Impact of Scaling in ELA Feature Calculation on Algorithm Selection Cross-Benchmark Transferability

Gjorgjina Cenikj,[1,2,*] Gašper Petelin[1,2] and Tome Eftimov[1]

[1]Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, 1000, Ljubljana, Slovenia and [2]Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000, Ljubljana, Slovenia

[*]Jamova cesta 39. Gjorgjina Cenikj, Jožef Stefan Institute, 1000 Ljubljana, Slovenia. gjorgjina.cenikj@ijs.si

Abstract

Exploratory Landscape Analysis (ELA) features are the most common choice for representing single-objective continuous optimization problem instances in Algorithm Selection (AS) methods. However, ELA features have been shown to have low generalization to unseen problems. Recently, it has also been shown that scaling objective function values before ELA feature calculation can be beneficial for AS methods evaluated on the Black-Box Optimization Benchmarking suite. In this paper, we aim to evaluate whether the same holds true for other benchmarks. In particular, we take into account four different benchmark suites and investigate the ability of an AS model trained on one benchmark to generalize to another benchmark. We also evaluate the impact of scaling objective function values before ELA feature calculation on AS performance. We observe a benefit of scaling objective function values in the case of the use of all ELA features together, however, conflicting outcomes are obtained when different feature groups are used individually. Our analysis shows no benefit of the joint use of the ELA features calculated on the original and scaled objective function values.

Key words: algorithm selection, exploratory landscape analysis, transfer learning

1. Introduction

Algorithm Selection (AS), the task of selecting an optimal optimization algorithm for a given problem, is driven by the desire to harness the diverse performance exhibited by different algorithms across different problem instances (Kerschke et al., 2019). AS can be addressed as a Machine Learning (ML) problem, where given a vectorized representation of the problem instance that represents its landscape characteristics, an ML model is trained to find the most suitable algorithm for that problem instance.

In the field of single-objective numerical optimization, problem instances are most commonly represented using the Exploratory Landscape Analysis (ELA) (Mersmann et al., 2011) features. These are mathematical and statistical features computed on candidate solutions sampled from the decision space of the problem instance. While ELA features have been extensively used across different AS studies (Jankovic et al., 2021; Prager and Trautmann, 2023; Škvorc et al., 2022; Kerschke et al., 2016), they also exhibit limitations such as sensitivity to sample size and the sampling method used for solution sampling from the decision space(Renau et al., 2020; Škvorc et al., 2021), and lack of invariance to transformations such as scaling and shifting of the problem (Škvorc et al., 2021, 2020). They have also demonstrated limited effectiveness in generalizing to new problems when used for AS (Petelin and Cenikj, 2023; Cenikj et al., 2024).

In (Prager and Trautmann, 2023), it was pointed out that the issue of ELA features being sensitive to problem transformations could be partially mitigated by performing a min-max normalization of the objective function values before feature calculation. This was also hypothesized to improve the ability of AS models to generalize to unseen functions, where the range of the objective function values differs substantially from the range of the objective function values of the problem instances in the training data. The utility of the proposed scaling has been shown on the The Black-Box Optimization Benchmarking (BBOB) (Hansen et al., 2009) suite, in three evaluation strategies (Prager and Trautmann, 2023), tackling the tasks of problem classification as well as AS. However, the BBOB benchmark is somewhat limited in problem diversity. Namely, it contains 24 problem classes, from which problem instances can be generated by applying some transformation (such as shifting or scaling) of the original problem. In this way, an arbitrary number of problem instances can be generated from a single problem class, however, the resulting problem instances are highly similar to each other.

To provide a more comprehensive analysis of the impact of scaling of the objective function values before ELA feature calculation, in this paper, we evaluate the generalization ability of AS models, trained and tested using four different benchmark suites. We take into account the BBOB benchmark (Hansen et al., 2009) and the recently proposed affine combinations of BBOB problems (Dietrich and Mersmann, 2022). We also use two problem generators (Kudela and Matousek, 2022; Tian et al., 2020) to generate two additional benchmark suites. These four benchmark suites have already been involved in a study analyzing the generalizability of AS models across benchmark suites (Cenikj et al., 2024) using the ELA features and the recently proposed TransOpt (Cenikj et al., 2023) features. However, in this case, all of the ELA features were explored together, and the impact of scaling objective function values was not taken into account. Other studies on the topic of generalizability of AS models also use some of these benchmark suites independently. For instance, (Škvorc et al., 2022) uses the BBOB and a random function generator, while (Petelin and Cenikj, 2023) uses the BBOB and its affine combinations. To the best of our knowledge, this is the first study that evaluates the generalization of individual ELA feature groups across four different benchmarks, also taking into account the impact of scaling objective function values before ELA feature calculation.

Consistent with previous work (Prager and Trautmann, 2023), our results indicate that only a few feature groups are affected by the scaling. Specifically, these are the *ela_meta*, *basic*, *pca*, and *ic* feature groups, where scaling also influences AS performance. Our results show that when the ELA features are computed using the original objective function values, problems with extreme values belonging to one benchmark might be located far away from problems of other benchmarks in the problem landscape, leading to lower transferability of the AS model. Scaling the objective function values results in the benchmarks being embedded closer together, which enhances the transferability of the AS model when all features are used together, however, conflicting outcomes are obtained when features from different feature groups are used individually.

The rest of the paper is organized as follows. Section 2 details the methodology and the experimental setup. Section 3 contains the results, while Section 4 concludes the paper.

Reproducibility: The data and code used in this study are available online at https://github.com/gjorgjinac/AS_ela_scaling.

2. Methodology

In this section, we first introduce the problem portfolios, i.e., the set of benchmarks used for training and evaluating the AS model. We then present the algorithms included in the algorithm portfolio, as well as the performance metric used to compare them. Next, we describe the ELA features utilized to characterize the problem instances. Finally, we present the training of the ML model used for AS, as well as the evaluation metric. Our methodology is closely aligned to that of (Cenikj et al., 2024), with the difference that we take into account individual ELA feature groups and analyze the impact of scaling objective function values before feature calculation. The problem portfolios and algorithm performance data are reused across both studies, as are the ELA features calculated without objective function values.

2.1. Problem Portfolios

Next, the benchmark suites utilized in this study are explained in more detail. We need to note that for all benchmarks, we use $3d$ problems, where d is the problem dimension. We have also performed the same experiments for $10d$ problems, however, due to space limitations, we do not present them within this manuscript. They can be found in our github repository.

BBOB benchmark - The BBOB (Hansen et al., 2009, 2020) suite is a set of 24 single-objective optimization problems. Different problem instances can be created from the original problem by applying a transformation (such as scaling or shifting). We use the first 100 problem instances from each of the 24 problems from the BBOB suite, resulting in a total of 2,400 problem instances from this benchmark.

AFFINE benchmark - We use the first five problem instances from the 24 BBOB problem classes to generate affine combinations as proposed in (Vermetten et al., 2023). The combinations are generated by combining problem instances from different problem classes that have the same instance ID. For example, the first instance of the first problem class is combined with the first instances of the other 23 problem classes. We need to emphasize here that instances with different IDs from different problem classes are not combined together. We do this to limit the number of generated instances. The combination is done with alpha values of 0.25, 0.50, and 0.75 for all pairs of problem instances. In this way, we get 8,280 problem instances.

RANDOM benchmark - We create 5,000 random problem instances using the random generator proposed in (Tian et al., 2020). The generator builds a tree representation by randomly combining mathematical operands and operators from a predefined pool, where each operand and operator has a certain probability of being chosen. We use the Python version of the random function generator, which was initially implemented in Matlab and was re-implemented in Python in (Long et al., 2022). Please note that some of the objective functions in this dataset are duplicated or very similar. This is due to the way function generation is done and is a limitation of the generator. From the initial pool of generated functions, we remove those that produce invalid or constant values, resulting in 4,446 functions used in the experiments.

ZIGZAG benchmark - We create 5,000 random problem instances by randomly initializing the parameters of the four zigzag functions proposed in (Kudela and Matousek, 2022). These four functions combine a zigzag function with different multimodal functions. Different instances from these functions can be generated by specifying three parameters, which we initialize as follows. The k parameter, controlling the period of the zigzag function, is randomly sampled as an integer in the range [0,30]. The m parameter, controlling its amplitude, is sampled in the range [0,1]. The λ parameter, controlling the location of the local minima, is also sampled in the range [0,1]. It has been shown that depending on the m, k, and λ parameters, the generator can produce diverse enough objective functions where some algorithms have difficulty solving them while others do not.

2.2. Algorithm Portfolio

The algorithm portfolio consists of four different algorithms: Differential Evolution (DE) (Storn and Price, 1997), Genetic Algorithm (GA) (Chahar et al., 2021), Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), and Evolutionary Strategy (ES) (Beyer and Schwefel, 2002). These algorithms are taken with their default configurations in the *pymoo* library, version 0.6.0. We have selected these algorithms due to their availability in the *pymoo* library, providing easy use with a standardized implementation, however, other algorithm portfolios can be tested as part of future work. All algorithms use Latin Hypercube Sampling (Menčík, 2016) to construct the initial population.

The algorithms are executed on all benchmarks in a fixed-budget scenario with a population size of $10d$, where d is the problem dimension. The performance of the algorithm is recorded at budgets of 10, 30, and 50 iterations. In the scope of this manuscript, we are only presenting the results of the AS for a budget of 50 iterations due to space limits. We have also performed experiments with budgets of 10 and 30 iterations, which can be found in our github repository. For a $3d$ problem, a budget of 50 iterations is equivalent to a total of 1,500 function evaluations (50 iterations with a population size of 30). We perform ten executions of each algorithm on all of the problem instances, and calculate the algorithm performance as described in the next section.

2.3. Algorithm Performance Metric

In our experimental setup, we adopt a fixed-budget scenario to assess the algorithms, wherein the evaluation involves recording the best-found objective function value after a specified budget of algorithm iterations. Given that the RANDOM and ZIGZAG algorithms lack a defined optimum for the considered problem instances, we deviate from the conventional approach of predicting achieved precision (distance of the best-found solution to the optimum) (Jankovic et al., 2021). Instead, we opt to use a relative score that was introduced in (Cenikj et al., 2024) and is defined as follows.

Let y_{arp} represent the best solution (lowest objective function value) found by algorithm a in run r for problem instance p. This value, y_{arp}, is normalized by the range of solutions found by other algorithms for the same problem instance and initial population (same run). Denoting b_{rp} and w_{rp} as the best and worst solutions found for problem instance p in run r by any algorithm in the set, respectively, we derive the scaled best objective function value achieved by algorithm a in run r on problem instance p:

$$s_{arp} = \frac{y_{arp} - b_{rp}}{w_{rp} - b_{rp}}. \tag{1}$$

This metric quantifies the extent to which the best solution found by algorithm a in run r outperforms other algorithms executed with the same initial population. The overall performance of algorithm a on problem instance p across all runs is summarized by calculating the final value s_{ap}, obtained by taking the median of all s_{arp}. It is important to note that this is a minimization function, meaning that the best algorithm for a given problem has the lowest score.

Table 1. Example of the calculation of the algorithm performance metric for one problem instance

run	DE	ES	GA	PSO
Objective value of best solution found (Per run)				
1	-15.10	-14.18	-14.95	-15.03
2	-15.09	-14.13	-14.37	-14.16
3	-15.12	-13.94	-14.89	-15.07
Normalized Algorithm Score (Per run)				
1	0.00	1.00	0.16	0.08
2	0.00	1.00	0.75	0.97
3	0.00	1.00	0.19	0.04
Final Algorithm Score (Median across all runs)				
	0.00	1.00	0.19	0.08

Table 1 provides an illustrative example of how algorithm performance scores are computed for a single problem instance and a specific budget. It is important to note that, in the example, each algorithm is executed three times, whereas, in reality, we execute the algorithms ten times on each problem instance. The table is segmented into three parts, each depicting a distinct stage in the algorithm performance metric calculation process. The initial segment presents the objective function values of the best solutions found by each algorithm after a fixed budget of function iterations. In the subsequent step, these values undergo normalization within each run through min-max normalization column-wise. This implies that, for each run, the best-performing algorithm attains a score of 0, while the worst algorithm obtains a score of 1. In this particular example, DE is identified as the best algorithm, and ES as the worst. The final algorithm score is computed by determining the median of the normalized algorithm scores across all runs. It is worth noting that, in certain cases where algorithms exhibit high variability between runs, it is possible that none of the algorithms will have a final score of 0 or 1.

We opt to use such a performance metric for several reasons: 1) As opposed to using multi-class classification (Škvorc et al., 2022) and predicting a single best algorithm, this approach captures the performance of all algorithms and allows us to perform multi-target regression predicting the performance of all algorithms at the same time, circumventing the issue that a model is penalized for predicting one out of two algorithms with very similar performance. 2) In contrast with the ranking approach, where each algorithm is assigned an integer score based on its performance, our approach captures a fine-grained difference in performance, allowing algorithms with similar best-obtained values to have similar scores.

2.4. Problem Representations

We compute ELA features from the following categories using the *flacco* R library (Kerschke and Trautmann, 2019): *basic*, *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we calculate a total of 93 features for each problem instance. We calculate these features twice, once using the original objective function values, and once using the objective function values scaled using min-max scaling in the range [0,1] within the samples of each problem individually.

We discard eight features which measure the additional function evaluations needed to calculate the feature (*costs_fun_evals* from each feature group) because they have a constant value for all problems. We also discard the eight features which measure the execution time needed to calculate the features (*costs_runtime* from each feature group), since this feature might differ for the unscaled and scaled sample, but the difference is not due to the scaling itself.

It is important to note that some of these features contain missing or constant values, which we remove before training the AS. When the features are calculated with the original objective function values, 1 *nbc*, 4 *pca* and 11 *basic* features are removed, apart from the *costs_runtime* and *costs_fun_evals*, which are removed for all feature groups. Calculating the ELA features with the scaled objective function values, results in the additional removal of 1 feature from the *pca* and *ic* feature groups, and 2 additional features from the *basic* group.

A full list of the removed features is available in our github repository. We calculate the ELA features on a sample of the problem instances obtained with Latin Hypercube Sampling in the range of $[-5,5]^d$, with a sample size of $50d$, where d is the problem dimension. This sample size has been previously shown to work well for the AS task (Kerschke et al., 2016).

2.5. Model

We use a Random Forest (RF) model to perform multi-target regression, where the model predicts the performance score of each algorithm. We chose the RF model because it has good performance on tabular data (Shwartz-Ziv and Armon, 2022) and it provides feature importance values that help with interpretability. The RF model is executed using the default configuration parameters in the *scikit-learn* (Pedregosa et al., 2011a) python library version 1.2.2. We do not perform parameter tuning in order to evaluate the impact of the proposed features only, on a fixed model configuration. A separate RF model is trained for each algorithm execution budget, each one predicting the scores obtained by the algorithms after this budget. To evaluate the impact of scaling objective function values before ELA feature calculation, we perform three experiments:

1) Using the ELA features calculated with the original objective function values,

2) Using the ELA features calculated with the objective function values scaled within the samples of each problem instance,

3) Using a concatenation of the ELA features calculated with the original objective function values and those calculated with the scaled objective function values.

To ensure the robustness of the results, the training of the RF models is repeated 10 times. In each execution, an entire benchmark is used to train the model, and the evaluation is carried out on the remaining three benchmarks. The training is repeated 10 times because different random seeds influence the RF learning process.

2.6. Algorithm Selector Loss

To capture the true performance of the algorithm predicted to be the best by the algorithm selector, we calculate the loss of the AS:

$$loss = \frac{1}{|\mathcal{P}|} \sum_{p \in P} 1 - (s_{sp} - s_{bp}) \tag{2}$$

where s_{sp} is the true score of the selected algorithm which was predicted to have the best performance for problem instance p, while s_{bp} is the score of the best-performing algorithm on problem instance p (i.e. the virtual best solver). This score would get a value of zero if the worst algorithm is predicted to be the best for all problem instances, and a value of one if the best algorithm is predicted to be the best for all problem instances.

Table 2 demonstrates how the loss is calculated for two problem instances and three algorithms. For the first instance, the true best algorithm is DE with a score of 0, and the model predicts it to be the best. The loss score is 1, because the metric is simply “flipped” so we are maximizing it. For the second instance, the GA algorithm is predicted to be the best, however, the true score of the GA algorithm is 0.2, and the true best-performing algorithm is DE with a score of 0.05. If one were to use the

Table 2. An example of how the loss metric is calculated for two problem instances

instance	True scores			Predicted scores			Loss
	DE	GA	ES	DE	GA	ES	
0	0.00	1.0	0.3	0.00	0.2	0.3	1.00
1	0.05	0.2	1.0	0.40	0.0	0.7	0.85

GA algorithm instead of the DE algorithm in practice, the choice of GA over DE, they would get a result that is 0.15 (0.2 - 0.05) worse than if they were to use the true best algorithm. The loss score in this case is 0.85 (1-0.15).

3. Results

In this section, we first identify the ELA features affected by the scaling of the objective function values. We then present the results of the AS obtained with different ELA feature groups. Finally, we analyze the reasons behind the observed impacts of the objective function value scaling on the results of the AS.

3.1. Features Affected By Scaling

To identify the features that are affected by the scaling of the objective function value, we calculate the mean absolute error (MAE) of the original feature values and the ones obtained with the scaled sample. Figure 1 shows the features for which the MAE exceeds 0.05. The vertical axis shows the name of the feature, as well as the original MAE value in brackets, while the y-axis shows the MAE in logarithmic scale. From the figure, we can see that the features impacted by the scaling come from the *ela_level*, *ela_meta*, *basic*, *pca*, *ela_distr*, and *ic* feature groups. However, it is important to note that the features from the *ela_level* group exhibit small MAE values, as is the case with the *pca.expl_var.cov_init*, *pca.expl_var_PC1.cov_init*, *ela_distr.number_of_peaks*, *ic.eps.s* and *ic.eps.ratio*. The features with the largest MAE values are *ela_meta.lin_simple.intercept*, *ela_meta.lin_simple.coef.min*,

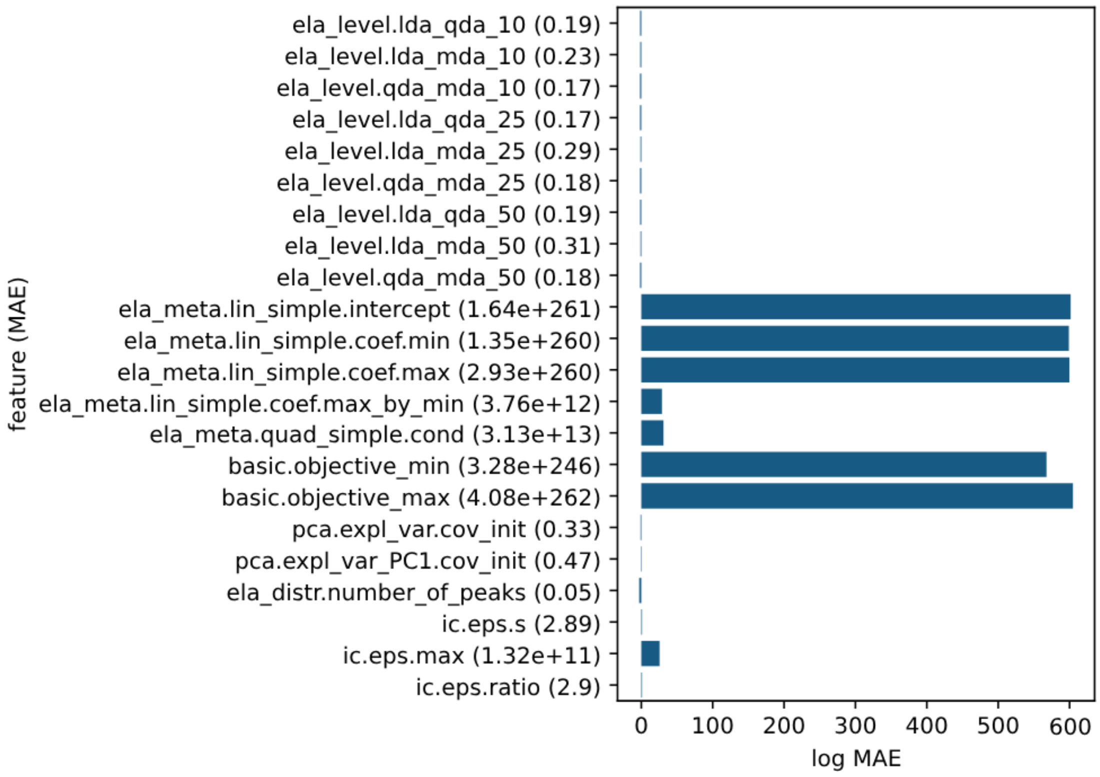


Fig. 1: ELA features affected by the scaling of the objective function values

ela_meta.lin_simple.coef.max, *ela_meta.lin_simple.coef.max_by_min*, *ela_meta.quad_simple.cond*, *basic.objective_min*, and *basic.objective_max*. Most of these *ela_meta* features capture the coefficients of a linear model fit to the samples of the objective function, while the two *basic* features simply extract the minimum and maximum of the objective function values obtained in each problem sample.

This analysis indicates that scaling affects the values of some of the features, while others remain unchanged by it, which is consistent with findings of previous studies (Prager and Trautmann, 2023).

3.2. Algorithm Performance Across Benchmarks

Next, we analyze the performance of the algorithms on the four benchmarks. Figure 2 shows the empirical cumulative distribution of the algorithm scores across the benchmarks. Each subplot refers to a different benchmark, while different colors denote different algorithms. This plot can be interpreted in the following way. Since the algorithm performance score (indicated on the x-axis) is a minimization metric, lower algorithm scores indicate better performance. The y-axis represents the proportion of instances which have a score which is equal to or lower than the corresponding value on the x-axis. The closer the algorithm is to the upper left corner, the better its overall performance on the benchmark. For example, on the AFFINE, BBOB, and ZIGZAG benchmarks, the ES algorithm performs the best. It achieves a score close to zero for approximately 75% of problem instances in the BBOB benchmark, 90% of problem instances in the AFFINE benchmark, 93% of problem instances in the ZIGZAG benchmark, but only 40% of problem instances on the RANDOM benchmark. This means that if one simply uses the single-best solver from the BBOB, AFFINE or ZIGZAG benchmark to solve problems from the RANDOM benchmark, poor results will be achieved compared to other algorithms, highlighting the need for AS models. The BBOB and AFFINE benchmarks produce a similar ranking of the algorithms in the order ES, PSO, DE, and GA. On the other hand, the RANDOM benchmark produces a completely different ordering. On this benchmark, the best algorithm is DE, while the ES algorithm provides the worst results.

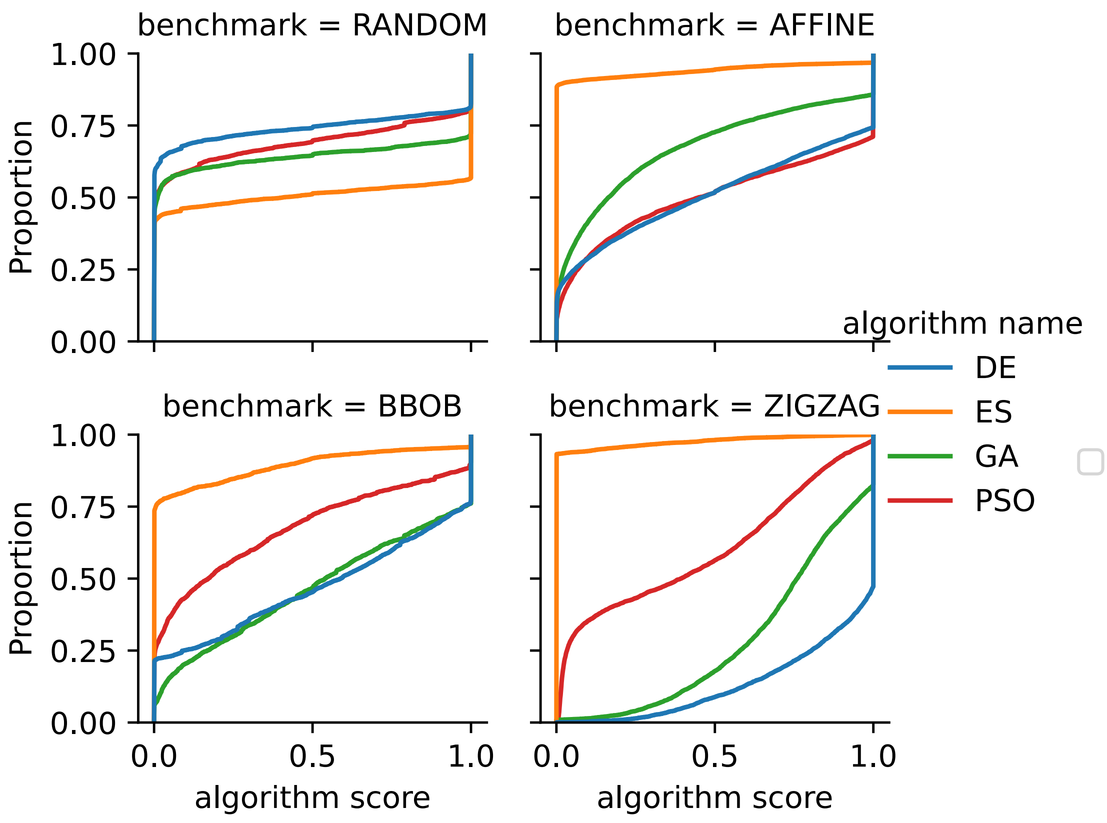


Fig. 2: Empirical cumulative distribution of algorithm scores across benchmarks

3.3. Algorithm Selection Results

Figure 3 shows the loss obtained by the AS model using different ELA feature groups. The subplots within each figure contain the AS loss obtained with a different feature group. We should note that we present results for all feature groups, regardless of whether the features were affected by scaling or not. We do this because we want to analyze the generalization power of the all of the feature groups, even if they are not impacted by scaling.

The horizontal axis contains the names of the training and testing benchmarks, in the form TRAIN-TEST. For instance, AFFINE-BBOB indicates that the model was trained on the AFFINE benchmark, and evaluated on the BBOB benchmark. On the other hand, the vertical axis contains the names of the features being used. In this case, “ELA” refers to the features obtained using the objective function values in their original range, “ELA Y Scaling” refers to the features calculated using the scaled objective function values, while “Merged” denotes the concatenated “ELA” and “ELA Y Scaling” features. “Dummy” denotes the baseline feature-less model which simply predicts the mean on the train set for each algorithm portfolio.

Looking at the topmost subplot, showing the results when all features are used, our first observation is that in many cases, the ELA-based AS model does not beat the dummy. This is the case for AFFINE-ZIGZAG, BBOB-AFFINE, BBOB-ZIGZAG, ZIGZAG-AFFINE, ZIGZAG-BBOB, and ZIGZAG-RANDOM. Such behavior is owed to the fact that the BBOB, AFFINE, and ZIGZAG benchmark have the same single best solver, as observed in the previous section. This means that simply predicting the best algorithm from the training benchmark performs very well on the testing benchmark. This is not the case when the training and testing benchmarks have a different single best solver. When all features are used, scaling the objective function values generally has a positive or neutral impact on the AS loss, meaning that the loss obtained using the ”ELA Y Scaling” features is greater or equal to the loss obtained using the original ”ELA” features. An improvement in the loss is observed in the case of AFFINE-BBOB, AFFINE-RANDOM, BBOB-AFFINE, BBOB-RANDOM and BBOB-ZIGZAG although the difference between the ”ELA” and ”ELA Y Scaling” is only substantial in the case of AFFINE-RANDOM.

Looking at the results obtained using the ELA features calculated without scaling, we can see that some feature groups obtain better results when used individually, compared to using all of the feature groups together. Such is the case for the *disp*, *ela_level*, *ela_meta*, *pca* on AFFINE-RANDOM, and *ela_level* on RANDOM-BBOB. Please note that here we are not considering the benchmark combination where the ELA-based models do not beat the dummy.

For the *basic* features, we can see that the scaling results in the AS having the same loss as the Dummy model. This is due to the fact that when the objective function values are scaled, the *basic* features have constant values for all problem instances, and therefore, the model essentially cannot learn anything. Examples of these features include the problem dimension and the number of samples from the problem (which are constant for all problems on which the AS is trained), as well as the minimum and maximum of the objective function values (which become constant for all problems when the objective function values are scaled).

The *ela_meta* features seem to benefit from scaling when the AS is trained on the RANDOM benchmark.

Scaling has a positive impact on the AS loss when the *ic* features are used in the case of AFFINE-RANDOM, BBOB-ZIGZAG, RANDOM-AFFINE, RANDOM-BBOB, and a negative impact in the case AFFINE-BBOB, AFFINE-ZIGZAG, BBOB-AFFINE, BBOB-RANDOM and RANDOM-ZIGZAG.

The *pca* features benefit from scaling in the cases AFFINE-BBOB, AFFINE-ZIGZAG, BBOB-AFFINE, BBOB-RANDOM, BBOB-ZIGZAG, RANDOM-AFFINE, RANDOM-BBOB, however, a negative effect is observed in the case of RANDOM-ZIGZAG.

In general, we do not observe a benefit of the joint use of the features calculated with and without scaling.

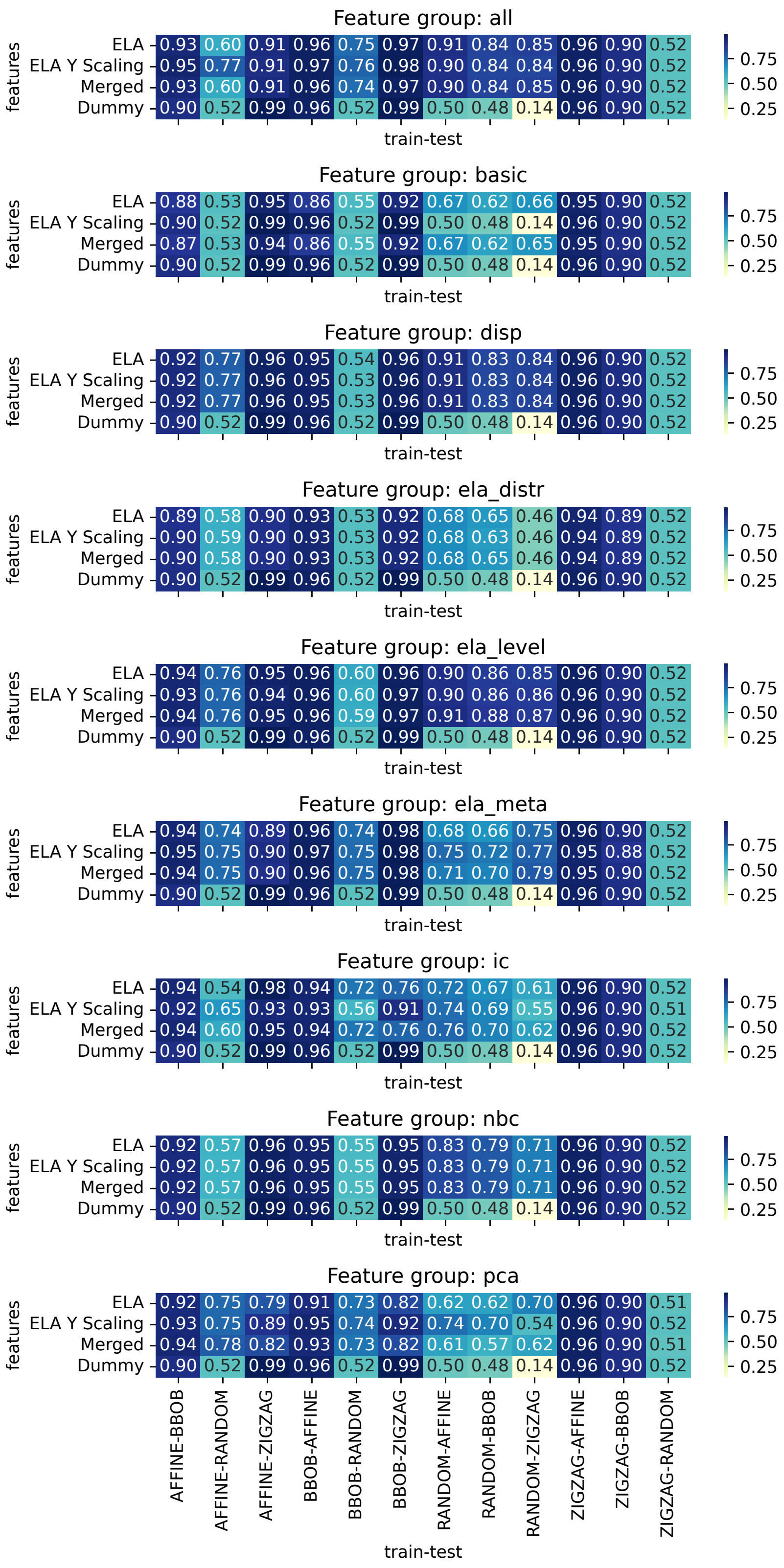


Fig. 3: AS loss obtained with each ELA feature group

3.4. Visualizations

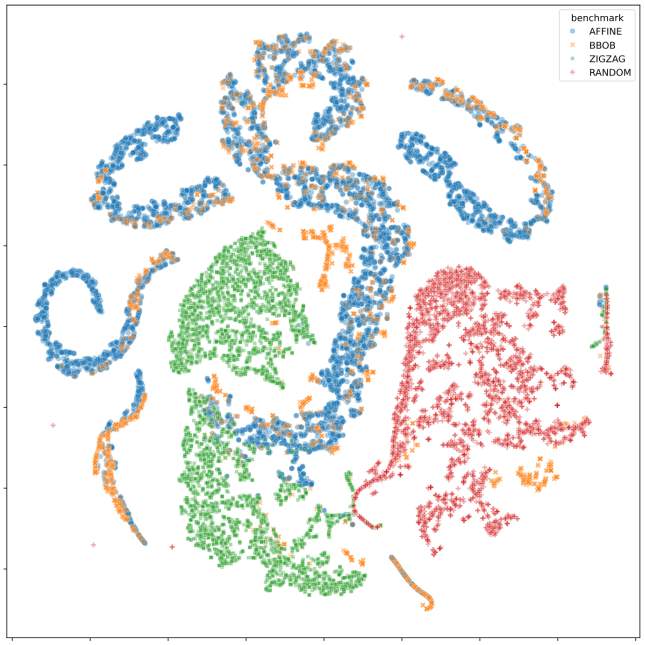
To understand why scaling impacts AS results as shown before, we analyze the 2-dimensional visualizations of the problem instances within benchmarks. To this end, the original ELA features and the ones obtained with scaled objective function values are reduced to two dimensions using t-distributed stochastic neighbor embedding (i.e., t-SNE) (van der Maaten and Hinton, 2008) with the default parameter values in the scikitlearn (Pedregosa et al., 2011b) library (euclidean distance, 1000 iterations and a perplexity of 30). Figure 4 shows the embedding of the benchmarks in 2-d space when all ELA features are used. The subfigure on the left shows the visualization generated with the ELA features calculated on the original objective function values, while the subfigure on the right shows the one generated with ELA features calculated on the scaled objective function values. In this figure, different colors indicate problems from different benchmarks. We can observe that when scaling is not applied, the different benchmarks can be clearly distinguished. The AFFINE and BBOB benchmarks are somewhat overlapping, while the ZIGZAG and RANDOM benchmarks occupy their own part of the problem landscape and can be visually divided from the others. To some degree, this can explain the lack of generalization between some benchmark suites. On the contrary, when the ELA features are calculated with scaling, the problem landscape seems to have no clear structure, with instances from different benchmarks being mixed together.

A similar pattern is present in Figure 5, demonstrating the visualization for the *ela_meta* features, for which we observed that scaling impacts the AS results in a positive way when the training is done on the RANDOM benchmark. When the *ela_meta* features are calculated on the original objective function values, the AFFINE and BBOB benchmarks are mostly occupying the same part of the problem landscape, while the RANDOM and ZIGZAG benchmark are somewhat divided from the other benchmarks. This is not the case when using the *ela_meta* calculated with scaling, where instances from different benchmarks are embedded close to each other. This explains the observed effect of scaling positively impacting the *ela_meta* features when the training is done on the RANDOM benchmark, since it results in the RANDOM problems being embedded close to the other benchmarks, allowing for better model generalizability.

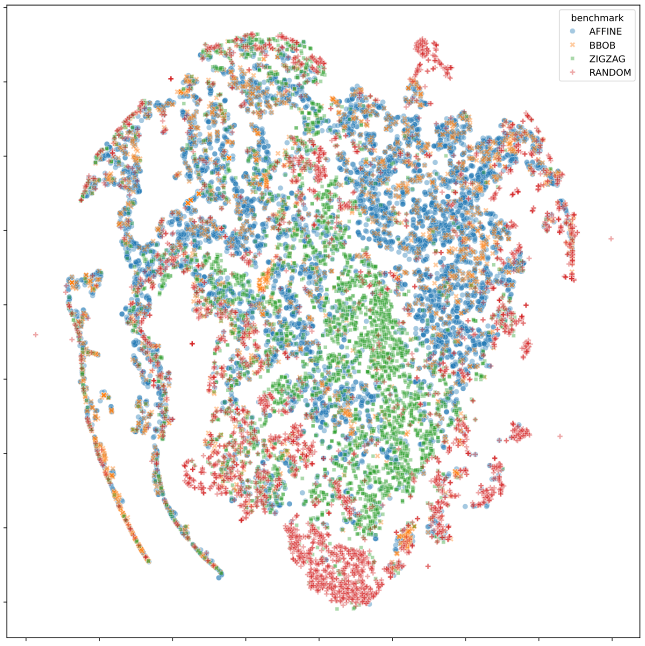
Due to page limitations, we omit the visualizations generated with the other feature groups, however, they are included in our github repository.

4. Conclusions

In this paper, we investigate the impact of scaling objective function values before ELA feature calculation, on the generalizability of an AS model across different benchmark suites. In line with previous work (Prager and Trautmann, 2023), we find that only a few feature groups are sensitive to the scaling. In particular, these are the *ela_meta*, *basic*, *pca*, and *ic* feature groups, where scaling also impacts AS generalization between several benchmark pairs. Our findings indicate that when the ELA features are calculated using the original objective function values, some benchmarks with extreme values are embedded far away from other benchmarks in the problem landscape, resulting in lower generalizability of the AS model. Scaling the objective function values results in the benchmarks being more closely embedded, which improves generalizability when all feature groups are used together. In most cases, a benefit of scaling is also observed when the *ela_meta* features are used individually. On the other hand, the scaling completely removes any predictive power from the *basic* feature group, which should be rather obvious, since scaling results

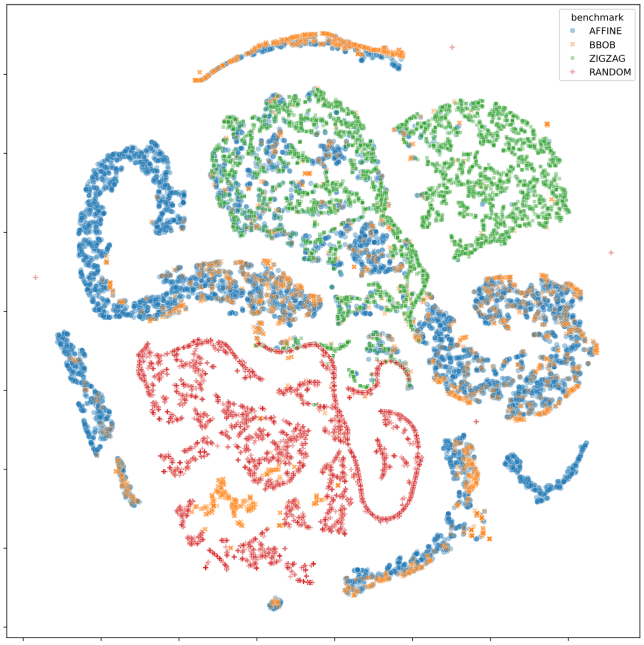


(a) Original

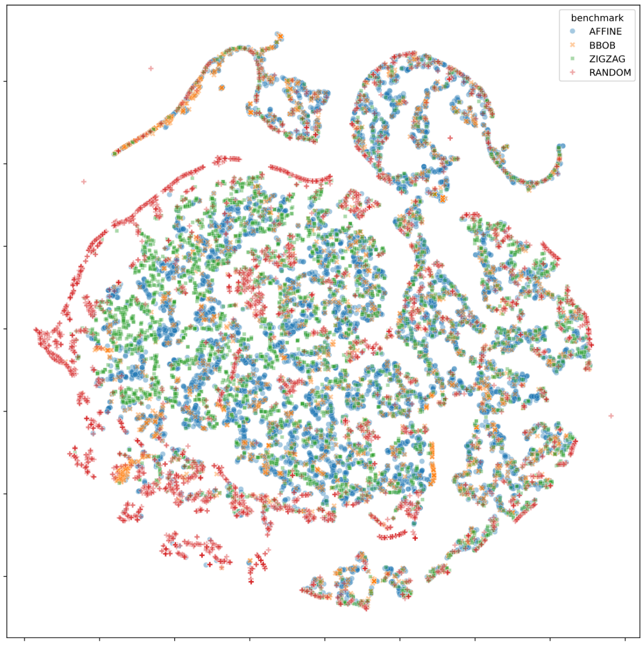


(b) Scaled

Fig. 4: Benchmarks represented using all ELA features, reduced in 2 dimensions using TSNE



(a) Original



(b) Scaled

Fig. 5: Benchmarks represented using the *ela_meta* features, reduced in 2 dimensions using TSNE

in these features having a constant value for all problems. For the *ic* and *pca* feature groups, we derive no clear conclusion on whether scaling has a positive or negative impact, since conflicting results were obtained for different pairs of training and testing benchmarks. Our results show no benefit of using a concatenation of the features calculated with and without scaling.

5. Acknowledgements

Funding in direct support of this work: Slovenian Research Agency: research core funding No. P2-0098, young researcher grants No. PR-12393 to GC and No. PR-11263 to GP, and project No. J2-4460.

References

H. Beyer and H. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002. doi: 10.1023/A:1015059928466.

G. Cenikj, G. Petelin, and T. Eftimov. Transopt: Transformer-based representation learning for optimization problem classification. *2023 IEEE Symposium Series on Computational Intelligence*, 2023. URL `http://arxiv.org/abs/2311.18035`.

G. Cenikj, G. Petelin, and T. Eftimov. A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization. *Swarm and Evolutionary Computation*, 87:101534, 2024. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2024.101534. URL `https://www.sciencedirect.com/science/article/pii/S2210650224000725`.

V. Chahar, S. Katoch, and S. Chauhan. A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80, 02 2021. doi: 10.1007/s11042-020-10139-6.

K. Dietrich and O. Mersmann. Increasing the diversity of benchmark function sets through affine recombination. In G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, editors, *Parallel Problem Solving from Nature – PPSN XVII*, pages 590–602, Cham, 2022. Springer International Publishing. ISBN 978-3-031-14714-2.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009. URL `https://hal.inria.fr/inria-00362633`.

N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, pages 1–31, 2020.

A. Jankovic, T. Eftimov, and C. Doerr. Towards feature-based performance regression using trajectory data. In *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, April 7–9, 2021, Proceedings 24*, pages 601–617. Springer, 2021.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995. doi: 10.1109/ICNN.1995.488968.

P. Kerschke and H. Trautmann. *Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-Package Flacco*, pages 93–123. Springer, 2019. ISBN 978-3-030-25147-5. doi: 10.1007/978-3-030-25147-5_7. URL `https://doi.org/10.1007/978-3-030-25147-5_7`.

P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann. Low-budget exploratory landscape analysis on multiple peaks models. pages 229–236, 07 2016. doi: 10.1145/2908812.2908845.

P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, 03 2019. ISSN 1063-6560. doi: 10.1162/evco_a_00242. URL `https://doi.org/10.1162/evco_a_00242`.

J. Kudela and R. Matousek. New benchmark functions for single-objective optimization based on a zigzag pattern. *IEEE Access*, 10:8262–8278, 2022. doi: 10.1109/ACCESS.2022.3144067.

F. X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, and T. Bäck. Learning the characteristics of engineering optimization problems with applications in automotive crash. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 1227–1236, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392372. doi: 10.1145/3512290.3528712. URL `https://doi.org/10.1145/3512290.3528712`.

J. Menčík. Latin hypercube sampling. In J. Mencik, editor, *Concise Reliability for Engineers*, chapter 16. IntechOpen, Rijeka, 2016. doi: 10.5772/62370. URL `https://doi.org/10.5772/62370`.

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

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011a.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011b.

G. Petelin and G. Cenikj. How far out of distribution can we go with ela features and still be able to rank algorithms? In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 341–346, 2023. doi: 10.1109/SSCI52147.2023.10371880.

R. P. Prager and H. Trautmann. Nullifying the inherent bias of non-invariant exploratory landscape analysis features. In *Applications of Evolutionary Computation*, page 411–425, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-30228-2. doi: 10.1007/978-3-031-30229-9_27. URL `https://doi.org/10.1007/`

978-3-031-30229-9_27.

Q. Renau, C. Doerr, J. Dreo, and B. Doerr. Exploratory landscape analysis is strongly sensitive to the sampling strategy. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, pages 139–153, 09 2020. ISBN 978-3-030-58115-2.

R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 01 1997. doi: 10.1023/A:1008202821328.

Y. Tian, S. Peng, X. Zhang, T. Rodemann, K. C. Tan, and Y. Jin. A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks. *IEEE Transactions on Artificial Intelligence*, 1(1):5–18, 2020. doi: 10.1109/TAI.2020.3022339.

L. van der Maaten and G. Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605, 11 2008.

D. Vermetten, F. Ye, T. Bäck, and C. Doerr. Ma-bbob: Many-affine combinations of bbob functions for evaluating automl approaches in noiseless numerical black-box optimization contexts. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, GECCO '23, page 813–821, New York, NY, USA, 06 2023. Association for Computing Machinery.

U. Škvorc, T. Eftimov, and P. Korošec. Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, 90:106138, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2020.106138. URL https://www.sciencedirect.com/science/article/pii/S1568494620300788.

U. Škvorc, T. Eftimov, and P. Korošec. The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1139–1146, 2021. doi: 10.1109/CEC45853.2021.9504739.

U. Škvorc, T. Eftimov, and P. Korošec. Transfer learning analysis of multi-class classification for landscape-aware algorithm selection. *Mathematics*, 10(3), 2022. ISSN 2227-7390. doi: 10.3390/math10030432. URL https://www.mdpi.com/2227-7390/10/3/432.