Using Machine Learning Methods to Assess Module Performance Contribution in Modular Optimization Frameworks

Ana Kostovska, Diederick Vermetten, Peter Korošec, Sašo Džeroski, Carola Doerr, Tome Eftimov

Using Machine Learning Methods to Assess Module Performance Contribution in Modular Optimization Frameworks

Ana Kostovska ana.kostovska@ijs.si
Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, 1000, Slovenia

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

Peter Korošec peter.korosec@ijs.si
Computer Systems Department, Jožef Stefan Institute, Ljubljana, 1000, Slovenia

Sašo Džeroski saso.dzeroski@ijs.si
Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, 1000, Slovenia

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

Tome Eftimov tome.eftimov@ijs.si
Computer Systems Department, Jožef Stefan Institute, Ljubljana, 1000, Slovenia

Abstract

Modular algorithm frameworks not only allow for combinations never tested in manually selected algorithm portfolios, but they also provide a structured approach to assess which algorithmic ideas are crucial for the observed performance of algorithms. In this study, we propose a methodology for analyzing the impact of the different modules on the overall performance. We consider modular frameworks for two widely used families of derivative-free black-box optimization algorithms, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and differential evolution (DE). More specifically, we use performance data of 324 modCMA-ES and 576 modDE algorithm variants (with each variant corresponding to a specific configuration of modules) obtained on the 24 BBOB problems for 6 different runtime budgets in 2 dimensions. Our analysis of these data reveals that the impact of individual modules on overall algorithm performance varies significantly. Notably, among the examined modules, the elitism module in CMA-ES and the linear population size reduction module in DE exhibit the most significant impact on performance. Furthermore, our exploratory data analysis of problem landscape data suggests that the most relevant landscape features remain consistent regardless of the configuration of individual modules, but the influence that these features have on regression accuracy varies. In addition, we apply classifiers that exploit feature importance with respect to the trained models for performance prediction and performance data, to predict the modular configurations of CMA-ES and DE algorithm variants. The results show that the predicted configurations do not exhibit a statistically significant difference in performance compared to the true configurations, with the percentage varying depending on the setup (from 49.1% to 95.5% for modCMA and 21.7% to 77.1% for DE).

Keywords

Evolutionary computation, CMA-ES, DE, modular algorithm frameworks

1 Introduction

Black-box optimization refers to optimization for problems where the structure of the objective function is unknown, unexploitable, or non-existent (Alarie et al., 2021). In such cases, information about the optimization problem is gathered by evaluating the objective function for newly sampled candidate solutions, without making use of knowledge of the underlying structure or characteristics of the problem (Molina et al., 2018). Iterative metaheuristic optimization algorithms are especially well-suited to tackle such problems as they search for the optimal solution by iteratively querying the objective function with different inputs (i.e., *solution candidates*), and they use only this information to steer their search towards the most promising regions of the search space. Some commonly used iterative metaheuristics include local search algorithms (Hoos and Stützle, 2004), evolutionary algorithms (Eiben and Smith, 2015), genetic algorithms (Kramer, 2017), particle swarm optimization (Kennedy and Eberhart, 1995), and ant colony optimization (Dorigo et al., 2006), among many others.

Many state-of-the-art algorithms are claimed to have been originally inspired by natural processes such as evolution and swarm intelligence (Hussain et al., 2019). Fueled by the variance in the performance of the algorithms on different problem types, researchers continue to seek inspiration from nature and employ diverse metaphors to develop and refine these techniques. However, a recent call for action by the evolutionary computation scientific community has highlighted three significant concerns related to metaphor-based metaheuristics (Aranha et al., 2022). Firstly, the usefulness of metaphors in metaheuristics is questionable, as many ”novel” algorithms inspired by metaphors often lack scientific justification and oversimplify or modify the metaphor to resemble an optimization process, making them differ greatly from their original inspiration. Secondly, there is a lack of originality, with researchers often rediscovering concepts that have already been published in earlier studies (under different names). Finally, the experimental validation and comparisons of these algorithms are often biased, with improper comparisons made between novel and non-state-of-the-art algorithms on benchmark problem instances that are under-representative of the diversity in the problem space. These issues highlight the need to develop novel approaches that can be used to better understand the behavior of metaphor-based metaheuristics (and metaheuristics in general) in order to identify genuine contributions to the field.

A commonly used method to understand the behavior of algorithms is the assessment of their performance through benchmarking and statistical analyses. Typically, this involves reporting the average performance across a selected set of benchmark problems (Hansen et al., 2020; Eftimov et al., 2017). However, this approach has faced criticism for its limitations in accurately interpreting algorithm behavior and its inability to generalize to new problems (Hooker, 1994, 1995; Hall and Posner, 2010). Furthermore, in these statistical analysis approaches, algorithms are treated as black-boxes much like optimization problems, hence, it is challenging to draw any conclusions about the characteristics of the algorithms that contribute most to their performance.

Another approach to understanding metaheuristics is the development of classification systems and taxonomies that try to categorize these algorithms based on their underlying mechanisms, search strategies, and other relevant factors (Stork et al., 2022; Lones, 2020; Stegherr et al., 2022). Unlike statistical approaches that treat algorithms as black boxes, these classification systems aim to provide a structured way of describing metaheuristics and help researchers identify similarities and differences between different algorithms. However, one limitation of these classification systems is the lack of connection between the algorithms and the optimization problems they are designed to solve, as well as the performance they exhibit on these optimization problems. Without this connection, it can be challenging to understand the performance of the algorithms on specific problem instances.

To overcome these limitations, new methods for assessing algorithm behavior are needed. These methods should consider the characteristics of the algorithm, the landscape characteristics of the problems, and the interaction between the two in terms of their influence on performance behavior. By understanding these factors, we can develop more effective algorithms that perform well on a range of problem instances.

One promising approach for improving the assessment of algorithm behavior is to use modular optimization algorithm frameworks (Dreo et al., 2021; de Nobel et al., 2021a; Andersen et al., 2022; Boks et al., 2020). These frameworks provide a flexible and modular way to design and evaluate metaheuristic algorithms. The idea is to break down the algorithm into smaller components that can be easily modified and combined to create new algorithms. By using a modular approach, researchers can better understand how individual components contribute to the algorithm's overall performance and identify areas for improvement. In these frameworks, 'modules' essentially represent the operators in optimization algorithms. For clarity and consistency throughout this paper, we will use the term 'module' instead of 'operator' in the context of modular frameworks. Modular optimization algorithm frameworks can also provide a way to bridge the gap between algorithm behavior and the optimization problems they are designed to solve. By designing algorithms as collections of interchangeable components, researchers can test different combinations of components on a variety of problem instances.

In this study, we use modular optimization algorithm frameworks to assess the different algorithmic ideas that were proposed in the literature. Our analysis is focused on examining each module individually, reflecting the common practice of proposing algorithmic ideas in isolation. Exploring the interactions between these modules and their collective impact on performance is an interesting aspect, yet it remains outside the scope of our current research.

We focus on the analysis of two different classes of metaheuristics: Differential Evolution (DE) (Storn and Price, 1997) and Covariance Matrix Adaptation Evolution Strategies (CMA-ES) (Hansen and Ostermeier, 1996). We use their decomposed versions on basic components/modules that are available in the modCMA-ES (de Nobel et al., 2021a) and modDE (Vermetten et al., 2023a) modular frameworks, respectively.

Our contributions and key findings: In this paper, our main contribution is the proposal of an empirical workflow for understanding the impact of modules on the performance of DE and CMA-ES algorithm. We analyze 324 modCMA-ES and 576 modDE algorithm variants across 24 BBOB problems to: (i) Evaluate the effect of individual modules on overall algorithm performance through analysis of performance data; (ii) Train ML regression models to predict algorithm performance, with a focus on understanding how problem landscape features impact these predictions. This approach provides an explainable ML model, linking feature importance directly to the model outcomes; and (iii) Train classifiers that use performance and landscape feature importance data to predict algorithm module configurations. High prediction accuracy signals variability in the performance data w.r.t. to the modules, suggesting that higher accuracy reflects greater variability or a stronger impact of the module on performance. Conversely, lower accuracy suggests a module's configuration has a minimal impact on overall performance.

Our analyses suggest that, among the CMA-ES modules, the elitism module has the most significant influence on performance, whereas the local restart module has the smallest effect, particularly for shorter runtime budgets. For DE, the linear population size reduction module has the greatest impact, whereas the mutation reference and adaptation method modules have relatively minor effects. However, we also observe that the results of our methodology can be inconclusive for some of the modules.

Interestingly, we find that the *set* of landscape features that are most relevant for accurate regression models that predict performance does *not* depend on the configuration of the modules, indicating that feature selection does not necessarily need to be tailored to the specific configuration. Finally, we have trained classifiers to predict modular configurations, with better performance gains achieved using performance-based meta-representations.

Extension: This study builds upon our previous work published at the Genetic and Evolutionary Computation Conference (GECCO 2022) (Kostovska et al., 2022b) on landscape feature importance in predicting the performance of modular CMA-ES variants. Initially focusing on 40 CMA-ES variants and two modules (*elitism* and *step size adaptation*), we now examine a broader scope encompassing 324 CMA-ES variants, obtained by changing the configurations of six modules. We also extend our analysis to 576 DE variants, generated by changing the configurations of seven modules. Furthermore, we investigate how the different CMA-ES and DE modules individually affect the algorithm's overall performance. Previously, we used single-output classifiers and problem landscape-based meta-representations to predict modular algorithm configurations. In this study, we have extended our methodology by incorporating both single and multi-output classifiers and integrating problem landscape and performance-based meta-representations. Moreover, we have conducted a statistical analysis of the resulting classification predictions to determine if there is a performance difference between the true and predicted configurations.

Outline: The paper is structured as follows. In Section 2, we review related work on empirical performance analysis of modular optimization algorithms, automated algorithm performance prediction, and explainable ML. Section 3 presents our methodology for obtaining algorithm meta-representations and using them to predict the algorithm's modular configuration. We describe our experimental design in Section 4. In Section 5, we discuss the key findings and results of our experiments. Finally, in Section 6, we summarize our contributions and outline several directions for future work.

Availability of data and code: Following best practices towards replicability and reproducibility, the full project data and code, as well as figures for all settings, are publicly available (Kostovska et al., 2023b).

2 Related Work

Diverse research has investigated the modular CMA-ES and DE algorithm families in various single-objective learning scenarios. This includes conducting empirical performance analysis of CMA-ES (Vermetten et al., 2023b) and DE (Das and Suganthan, 2010), predicting CMA-ES (Trajanov et al., 2021) and DE (Nikolikj et al., 2022a) algorithm performance, automated algorithm selection (Jankovic and Doerr, 2020), and automated algorithm configuration (Prager et al., 2020; Belkhir et al., 2017).

The empirical performance analysis (Vermetten et al., 2023b; Das and Suganthan, 2010) has focused on providing empirical results through descriptive statistics of the performance achieved on a particular benchmark suite. Another way to compare algorithms' behavior using information from the performance space is to use *performance2vec* meta-representations (Eftimov et al., 2020). Here, the results obtained by multiple runs of an algorithm instance on a particular problem are averaged and stored as a vector representation that consists of the results for all benchmark problems. Further, the similarity between algorithm instances is assessed as the similarity between the vector representations obtained by using *performance2vec*.

The studies performed in automated algorithm performance prediction allow us to develop an explainable ML predictive model. For this purpose, landscape properties (Mersmann et al., 2011) of the problem instances are used as input features to train an ML predictive model that links them to the performance of the algorithm achieved after some function evaluations. Further, by applying post-hoc explainable techniques, the contribution of each landscape feature to the accuracy of performance prediction can be analyzed. Recently, the SHAP (Rozemberczki et al., 2022) feature ranking method has been explored for such analyses, since it provides explanations both at a *global* level (i.e., all benchmark problem instances) and at a *local* level (i.e., per problem instance). The SHAP explanations can be used for algorithm behavior meta-representation that facilitates the capture of the interactions between the problem landscape properties and the performance of the algorithm instance. These meta-representations have been used with unsupervised techniques to find similar groups of algorithm behavior of CMA-ES (Trajanov et al., 2021) and DE (Nikolikj et al., 2022a) configurations.

The mentioned studies integrate into a broader range of research that aims to understand the behavior of modular CMA-ES and modular DE. However, despite significant efforts in this direction, most of the studies that focus on automated algorithm performance prediction and selection treat the CMA-ES or DE configurations as black boxes, without exploring the impact of the individual modules on the final performance of a configuration. While some studies have used time-series features calculated from the global state variables to classify isolated CMA-ES modules (de Nobel et al., 2021b), there is no information on how these features are linked to each module separately. Another study (Prager et al., 2020) has investigated a problem instance-based configuration model that selects optimal CMA-ES modules using landscape features of problem instances but does not provide any insight into the importance of the landscape features.

Modular algorithm components have also been investigated in multi-objective optimization. Bezerra et al. (2015) focus on the automatic design of novel multi-objective evolutionary algorithms (MOEAs) through the utilization of a conceptual framework encompassing various MOEA components. However, the study does not investigate the impact of each of those modules on the overall performance of the algorithm, nor does it provide insight into the importance of problem landscape features.

A purely performance-oriented view on the modular algorithm framework was taken by Aziz-Alaoui et al. (2021), where a modular suite of pseudo-Boolean optimization algorithms is implemented within the ParadisEO framework (Cahon et al., 2004) and tuned on a collection of W-model problem instances (Weise and Wu, 2018; Doerr et al., 2020) using the irace algorithm configurator (López-Ibáñez et al., 2016). Here, the goal is to identify module combinations that work well together, rather than to explore their complementarity.

3 Methodology

Our methodology comprises three main components: (i) generating meta-representations of the modular algorithms (described in Section 3.1); (ii) exploratory analysis, investigating the impact of the modules on the performance and investigating the importance of the landscape features for algorithm performance prediction (Section 3.2); and (iii) using the learned meta-representations in a supervised classification task to predict the modular configuration of different algorithm instances (Section 3.3).

3.1 Generating meta-representations of modular algorithms

We investigate two types of algorithm meta-representations, *performance-based* and *Shapley-based*.

3.1.1 Performance-based meta-representations

Performance-based meta-representations (Eftimov et al., 2020) rely solely on performance data, enabling us to develop an understanding of how the different modules contribute to the performance of the algorithm variant. To obtain this data, we execute each modular algorithm variant (i.e., algorithm instance) on a range of problem instances from diverse classes.

Considering the stochastic nature of the algorithms, to obtain reliable estimates of the performance of each variant on each problem instance, we conduct r independent runs of the algorithm variant. In each run, we measure the *precision*, i.e., the absolute difference $f(x^{\text{best}}) - f^*$, between the best solution x^{best} found by the algorithm in the considered run and the global optimum $f^* := \inf_x f(x)$. The solution quality (or performance) for instance j in class i, referred to as q_{ij}, is determined as the median of these precision values.

To summarize the algorithm variant's performance on a problem class level, we calculate the mean $p_i = \frac{1}{m}\sum_{j=1}^{m} q_{ij}$ of the solution qualities q_{ij} across the m instances in class i.

Finally, the overall performance of an algorithm instance across n classes is summarized in an n-dimensional vector $P = (p_1, p_2, ..., p_n)$.

3.1.2 Shapley-based meta-representations

Shapley-based meta-representations consist of problem landscape feature importance scores derived from regression models that predict algorithm performance. These scores, known as Shapley values, quantify the marginal contribution of each input feature (in our case problem landscape features) to the model's predictions (Molnar, 2020). Shapley values have been widely used as an explainability technique in ML (Chen et al., 2022; Kumar et al., 2020). Unlike classical ML feature importance approaches that provide global importances on a model-level, Shapley provides feature importances on a local level, for each prediction. This local interpretability aspect provides valuable insights into the model's decision-making process, enhancing its explainability. For calculating the Shapley values, we use the SHAP (SHapley Additive exPlanations) algorithm (Lundberg and Lee, 2017).

To construct these meta-representations, we first train regression models for performance prediction for each variant of the modular algorithms, separately. We consider a portfolio of problem classes with size n and m instances of each problem class, resulting in a total of $n \times m$ problem instances. Each problem instance is represented as a vector of ℓ problem landscape features, $(x_1, x_2, ..., x_\ell)$, which serve as input for training the regression models. The target output y that we aim to predict is the algorithm's performance within a fixed budget of function evaluations, as detailed in Section 3.1.1.

Table 1: Illustrative example of groups of CMA-ES algorithm variants when we investigate the impact of the elitism module on the algorithm's performance.

	Elitism	Base sampler	Step-size adaptation
1	True	Gaussian	CSA
2	True	Gaussian	PSR
3	True	Sobol'	CSA
4	True	Sobol'	PSR
5	True	Halton	CSA
6	True	Halton	PSR

(a) Algorithm variants with elitism

	Elitism	Base sampler	Step-size adaptation
7	False	Gaussian	CSA
8	False	Gaussian	PSR
9	False	Sobol'	CSA
10	False	Sobol'	PSR
11	False	Halton	CSA
12	False	Halton	PSR

(b) Algorithm variants without elitism

After training the regression model for performance prediction, we calculate the Shapley values of the landscape features. Applying the Shapley value calculation on the regression models that predict the performance of each algorithm instance separately gives us the Shapley-based meta-representations as an ℓ-dimensional vector, $(s_1, s_2, ..., s_\ell)$. We need to point out here that the Shapley meta-representations are model-specific and depend on the ML algorithm used for learning the predictive model.

3.2 Exploratory analysis using the meta-representations

We use the learned algorithm meta-representations in two types of exploratory analysis: (1) to investigate the impact of a module's configuration on the algorithm's performance and (2) to investigate the importance of the landscape features when predicting the algorithm's performance across the different module configurations.

3.2.1 The impact of module configuration on algorithm performance

Here, we make use of the performance-based meta-representations. First, to investigate the impact of module configuration on algorithm performance, from a selected set of algorithm modules and their configurations, we generate all possible configurations of the algorithm variants that we will investigate. Then, the different algorithm variants are grouped with respect to a given module to observe whether there are some differences in the performance when we change the module configuration and introduce some specific structural changes to the algorithm.

Consider as an example the modular CMA-ES algorithm (de Nobel et al., 2021a). This algorithm has multiple configurable modules, but in this illustrative example, for simplicity, we focus on three: elitism (which can take values of either true or false), the base sampler (which offers different sampling techniques such as Gaussian, Sobol', and Halton), and the step size adaptation mechanism (which includes Cumulative Step Size Adaptation (CSA) and Step Size Adaptation with Population Success Rule (PSR)). By combining the settings of these three modules, we can create a total of 12 different algorithm variants shown in Table 1. To assess the impact of elitism, we divide the configurations into two distinct groups: one with elitism activated (elitism = True) and another without it (elitism = False). This division allows us to analyze and compare the performance of the algorithm variants under different settings, e.g., for elitism.

For each group, we visualize the distribution of the achieved performance on all problem instances and problem classes. Differences in these distributions would indicate that certain modular configurations perform better/worse on the overall problem instance portfolio. We repeat this process for the remaining modules. Additionally, the same analysis can be performed at problem class level to investigate whether there are differences in performance in the different problem classes.

3.2.2 The importance of the landscape features in algorithm performance prediction for the different module configurations

Compared to the performance-based meta-representations, the Shapley-based ones come with the benefit that they can be employed in an exploratory analysis pipeline where we can investigate the importance of the landscape features across the different modules and across the different configurations of a given module. To this end, we perform the same process of grouping the algorithm variants as described in Section 3.2.1. We then calculate the importance of the landscape features for each group separately and average them across all problem instances. We repeat this process for the remaining modules. This approach facilitates the exploratory analysis of the effect each of the modules has on the final performance of the algorithm. Furthermore, trends in the landscape space can be observed. More specifically, some problem landscape features can be found to hold more predictive value than the rest by observing the SHAP values across the different problems (in multiple dimensionalities) for different budgets.

3.3 Prediction of a module's configuration of the algorithm instances

The meta-representations (both performance- and Shapley-based) of the modular algorithms variants can be assigned labels that indicate their modular configuration. This labeled data serves as input to train ML classifiers, which predict the configuration of the algorithm's modules. These classifiers are beneficial, for example, in cases where we have the performance data of an algorithm that is achieved after some function evaluations on a particular benchmark suite, but we don't have information about the configuration of the algorithm.

By using its meta-representation we may be able to identify a modular configuration with similar performance behavior. This may help us in a lot of studies for which the performance data is publicly available, but details about the tested configurations are missing. For example, if we have CMA-ES performance data, by using the learned classifiers we can identify a modular CMA-ES configuration with similar behavior.

To test the power of the classifiers, we use the meta-representations of each modular configuration, and we use the classifiers to predict the modules that are activated with their values. Further, we report the F1 score (macro F1 score in the case of multi-class classification) of the predictions across all modules, problem dimensions, and different cut-off budgets. However, the classifiers may make wrong predictions for the configuration, and the prediction may differ from the true configuration in one or several modules. The wrong predictions affect the performance of the classifier, but the predicted configuration and the true one may still have similar behavior. To evaluate this, we perform a statistical analysis based on hypothesis testing including the raw performance data for the true and the predicted modular configuration. For this purpose, we use the Deep Statistical Comparison (DSC) approach (Eftimov et al., 2017) that ranks the true and the predicted configuration for each problem instance separately, by comparing the distribution of their raw performance data (for each problem instance separately). The ranked data obtained for the true and the predicted configuration across all benchmark problem instances is further analyzed by the Wilcoxon signed-ranks test to find if there is a statistically significant difference in the performance of the true and predicted configuration on the selected benchmark suite.

4 Experimental Design

In this section, we provide details on the experimental setup, which consists of several components. We describe our problem portfolio, problem landscape data, algorithm portfolio, and algorithm performance data. Additionally, we provide information on the regression models for algorithm performance prediction and the classifiers for the prediction of the modular configuration of each algorithm instance.

4.1 Problem instance portfolio

The problem instance portfolio consists of the 24 single-objective, noiseless black-box optimization problems sourced from the BBOB benchmark suite (Hansen et al., 2009) of the COCO benchmark environment (Hansen et al., 2020). Multiple instances of each BBOB problem can be generated by using linear and non-linear transformation processes. These involve adding an offset, rotating the axes, and scaling the coordinates of each basic function. The BBOB benchmark suite already contains multiple instances of each problem. In this study, we consider the first 5 instances of each of the 24 BBOB functions, both with dimension $D = 5$ and $D = 30$. This results in two separate problem instance portfolios, one for each dimension, with each portfolio containing a total of 120 problem instances.

4.2 Landscape features

For representing the problem landscape, we utilize the ”cheap” exploratory landscape analysis (ELA) features implemented in the R package `flacco` (Kerschke and Trautmann, 2016). The ELA features are used as numerical vector representations of the problem instances that capture the landscape characteristics of optimization problems. We considered a total of 46 different ELA features, which were not calculated from scratch, but reused from Renau et al. (2020). The selected ELA features were calculated by using the Sobol' sampling strategy on a sample of size $100D$ on a total of 100 independent repetitions. To represent the landscape of each problem instance, we calculate the median value for each feature over the 100 independent repetitions.

It is worth mentioning that we deliberately allocated a substantial sample size for ELA computation to eliminate the effects of noisy feature evaluations. Additionally, we do not perform feature selection, even though it has been demonstrated to improve results in performance prediction tasks (Renau et al., 2021). However, we anticipate that our findings will hold for other types of features as well, based on previous work (Jankovic et al., 2021a; Nikolikj et al., 2022b).

4.3 Algorithm portfolio

We examine two black-box optimization algorithms that have modular implementations available, namely CMA-ES and DE. For CMA-ES, we utilize the modCMA-ES framework (de Nobel et al., 2021a), which encompasses various versions of the core algorithm. These modifications include changes in the sampling distribution (such as mirrored or orthogonal sampling), weighting schemes for recombination, and restart strategies, to name a few. This modular structure allows for the creation of at least $36\,288$ configurations of CMA-ES, and additionally provides access to a large set of control parameters (population size, update rates,. . .).

We utilized the modDE (Vermetten et al., 2023a) package[1] for DE. This package provides a diverse array of mutation mechanisms and modules for selecting the base component, the number of differences included, and the use of an archive for some

[1]modDE package (version 0.0.1-beta) accessible at `https://github.com/Dvermetten/ModDE`

Table 2: The complete list of modCMA-ES modules and their respective parameter space yielding a total of 324 algorithm configurations.

Module	Parameter space
Elitist	True, False
Mirrored_sampling	None, mirrored, mirrored pairwise
base_sampler	gaussian, Sobol', halton
weights_option	default, equal, (1/2)^λ
local_restart	None, IPOP, BIPOP
step_size_adaptation	csa, psr

Table 3: The complete list of modDE modules and their respective parameter space yielding a total of 576 algorithm configurations.

Module	Parameter space
mutation_base	rand, best, target
mutation_reference	None, pbest, best, rand
mutation_n_comps	1, 2
use_archive	True, False
crossover	bin, exp
adaptation_method	None, shade, jDE
lpsr	True, False

of the difference components. Additionally, the package enables the usual crossover mechanisms and incorporates update mechanisms for internal parameters based on several state-of-the-art DE versions. In total, this package allows for the creation of at least $1\,474\,560$ configurations of DE.

4.4 Performance data

Due to the computational infeasibility of collecting data for all possible combinations of modular CMA-ES and modular DE algorithms, we opted to use a subset of, specifically 324 algorithm variants for modular CMA-ES and 576 variants for modular DE. The performance data for this subset of algorithm variants is taken from Kostovska et al. (2023a). We show the modules and parameter spaces used for CMA-ES and DE in Table 2 and Table 3, respectively. To obtain the algorithm variants, we created a Cartesian product of the modules and the selected module parameter spaces.

To evaluate the performance of each algorithm variant, 10 independent runs have been conducted and the median objective function value has been recorded for each problem instance. All experiments make use of the IOHexperimenter module (de Nobel et al., 2024) of the IOHprofiler benchmarking environment (Doerr et al., 2018). Our objective function measures the precision of the algorithm's solution, i.e., the distance to the optimum, within a fixed budget of function evaluations. We considered six different budget values, $B \in \{50D, 100D, 300D, 500D, 1\,000D, 1\,500D\}$, where D is the problem dimensionality. We report the best precision achieved by each algorithm variant at the different cut-off budgets for the $5D$ and $30D$ problem instance portfolios. The population size for both CMA-ES and DE is set to $4 + \lfloor 3\log(D) \rfloor$.

4.5 Regression models for algorithm performance prediction

In this study, we train regression models for algorithm performance prediction as part of the pipeline of obtaining Shapley-based algorithm meta-representations. Previous studies have investigated the use of ML in algorithm performance prediction, including the use of Random Forest (RF) regression models (Muñoz et al., 2012; Collautti

Table 4: Parameters of the RF approach and their corresponding values considered in the grid search.

Hyperparameter	Search space
n_estimators	$[10, 50, 100, 500,]$
max_features	$['auto', 'sqrt', 'log2']$
max_depth	$[4, 8, 15, None]$
min_samples_split	$[2, 5, 10]$

et al., 2013; Kerschke and Trautmann, 2019; Kostovska et al., 2022a). RF, an ensemble-based decision tree method, is thoroughly described in the seminal work by Breiman (2001). In our work, we employ the RF approach to learn performance prediction regression models, as they have been shown to provide promising results in this context (Jankovic et al., 2021b; Trajanov et al., 2021) and we tune their hyperparameters. For training the models we use the RF algorithm as implemented in the Python package `scikit-learn` (Pedregosa et al., 2011).

To ensure optimal results, we trained separate regression models (single-output models) for each modular variant. This decision was based on findings by Trajanov et al. (2021), which showed that multi-output models (models that predict the output for several algorithm instances simultaneously) did not demonstrate performance gains compared to single-output models.

For learning the performance prediction models, a vector of 46 ELA features is used to describe each problem instance. Our objective is to predict the precision, i.e., the distance to the optimum that each algorithm in the portfolio will attain on a problem instance, given a fixed budget of function evaluations and problem dimensionality. In this study, we log10-transform the target variable (the median of the 10 independent runs) as it has been shown to improve the performance of the learned predictive models when the target variable is the distance to the optimum (Jankovic and Doerr, 2020). We also cap the target variable to 10^{-8} prior to performing the logarithmic transformation.

Hyperparameter tuning and model evaluation. To assess the learned ML models' performance, we use a nested cross-validation (CV) technique that involves two stages. In the outer loop, we partition the data into training and testing sets, while the inner loop determines the optimal parameters of the ML method. This evaluation approach may require significant computational resources, but yields more reliable estimates of the model's generalization ability as compared to traditional train/val/test data splitting or standard CV (Varma and Simon, 2006; Bates et al., 2023).

To implement the outer loop, we apply a leave-one-group-out CV, which segments the data into groups/folds based on the unique ID of each problem instance. Since our study involves the first 5 instances of each of the 24 BBOB problems, we create 5 folds by assigning 4 for training and 1 for testing. We repeat this process five times, each time selecting a different fold for testing while using the remaining four for training.

The inner loop adopts a grid search approach to tune the parameters and selects the optimal ones based on the average performance of the inner CV's holdout folds. A leave-one-group-out CV is applied to the training data (i.e., the four folds) obtained from the outer loop. The R^2 score is used as a performance metric. The parameters chosen for tuning and their corresponding search spaces can be found in Table 4.

After the optimal parameters have been determined, the model is trained on the entire training data, and its performance is assessed using the test set from the outer loop.

4.6 Classification models for predicting/identifying the modular configuration of algorithm variants

To train these classifiers, we use the algorithm meta-representation as input data and apply the RF classifiers implemented in the Python package `scikit-learn` (Pedregosa et al., 2011). We consider two scenarios: (1) Single-output classifiers – we train a classifier for each module separately. Depending on the number of possible configurations for each module, we perform binary classification (when there are 2 possible configurations of the given module, leading to a binary output/target variable) or multi-class classification (when there are more than 2 possible configurations of the given module, resulting in a discrete datatype for the output/target variable), and (2) Multi-output classifiers – we train a single classifier to predict the configuration of all modules simultaneously. Here, the output/target variable is a record of discrete values.

We evaluate both types of classifiers as they have not been studied in this context before. Note that for different problem dimensionalities and cut-off budgets, we train separate classifiers.

In addition, we assess the performance of TabPFN, a pre-trained Transformer model that approximates probabilistic inference for a novel prior in a single forward pass. TabPFN was shown to have fast training time and competitive performance on tabular prediction tasks by (Hollmann et al., 2023) and we use their implementation.

All classifiers are trained using default (hyper-)parameter values. To evaluate the performance of the learned models, we partition the data into training and testing sets using leave-one-group-out cross-validation, which segments the data into train/test folds based on the unique ID of each benchmark problem instance. As a performance indicator, we report the F1 scores of the classifiers.

5 Results and Discussion

Following the methodology described in Section 3 and the experimental protocol given in Section 4, we first perform an exploratory analysis using the algorithm meta-representations (Section 5.1). We then present results on the task of predicting the modular configuration of the algorithm variants from algorithm behavior meta-representations in Section 5.2.

5.1 Exploratory analysis

5.1.1 The impact of the modules on the performance of the algorithms

We investigated how different configurations of modules impact the performance of the CMA-ES and DE algorithms, using performance-based meta-representations in a log-10 scale. The distribution of the precision achieved by different variants of the CMA-ES algorithm on $5D$ problem instances is presented in Figure 1. We tested 6 different modules (elitist, mirrored, base sampler, weights option, local restart, and step size adaptation) across the 6 cut-off budgets. Each violin plot in the figure shows the precision across all CMA-ES algorithm variants that have the same value for a given module.

For instance, there were 324 algorithm variants selected as the Cartesian product of the 6 modules, and 162 algorithm variants had the elitism module activated, while 162 did not. Therefore, the violin plot for 'elitism = true' is based on the precision values of 162 algorithm variants, where the precision value of an algorithm variant is the mean value of the performance-based meta-representations (i.e., the mean value of a numerical vector representation of size 24) in a log-10 scale as detailed in Section 3.

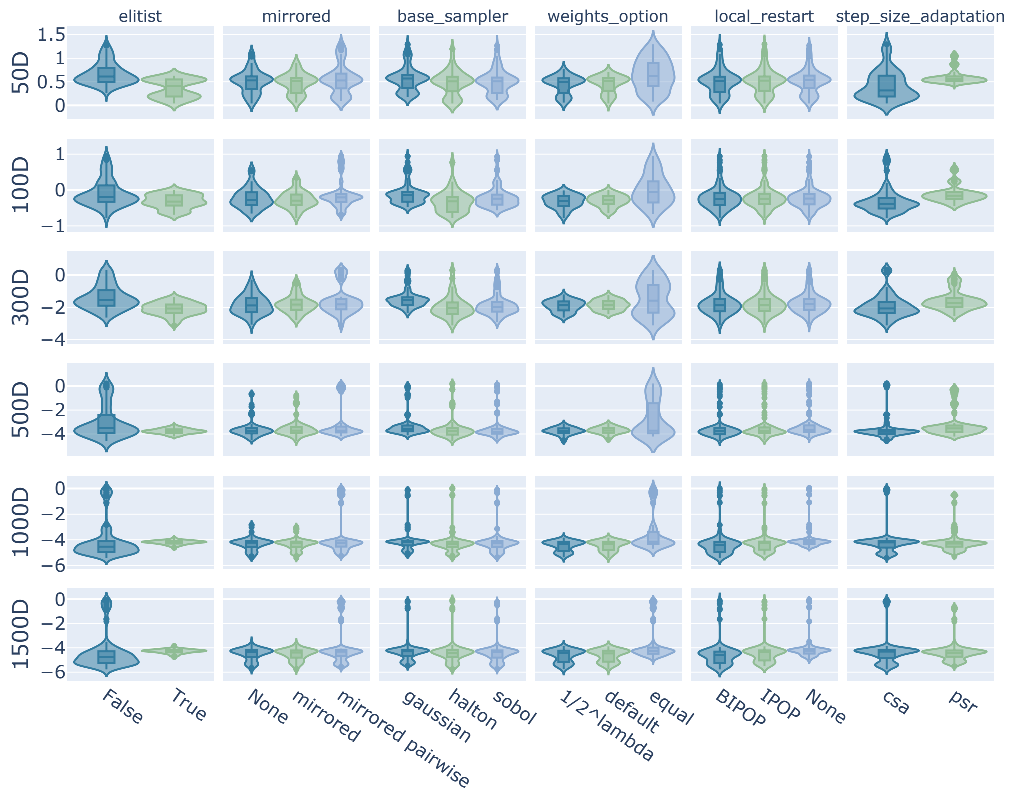


Figure 1: Distribution of the precision achieved by different variants of the **CMA-ES** algorithm on $5D$ problem instances for different modular configurations, across different cut-off budgets. The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

We analyzed the results displayed in Figure 1 and made the following observations: (i) The activation of elitism in the algorithms leads to improved performance for smaller evaluation budgets. As the budget increases, this trend reverses and elitist configurations are overtaken by their non-elitist counterparts; (ii) Algorithm variants that have activated mirrored orthogonal sampling with pairwise selection (mirrored pairwise) demonstrate a longer tail towards poorer performance than those that use mirrored sampling without pairwise selection and those that do not use mirrored sampling at all, although on average they perform similarly; (iii) At the lower budget cut-offs, the Halton sampling showed the best performance, Sobol' sampling came second, and Gaussian sampling demonstrated the worst performance out of the three. As the budget increases, the differences are less evident; (iv) Algorithms with recombination weights set to $(1/2)\hat{}\lambda$ and default weights have similar distributions. Also, all three configuration setups have similar average performance; (v) For the lower budgets, we observe that the local restart module achieves comparable performance for the three modular configurations (BIPOP, IPOP, and no restart) across the different budgets. This makes intuitive sense, as at low budgets the algorithm will not have had a chance to trigger any of the restart criteria. As the budget increases, IPOP and BIPOP local restart

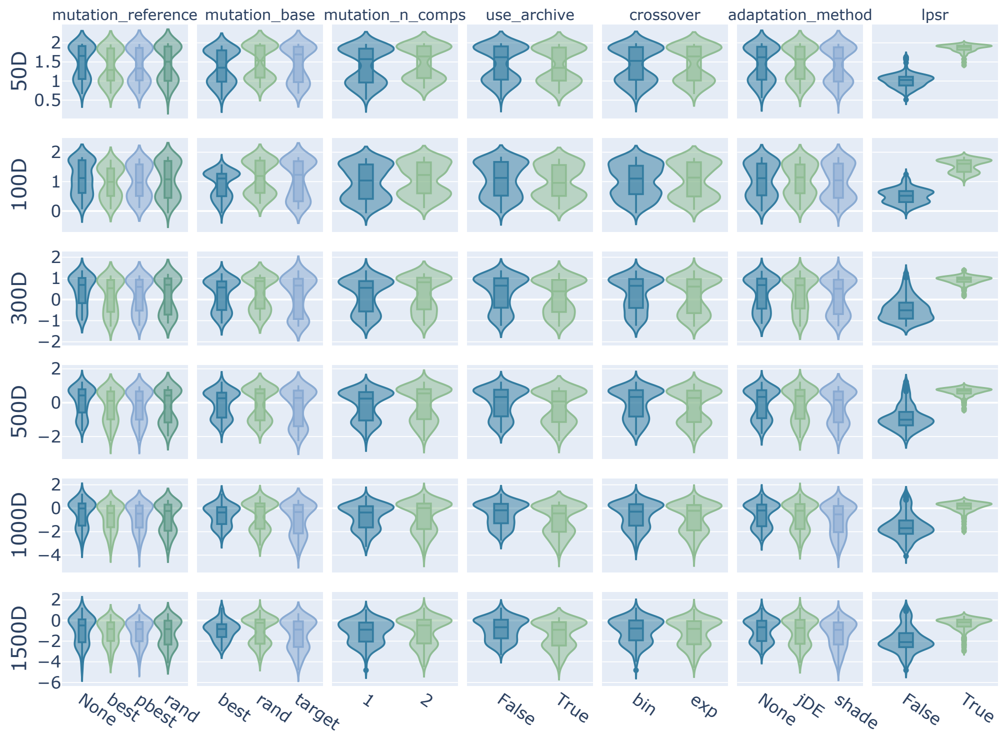


Figure 2: Distribution of the precision achieved by different variants of the **DE** algorithm on $5D$ problem instances for different modular configurations, across different cut-off budgets. The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

techniques show slightly better performance compared to algorithm variants without a restart mechanism, which matches observations made in previous work (Hansen et al., 2010); and (vi) In the case of step size adaptation, for smaller budgets, cumulative step size adaptation (CSA) exhibits better performance than step size adaptation using the population success rule (PSR).

For the DE configurations, we show the same type of visualization in Figure 2. From this figure, we can see that the overall performance differences between DE module options are much smaller than those seen for CMA-ES. The clear exception is the LSPR module that, if enabled, results in much worse performance for smaller budgets. This matches our intuition since LPSR changes the initial population size to $20D$ at the beginning of the search. This much larger initial population size leads to a slower convergence at the beginning of the search. The difference to no-LPSR slowly decreases over time, but it does not manage to overtake it within our maximum budget of $1\,500D$ function evaluations. This observation also seems to suggest that the population size is a critical parameter of DE, which matches previous observations (Piotrowski, 2017). For the other modules, we observe that the mutation base and reference settings, which are more elitist (best and pbest) show improved performance for low budgets, matching the observations for CMA-ES.

We have also conducted the same empirical analysis for the 30D problem instances, and the results are available in our Zenodo repository (Kostovska et al., 2023b). Similar observations can be made for the 30D problem instances as in the case of using the 5D problem instances.

5.1.2 The importance of the ELA features for different modular setups

Following our experimental design, to obtain the Shapley-based meta-representations (i.e., landscape feature importance scores), we first trained separate regression models for each algorithm variant. Table 5 presents the average R^2 scores of the RF and baseline regression models for the CMA-ES and DE algorithm variants, for both $5D$ and $30D$ problems and the six different cut-off budgets. Additionally, Table 6 summarizes the MSE scores for these models. As a baseline, we employ a model that consistently predicts the overall mean algorithm performance. One interesting pattern that can be observed is that the models perform better for the $30D$ problems. This improved performance may be due to variations in the distributions of the target variable across different combinations of budget and dimensionality. The different distribution characteristics, such as skewness, can impact the performance of the RF model. Additionally, in a higher-dimensional space, where the points are spread out, certain ELA features might converge to specific values which can make the data simpler and easier for models to learn from.

Subsequently, we have utilized the SHAP algorithm to determine the feature importance of each of the 46 ELA features at the problem instance level. In the outer loop of the nested cross-validation, we have employed a leave-one-group-out CV validation with five groups (four for training and one for testing). However, we specifically focused on the Shapley values of the training folds, as this data is used to learn the predictive models and provides insight into the algorithms' workings.

To generate a Shapley value for each ELA feature and problem instance, we calculated the value four times (due to each problem instance appearing four times in the training data and once in the testing data) and then took the mean of the four values. Lastly, we averaged the Shapley values for each ELA feature across all problem instances, which gave us a single vector for each algorithm instance. For this purpose, we have leveraged TreeSHAP. TreeSHAP is tailored for tree-based models such as decision trees, random forests, and boosting machines. It is designed to be computationally efficient by exploiting the tree structure for faster calculations, which enables it to manage more complex scenarios effectively. One of the key advantages of TreeSHAP is its consistency property: if a model relies more on a particular feature, the attributed importance of that feature will not decrease, ensuring reliable feature attribution. Alternatively, KernelSHAP can be used for interpreting the impact of features in any model, as it employs a model-agnostic approach. While KernelSHAP offers flexibility across various model types, it comes at the cost of computational efficiency. This makes KernelSHAP less suitable for complex, high-dimensional situations or applications requiring real-time explanations. Given that we are working with tree-based predictive models, TreeSHAP was the appropriate choice for our study. It provided the necessary computational efficiency. The calculation of TreeSHAP is detailed in Lundberg and Lee (2017), where it is demonstrated that Shapley values can be used to interpret model performance regardless of whether the model performs well or poorly.

To investigate the importance of the ELA features, we conduct exploratory analysis by selecting the top K most important features, where K is chosen from the set $\{10, 15, 20\}$. Next, we tally the frequency of appearance for each feature in the top K across all algorithm variants within the same group of modular algorithm variants. This is done by calculating the number of times a feature appears in the top K as indicated by their Shapley values. The resulting value ranges between 0 and the total number of algorithm variants in the group.

Analyzing the results in Figure 3 and through our analysis of feature importance in various other scenarios, we have observed that a similar set of ELA features are the most important predictors of performance, regardless of the algorithm, modular configuration, problem dimensionality, or cut-off budget. This suggests that we can perform feature selection for all algorithms simultaneously, irrespective of their configurations. However, we recommend training separate regression models for each algorithm configuration to obtain the most accurate predictions.

Table 5: The R^2 scores of the RF regression models / R^2 scores of the baseline regression models averaged over the CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the best precision is reached after $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES		DE	
	$5D$	$30D$	$5D$	$30D$
$50D$	0.7577/-0.0072	0.9400/-0.0005	0.8788/-0.0019	0.9403/-0.0009
$100D$	0.7689/-0.0069	0.9179/-0.0008	0.8783/-0.0017	0.9433/-0.0008
$300D$	0.6146/-0.0072	0.8457/-0.0031	0.8587/-0.0016	0.9362/-0.0013
$500D$	0.7045/-0.0055	0.8322/-0.003	0.8368/-0.0024	0.9361/-0.0015
$1000D$	0.7272/-0.0046	0.8072/-0.0029	0.7795/-0.0043	0.9242/-0.002
$1500D$	0.7288/-0.0048	0.8391/-0.0023	0.7508/-0.0051	0.9191/-0.0023

Table 6: The MSE scores of the RF regression models / MSE scores of the baseline regression models averaged over the CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the best precision is reached after $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES		DE	
	$5D$	$30D$	$5D$	$30D$
$50D$	0.7829/4.0248	0.1482/2.461	0.3716/3.383	0.2642/3.7964
$100D$	1.2195/5.1834	0.2692/3.2073	0.4326/3.9106	0.2458/4.225
$300D$	3.9828/8.9112	1.0303/5.8091	0.809/5.6781	0.3055/4.9577
$500D$	4.8498/13.9604	1.2684/6.6078	1.0403/6.1126	0.3803/6.1386
$1000D$	5.2191/15.2566	1.7192/7.9612	1.9462/6.8843	0.5378/7.0673
$1500D$	5.1945/15.198	1.8771/10.9973	2.3355/7.4111	0.634/7.6636

5.2 Predicting the modular configuration of an algorithm using its behavior meta-representation

After exploring the performance and ELA data on which the meta-representations are built, we now analyze whether these meta-representations are powerful enough to predict/identify the corresponding algorithm variant.

First, we compare the single- and multi-output approaches for training classifiers using the RF method. The F1 scores for the classifiers obtained on the test data aggregated across the 2 problem dimensionalities, 5 budgets, and algorithm modules, are listed in Table 7. We have observed that comparable results can be obtained when employing single-output and multi-output RF techniques on both CMA-ES and DE algorithms, using both performance- and Shapley-based meta-representations. However,

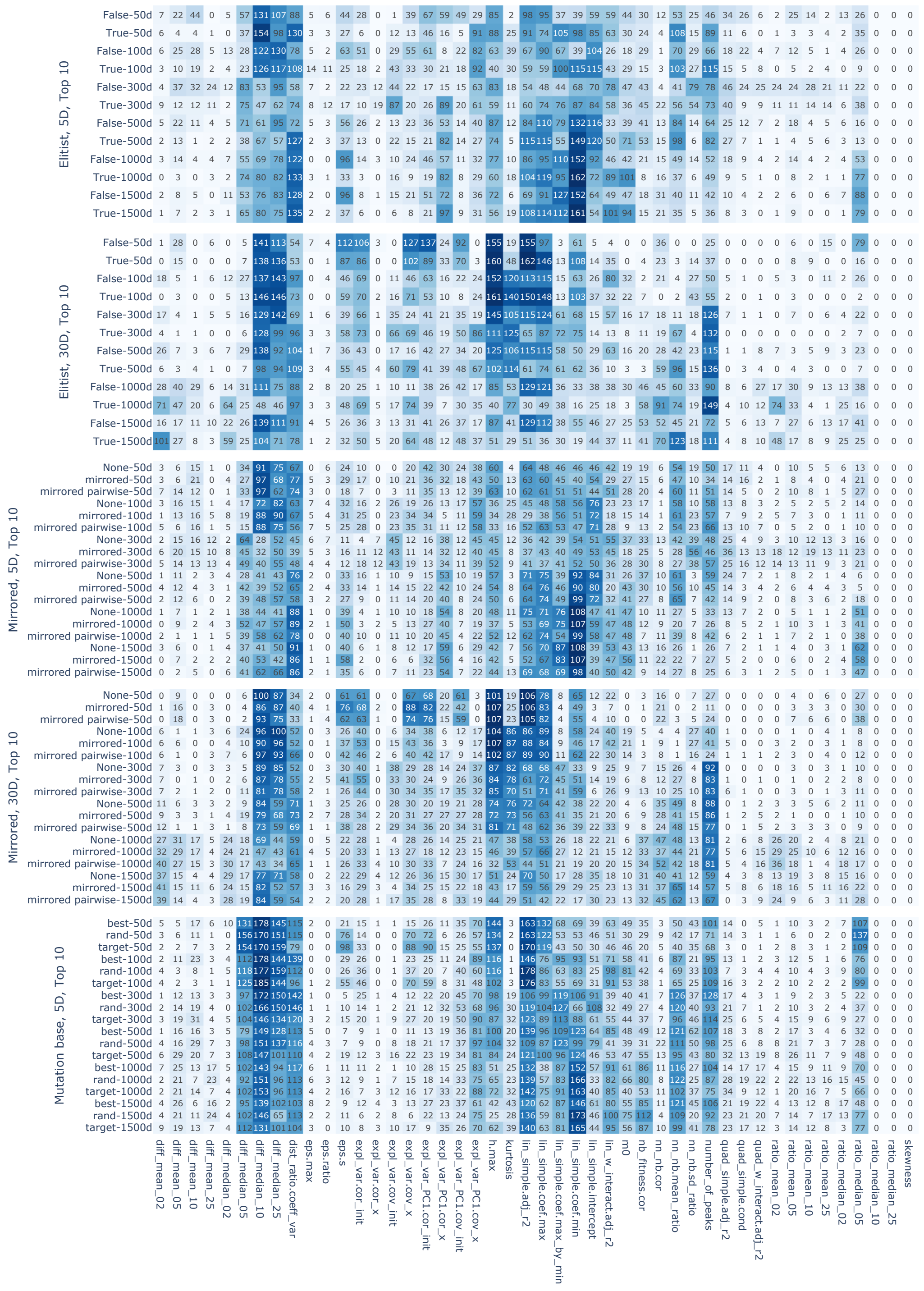
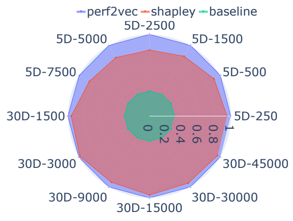


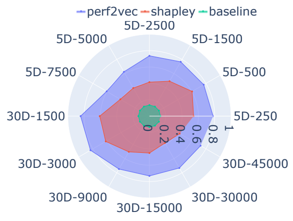
Figure 3: Frequency of appearance of the ELA features as top 10 most important features for performance prediction of two modCMA-ES modules (elitist and mirrored; first four groups) on the 24 BBOB functions in both 5 and 30 dimensions and for six different evaluation budgets $B \in \{50D, 100D, 300D, 500D, 1\,000D, 1\,500D\}$. The fifth group provides the same data for one DE module (mutation base).

Table 7: F1 scores of the single-output RF, multi-output RF, and single-output TabPFN models, computed by averaging over the CMA-ES and DE algorithm variants. The F1 scores are further averaged for both 5 and 30 dimensions, and across the 5 cut-off budgets for function evaluation ($B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$).

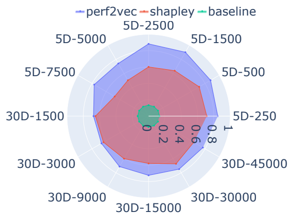
Algo	Performance			Shapley		
	single-output RF	multi-output RF	single-output TabPFN	single-output RF	multi-output RF	single-output TabPFN
CMA-ES	0.794	0.772	0.811	0.623	0.618	0.629
DE	0.758	0.744	0.790	0.603	0.601	0.589



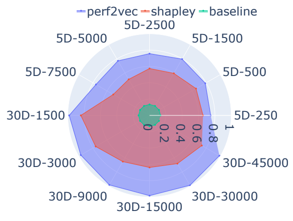
(a) elitist



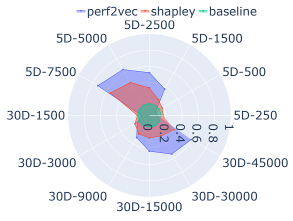
(b) mirrored



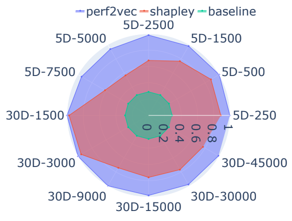
(c) base_sampler



(d) weights_option



(e) local_restart



(f) step_size_adaptation

Figure 4: The F1 scores of the RF classifiers for predicting the modular configuration of the CMA-ES algorithm variants. Results are presented for each CMA-ES module separately, for $5D$ and $30D$ BBOB problem instances, and for 5 different cut-off budgets. The baseline is the majority classifier.

it is worth noting that the multi-output RF approach exhibits slightly inferior performance as compared to single-output RF.

Additionally, we have compared the performance of single-output RF classifiers with TabPFN classifiers. Both classifiers showed similar F1 scores, as indicated in Table 7. For instance, when using Shapley-based meta-representations, the RF classifier's average F1 score for predicting the modular configuration of CMA-ES variants was 0.623, while the TabPFN classifier's F1 score was 0.629, indicating slightly better performance for TabPFN. However, for the DE variants, we observed the opposite situation, with RF classifiers achieving an average F1 score of 0.603 and TabPFN of 0.589.

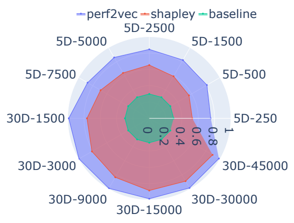
In Table 7, the F1 scores are averaged over all algorithm modules. To further analyze the classifiers trained using the single-output RF method, in Figure 4 we present the F1 scores of the classifiers for each CMA-ES module separately. As a baseline, we use the majority classifier. Figure 4 shows that the highest performance scores are achieved in predicting the setting of the elitist and step-size adaptation modules. The higher F1 scores for the elitist and step-size adaptation modules compared to the other four CMA-ES modules are expected because we only investigated two module options for these two modules, while the remaining four modules used three different module options. By having fewer classes to distinguish between, the classification problem is simplified, making it easier to solve.

The highest F1 scores among the remaining four modules have been observed for the weights option, followed by the base sampler and mirrored. The configuration of the local restart module is the most difficult to predict. In general, all classifiers for predicting the status of each module outperform the baseline across the different modules, problem dimensions, and budgets (see Figure 4).

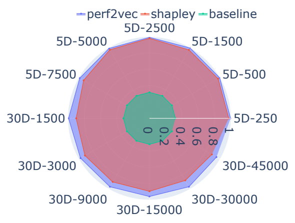
Further, we have observed that the performance-based meta-representations have better predictive power than the Shapley-based meta-representations.

In Figure 5, we show the F1 scores of the RF classifiers for each DE module. For the `mutation_n_comps`, `use_archive`, `crossover`, and `lpsr` modules we have considered two different module options. As can be seen in Figure 5, for these four modules the classifiers have the highest F1 scores, with `lpsr` classifiers performing the best, followed by `crossover`, `mutation_n_comps`, and `use_archive`. As the number of considered modular options increases (three different options for the mutation base and adaptation method modules and four different options for mutation reference), the F1 scores of the classifiers tend to decrease. For both CMA-ES and DE, it is worthwhile to note that the modules that have limited initial impact (local-restart and adaptation mechanism) are indeed more challenging to predict, especially for small budgets. Nevertheless, in all cases, the classifiers outperform the baseline classifier. Furthermore, the RF classifiers that used performance-based meta-representations consistently outperformed those that used Shapley-based meta-representations.

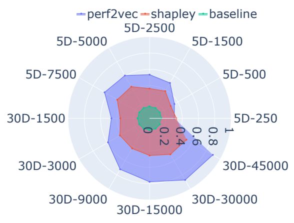
Performance difference between algorithm variants. By combining the predictions of the RF classifiers for all modules, we can predict the modular configuration of the algorithm instance. To judge the effectiveness of this prediction, we analyze the difference in performance between the true and predicted modular configurations. Even though there might be a difference in the configuration, the true and the predicted algorithms may have similar performance behavior. To evaluate this, we use the DSC approach to test for statistical significance in the performance of the true and the predicted configuration across all benchmark problem instances. First, we apply the DSC ranking scheme that ranks the true and the predicted configuration by comparing the distribution of their raw performance data for each problem instance separately. For comparing



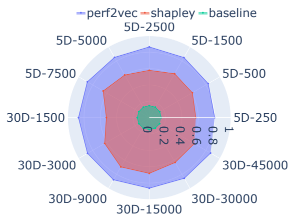
(a) crossover



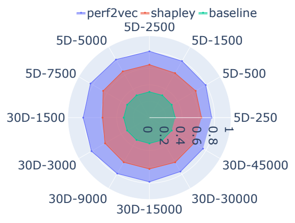
(b) lpsr



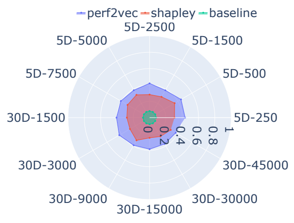
(c) Adaptation method



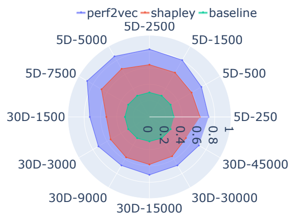
(d) mutation_base



(e) mutation_n_comps



(f) mutation_reference



(g) use archive

Figure 5: The F1 scores of the RF classifiers for predicting the modular configuration of the DE algorithm variants. Results are presented for each DE module separately, for $5D$ and $30D$ BBOB problem instances, and for 5 different cut-off budgets. The baseline is the majority classifier.

the distributions, the two-sample Anderson-Darling test is used by the DSC ranking scheme. Since most of the statistical tests require the independence condition, we have aggregated the rankings per problem class by calculating the average of the DSC rankings obtained for the five problem instances that belong to that problem class. Next, the ranked data is analyzed with a statistical test. The rankings obtained for the 24 problem classes have been analyzed with the Wilcoxon signed-ranks test to find if there is a statistically significant difference (at a p-value of 0.05) in the performance of the true and predicted configuration on the selected benchmark suite. After determining the statistical significance of the difference between each true and predicted algorithm pair, we calculate the percentage of pairs with performance differences that are not statistically significant. Additionally, we generate 5 different random predictions for the modular configuration of each algorithm instance and perform the DSC analysis on them. The results for CMA-ES and DE across different problem dimensions, budgets, and meta-representations are shown in Table 8 and Table 9, respectively.

In both Table 8 and Table 9, we can observe that the percentage of algorithm pairs, consisting of true and predicted configurations, with performance differences that are not statistically significant (based on the predictions generated by our classifiers) is significantly higher as compared to the scenario where predictions are randomly generated for the modular configuration. This affirms the robust predictive capabilities exhibited by our classifiers.

Table 8: The DSC results on the statistical difference in the performance of **CMA-ES** algorithm pairs. Results are reported in the format: percentage of (true, predicted)-pairs with performance differences that are not statistically significant/percentage of (true, random)-pairs with performance differences that are not statistically significant. The numbers in brackets correspond to the standard deviation for the latter percentage since this was computed over 5 independent runs.

Budget	SHAP		Performance	
	$5D$	$30D$	$5D$	$30D$
$50D$	74.4 / 33.6 (2.0)	68.2 / 24.6 (2.7)	89.2 / 36.4 (2.0)	93.2 / 22.5 (1.2)
$100D$	80.9 / 62.2 (3.5)	68.2 / 26.9 (2.3)	91.0 / 64.0 (1.6)	93.5 / 27.0 (3.1)
$300D$	75.0 / 57.5 (1.6)	59.6 / 35.2 (3.0)	87.3 / 58.4 (3.1)	88.9 / 36.5 (1.7)
$500D$	67.6 / 57.2 (2.3)	55.2 / 33.5 (2.4)	89.5 / 55.8 (3.1)	83.6 / 34.8 (1.5)
$1000D$	74.1 / 48.7 (2.3)	54.3 / 36.8 (1.1)	84.9 / 48.5 (0.6)	82.7 / 38.9 (1.7)
$1500D$	72.2 / 43.6 (1.7)	49.1 / 34.0 (2.1)	88.0 / 42.7 (3.0)	81.2 / 34.1 (2.9)

Table 9: The DSC results on the statistical difference in the performance of **DE** algorithm pairs. Results are reported in the format: percentage of (true, predicted)-pairs with performance differences that are not statistically significant / percentage of (true, random)-pairs with performance differences that are not statistically significant. The numbers in brackets correspond to the standard deviation for the latter percentage since this was computed over 5 independent runs.

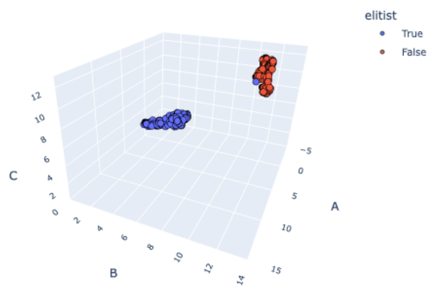
Budget	SHAP		Performance	
	$5D$	$30D$	$5D$	$30D$
$50D$	50.5 / 14.5 (1.2)	21.7 / 12.2 (1.5)	77.1 / 12.6 (1.7)	53.6 / 10.8 (0.7)
$100D$	45.0 / 11.2 (0.8)	23.6 / 10.9 (1.6)	70.5 / 12.2 (1.3)	54.3 / 11.7 (1.1)
$300D$	31.9 / 10.0 (1.4)	27.3 / 12.3 (0.7)	57.1 / 9.9 (1.1)	45.3 / 13.2 (0.9)
$500D$	31.1 / 11.5 (0.9)	28.6 / 16.6 (1.2)	56.9 / 11.3 (1.6)	48.1 / 15.7 (1.1)
$1000D$	31.2 / 13.0 (1.1)	31.8 / 19.5 (1.5)	49.5 / 12.6 (1.0)	45.5 / 19.0 (1.0)
$1500D$	33.0 / 16.0 (1.5)	30.4 / 23.2 (0.9)	54.5 / 15.0 (1.7)	46.5 / 22.5 (1.8)

An additional noteworthy observation is that the percentage of pairs with performance differences that are not statistically significant is higher for the CMA-ES algorithm variants as compared to DE. To further investigate this observation, we use UMAP (McInnes et al., 2018) as a dimensionality reduction technique to depict the performance-based meta-representations of CMA-ES and DE algorithm variants. Specifically, we focus on the $5D$ problems and $300D$ budget cut-off.

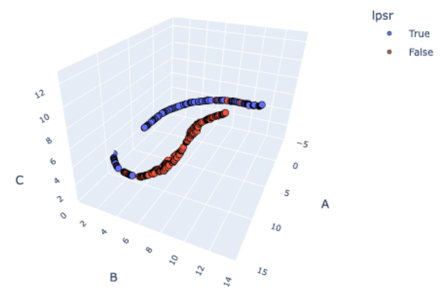
Figure 6 showcases the UMAP plots, allowing for a visual examination of the performance space. Notably, the CMA-ES algorithm variants exhibit closer proximity to one another, forming two distinct clusters. This close grouping suggests similar performance characteristics among these variants. We have observed that the elitism module almost perfectly separates the algorithm variants into two clusters. In contrast, the DE algorithm variants display more pronounced differences in performance, leading to a lower percentage of pairs with performance differences that are not statistically significant. In this case, the purest clusters are formed by taking into consideration the configurations of the `lpsr` module, indicating that this module exerts the greatest influence on performance as compared to the other modules.

6 Conclusion

In this study, we propose a methodology for examining the impact of different modules of optimization algorithms on the overall algorithm performance. We



(a) CMA-ES



(b) DE

Figure 6: UMAP embeddings of the performance-based meta-representations of the 324 CMA-ES and 576 DE algorithm variants.

demonstrate its relevance within two pre-existing modular optimization frameworks, namely modCMA-ES and modDE. To this end, we analyze performance data from 324 modCMA-ES and 576 modDE algorithm variants across 24 noiseless BBOB problems. Among the investigated CMA-ES modules, we have found that the elitism module has the most pronounced influence on performance, while the local restart module has the smallest influence, particularly for smaller runtime budgets. These findings are aligned with existing work analyzing these algorithms. Regarding DE, out of the seven modules examined, we observe that the linear population size reduction module exerts the greatest influence on performance. The mutation reference and adaptation method modules have considerably smaller effects as compared to the other modules.

Although our findings on some modules are not conclusive, our methodology is adaptable and can be applied to other modular optimization frameworks, where it may yield different insights. In our future work, we plan to apply our methodology to other modular optimization frameworks, such as ParadisEO (Cahon et al., 2004; Dreo et al., 2021) and the modular hybridization framework of particle swarm optimization and differential evolution (Boks et al., 2020).

Through the observation of variations in the impact of different modules on performance, we reach the conclusion that to accurately assess the contribution of a new idea or algorithm design, it is crucial to compare algorithm modules rather than algorithms themselves.

Furthermore, we have successfully trained classifiers to predict the modular configuration of algorithm variants. We find that the classifiers achieve higher F1 scores, in both cases of using performance-based and Shapley-based algorithm meta-representations when the impact of the module on performance is more substantial. This observation is expected because, in cases where the impact is less significant, algorithm variants tend to be closer in the meta-representation space, making it challenging for the ML model to differentiate effectively.

The classifiers built from performance-based meta-representations demonstrate superior predictive performance as compared to those built from Shapley-based meta-representations. However, it is worth noting that performance-based meta-representations are less flexible when it comes to accommodating new problem classes, as the vector size is predetermined by the number of classes. On the other hand, Shapley-based meta-representations, which rely on problem landscape features, maintain a consistent vector size when introducing new problem classes. Nonetheless, a limitation arises when new classes are introduced, requiring the retraining of regression models used to determine the Shapley-based landscape feature importance.

With respect to the importance of the landscape features, it seems that the same ELA features appear to be the most important features that contribute to the performance of the algorithm performance prediction regression models, regardless of the possible values of each module.

With the performance prediction in place, we can hope to deploy the modular algorithm frameworks in our *per-run algorithm selection* (Kostovska et al., 2022a) context, where we strive to select online which algorithm to choose for a given phase of optimizing a given problem instance. Our results indicate that switching from an elitist to a non-elitist selection rule could be beneficial for CMA-ES on a broad range of problems.

Finally, as future work, we also plan to explore the interplay between the different modules and their influence on the algorithm performance.

Acknowledgments

The authors acknowledge the support of the Slovenian Research Agency through program grant No. P2-0103 and P2-0098, a young researcher grant No. PR-09773, project grant J2-4460, and a bilateral project between Slovenia and France, grant No. BI-FR/23-24-PROTEUS-001 (PR-12040), as well as the EC through grant No. 952215 (TAILOR). Our work is also supported by ANR-22-ERCS-0003-01 project VARIATION.

References

Alarie, S., Audet, C., Gheribi, A. E., Kokkolaras, M., and Le Digabel, S. (2021). Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, 9:100011.

Andersen, B., Delipei, G., Kropaczek, D., and Hou, J. (2022). MOF: A modular framework for rapid application of optimization methodologies to general engineering design problems. *arXiv preprint arXiv:2204.00141*.

Aranha, C., Camacho Villalón, C. L., Campelo, F., Dorigo, M., Ruiz, R., Sevaux, M., Sörensen, K., and Stützle, T. (2022). Metaphor-based metaheuristics, a call for action: the elephant in the room. *Swarm Intelligence*, 16(1):1–6.

Aziz-Alaoui, A., Doerr, C., and Dreo, J. (2021). Towards large scale automated algorithm design by integrating modular benchmarking frameworks. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 1365–1374. ACM.

Bates, S., Hastie, T., and Tibshirani, R. (2023). Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association*, pages 1–12.

Belkhir, N., Dréo, J., Savéant, P., and Schoenauer, M. (2017). Per instance algorithm configuration of CMA-ES with limited budget. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 681–688. ACM.

Bezerra, L. C., López-Ibánez, M., and Stützle, T. (2015). Automatic component-wise design of multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417.

Boks, R., Wang, H., and Bäck, T. (2020). A modular hybridization of particle swarm optimization and differential evolution. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 1418–1425. ACM.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Cahon, S., Melab, N., and Talbi, E.-G. (2004). ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380.

Chen, H., Lundberg, S. M., and Lee, S.-I. (2022). Explaining a series of models by propagating Shapley values. *Nature communications*, 13(1):4512.

Collautti, M., Malitsky, Y., Mehta, D., and O'Sullivan, B. (2013). SNNAP: Solver-Based Nearest Neighbor for Algorithm Portfolios. In *Machine Learning and Knowledge Discovery in Databases*, pages 435–450. Springer.

Das, S. and Suganthan, P. N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31.

de Nobel, J., Vermetten, D., Wang, H., Doerr, C., and Bäck, T. (2021a). 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, Companion Material)*, pages 1375–1384. ACM.

de Nobel, J., Wang, H., and Baeck, T. (2021b). Explorative data analysis of time series based algorithm features of CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 510–518.

de Nobel, J., Ye, F., Vermetten, D., Wang, H., Doerr, C., and Bäck, T. (2024). IOHexperimenter: Benchmarking platform for iterative optimization heuristics. *Evolutionary Computation*, pages 1–6. Available online at https://doi.org/10.1162/evco_a_00342.

Doerr, C., Wang, H., Ye, F., van Rijn, S., and Bäck, T. (2018). IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *CoRR*, abs/1810.05281. An up-to-date documentation of IOHprofiler is available at https://iohprofiler.github.io/.

Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., and Bäck, T. (2020). Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dreo, J., Liefooghe, A., Verel, S., Schoenauer, M., Merelo, J. J., Quemy, A., Bouvier, B., and Gmys, J. (2021). ParadisEO: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of paradiseo. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 1522–1530. ACM.

Eftimov, T., Korošec, P., and Seljak, B. K. (2017). A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics. *Information Sciences*, 417:186–215.

Eftimov, T., Popovski, G., Kocev, D., and Korošec, P. (2020). Performance2vec: a step further in explainable stochastic optimization algorithm performance. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 193–194. ACM.

Eiben, Á. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Natural Computing. Springer, 2 edition.

Hall, N. G. and Posner, M. E. (2010). The generation of experimental data for computational testing in optimization. In *Experimental methods for the analysis of optimization algorithms*, pages 73–101. Springer.

Hansen, N., Auger, A., Ros, R., Finck, S., and Posík, P. (2010). Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 1689–1696. ACM.

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., and Brockhoff, D. (2020). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144.

Hansen, N., Finck, S., Ros, R., and Auger, A. (2009). Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Technical Report RR-6829, INRIA.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 312–317. IEEE.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. (2023). TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*.

Hooker, J. N. (1994). Needed: An empirical science of algorithms. *Operations research*, 42(2):201–212.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1:33–42.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Hussain, K., Mohd Salleh, M. N., Cheng, S., and Shi, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial intelligence review*, 52:2191–2233.

Jankovic, A. and Doerr, C. (2020). Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 841–849. ACM.

Jankovic, A., Eftimov, T., and Doerr, C. (2021a). Towards Feature-Based Performance Regression Using Trajectory Data. In *Proc. of Applications of Evolutionary Computation (EvoApplications 2021)*, volume 12694 of *LNCS*, pages 601–617. Springer.

Jankovic, A., Popovski, G., Eftimov, T., and Doerr, C. (2021b). The Impact of Hyper-Parameter Tuning for Landscape-Aware Performance Regression and Algorithm Selection. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*. ACM.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proc. of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Kerschke, P. and Trautmann, H. (2016). The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 5262–5269. IEEE.

Kerschke, P. and Trautmann, H. (2019). Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation*, 27(1):99–127.

Kostovska, A., Jankovic, A., Vermetten, D., de Nobel, J., Wang, H., Eftimov, T., and Doerr, C. (2022a). Per-run algorithm selection with warm-starting using trajectory-based features. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, pages 46–60. Springer.

Kostovska, A., Vermetten, D., Džeroski, S., Doerr, C., Korosec, P., and Eftimov, T. (2022b). The importance of landscape features for performance prediction of modular CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 648–656. ACM.

Kostovska, A., Vermetten, D., Džeroski, S., Panov, P., Eftimov, T., and Doerr, C. (2023a). Using knowledge graphs for performance prediction of modular optimization algorithms. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 253–268. Springer.

Kostovska, A., Vermetten, D., Korošec, P., Džeroski, S., Doerr, C., and Eftimov, T. (2023b). Linking Problem Features with Configurations of Modular Black-box Optimization Algorithms. Data, code, additional figures for this work are available at `https://doi.org/10.5281/zenodo.8151814`.

Kramer, O. (2017). *Genetic Algorithm Essentials*, volume 679 of *Studies in Computational Intelligence*. Springer.

Kumar, I. E., Venkatasubramanian, S., Scheidegger, C., and Friedler, S. (2020). Problems with Shapley-value-based explanations as feature importance measures. In *International Conference on Machine Learning*, pages 5491–5500. PMLR.

Lones, M. A. (2020). Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms. *SN Computer Science*, 1(1):49.

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

Lundberg, S. M. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In *Proc. of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

McInnes, L., Healy, J., Saul, N., and Großberger, L. (2018). UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29):861.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, pages 829–836.

Molina, D., LaTorre, A., and Herrera, F. (2018). An insight into bio-inspired and evolutionary algorithms for global optimization: review, analysis, and lessons learnt over a decade of competitions. *Cognitive Computation*, 10:517–544.

Molnar, C. (2020). *Interpretable Machine Learning*. Lulu Press.

Muñoz, M. A., Kirley, M., and Halgamuge, S. K. (2012). A meta-learning prediction model of algorithm performance for continuous optimization problems. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, pages 226–235. Springer.

Nikolikj, A., Lang, R., Korošec, P., and Eftimov, T. (2022a). Explaining differential evolution performance through problem landscape characteristics. In *Proc. of Bioinspired Optimization Methods and Their Applications (BIOMA))*, pages 99–113. Springer.

Nikolikj, A., Trajanov, R., Cenikj, G., Korošec, P., and Eftimov, T. (2022b). Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830.

Piotrowski, A. P. (2017). Review of differential evolution population size. *Swarm Evol. Comput.*, 32:1–24.

Prager, R. P., Trautmann, H., Wang, H., Bäck, T. H., and Kerschke, P. (2020). Per-Instance Configuration of the Modularized CMA-ES by Means of Classifier Chains and Exploratory Landscape Analysis. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 996–1003. IEEE.

Renau, Q., Doerr, C., Dréo, J., and Doerr, B. (2020). Exploratory landscape analysis is strongly sensitive to the sampling strategy. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, volume 12270 of *LNCS*, pages 139–153. Springer. Data available at: https://doi.org/10.5281/zenodo.3886816.

Renau, Q., Dréo, J., Doerr, C., and Doerr, B. (2021). Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions. In *Proc. of Applications of Evolutionary Computation (EvoApplications 2021)*, volume 12694 of *LNCS*, pages 601–617. Springer.

Rozemberczki, B., Watson, L., Bayer, P., Yang, H.-T., Kiss, O., Nilsson, S., and Sarkar, R. (2022). The Shapley value in machine learning. In *The 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence*.

Stegherr, H., Heider, M., and Hähner, J. (2022). Classifying metaheuristics: Towards a unified multi-level classification system. *Natural Computing*, 21(2):155–171.

Stork, J., Eiben, A. E., and Bartz-Beielstein, T. (2022). A new taxonomy of global optimization algorithms. *Natural Computing: An International Journal*, 21(2):219–242.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Trajanov, R., Dimeski, S., Popovski, M., Korošec, P., and Eftimov, T. (2021). Explainable landscape-aware optimization performance prediction. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08. IEEE.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7(1):1–8.

Vermetten, D., Caraffini, F., Kononova, A. V., and Bäck, T. (2023a). Modular differential evolution. In *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 864–872. ACM.

Vermetten, D., López-Ibáñez, M., Mersmann, O., Allmendinger, R., and Kononova, A. V. (2023b). Analysis of modular CMA-ES on strict box-constrained problems in the SBOX-COST benchmarking suite. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 2346–2353. ACM.

Weise, T. and Wu, Z. (2018). Difficult features of combinatorial optimization problems and the tunable W-model benchmark problem for simulating them. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, pages 1769–1776. ACM.