo. 1992. *Optimization, Learning and Natural Algorithms*. Ph.D. Dissertation. Politecnico di Milano.

[14] Hossam Faris, Ibrahim Aljarah, Seyedali Mirjalili, Pedro A Castillo, and Juan Julián Merelo Guervós. 2016. EvoloPy: An open-source nature-inspired optimization framework in python. *IJCCI (ECTA)* 1 (2016), 171–177.

[15] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2021. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software* 36, 1 (2021), 114–144.

[16] 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

[17] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proc. of ICNN'95 - International Conference on Neural Networks*, Vol. 4. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

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

[19] Anna V Kononova, Diederick Vermetten, Fabio Caraffini, Madalina-A Mitran, and Daniela Zaharie. 2023. The importance of being constrained: Dealing with infeasible solutions in differential evolution and beyond. *Evolutionary Computation* (2023), 1–46.

[20] Jakub Kudela. 2023. The Evolutionary Computation Methods No One Should Use. *arXiv preprint arXiv:2301.01984* (2023).

[21] Manuel López-Ibáñez, Juergen Branke, and Luís Paquete. 2021. Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization* 1, 4 (2021), 1–21.

[22] Zhongqiang Ma, Guohua Wu, Ponnuthurai Nagaratnam Suganthan, Aijuan Song, and Qizhang Luo. 2023. Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. *Swarm and Evolutionary Computation* 77 (2023), 101248.

[23] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

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

[25] Laurent Meunier, Herilalaina Rakotoarison, Pak-Kan Wong, Baptiste Rozière, Jérémy Rapin, Olivier Teytaud, Antoine Moreau, and Carola Doerr. 2022. Black-Box Optimization Revisited: Improving Algorithm Selection Wizards Through Massive Benchmarking. *IEEE Trans. Evol. Comput.* 26, 3 (2022), 490–500. https://doi.org/10.1109/TEVC.2021.3108185 Free version available at https://arxiv.org/abs/2010.04542.

[26] Jérémy Rapin and Olivier Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.

[27] Kenneth Sörensen. 2015. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research (ITOR)* 22 (2015), 3–18.

[28] Kenneth Sörensen, Marc Sevaux, and Fred W. Glover. 2018. A History of Metaheuristics. In *Handbook of Heuristics*, Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende (Eds.). Springer, 791–808. https://doi.org/10.1007/978-3-319-07124-4_4

[29] Ryoji Tanabe and Alex S Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)*. IEEE, 1658–1665.

[30] Niki van Stein, Diederick Vermetten, Anna V. Kononova, and Thomas Bäck. 2024. Explainable Benchmarking for Iterative Optimization Heuristics. arXiv:2401.17842 [cs.NE]

[31] Nguyen Van Thieu and Seyedali Mirjalili. 2023. MEALPY: An open-source library for latest meta-heuristic algorithms in Python. *Journal of Systems Architecture* 139 (2023), 102871.

[32] Luis Velasco, Hector Guerrero, and Antonio Hospitaler. 2023. A Literature Review and Critical Analysis of Metaheuristics Recently Developed. *Archives of Computational Methods in Engineering* (2023), 1784–1886. https://doi.org/10.1007/s11831-023-09975-0

[33] Diederick Vermetten, Fabio Caraffini, Anna V. Kononova, and Thomas Bäck. 2023. Modular Differential Evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, Sara Silva and Luís Paquete (Eds.). ACM, 864–872. https://doi.org/10.1145/3583131.3590417

[34] Diederick Vermetten, Carola Doerr, Hao Wang, Anna V Kononova, and Thomas Bäck. 2024. Reproducibility files and additional figures. (2024). Code and data repository (Zenodo): doi.org/10.5281/zenodo.10561215 Figure repository (Figshare): doi.org/10.6084/m9.figshare.25060151.

[35] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Online selection of CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 951–959. https://doi.org/10.1145/3321707.3321803 Free version available at https://arxiv.org/abs/1904.07801.

[36] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. 2020. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM, 654–662. https://doi.org/10.1145/3377930.3390189 Free version available at https://arxiv.org/abs/2006.06586.

[37] Grega Vrbančič, Lucija Brezočnik, Uroš Mlakar, Dušan Fister, and Iztok Fister. 2018. NiaPy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software* 3, 23 (2018), 613.

[38] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. 2022. IOHanalyzer: Detailed Performance Analyses for Iterative Optimization Heuristics. *ACM Trans. Evol. Learn. Optim.* 2, Article 3 (2022). https://doi.org/10.1145/3510426 Free version available at https://arxiv.org/abs/2007.03953.

[39] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research* 32 (2008), 565–606.