Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions

Quentin Renau, Johann Dréo, Carola Doerr, Benjamin Doerr

Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions

Quentin Renau[1,2], Johann Dreo[1], Carola Doerr[3], and Benjamin Doerr[2]

[1] Thales Research & Technology, Palaiseau, France
[2] École Polytechnique, Institut Polytechnique de Paris, CNRS, LIX, France
[3] Sorbonne Université, CNRS, LIP6, Paris, France

Abstract. Facilitated by the recent advances of Machine Learning (ML), the automated design of optimization heuristics is currently shaking up evolutionary computation (EC). Where the design of hand-picked guidelines for choosing a most suitable heuristic has long dominated research activities in the field, automatically trained heuristics are now seen to outperform human-derived choices even for well-researched optimization tasks. ML-based EC is therefore not any more a futuristic vision, but has become an integral part of our community.
A key criticism that ML-based heuristics are often faced with is their potential lack of explainability, which may hinder future developments. This applies in particular to supervised learning techniques which extrapolate algorithms' performance based on exploratory landscape analysis (ELA). In such applications, it is not uncommon to use dozens of problem features to build the models underlying the specific algorithm selection or configuration task. Our goal in this work is to analyze whether this many features are indeed needed. Using the classification of the BBOB test functions as testbed, we show that a surprisingly small number of features – often less than four – can suffice to achieve a 98% accuracy. Interestingly, the number of features required to meet this threshold is found to decrease with the problem dimension. We show that the classification accuracy transfers to settings in which several instances are involved in training and testing. In the leave-one-instance-out setting, however, classification accuracy drops significantly, and the transformation-invariance of the features becomes a decisive success factor.

Keywords: Exploratory Landscape Analysis · Feature Selection · Black-Box Optimization.

1 Introduction

Evolutionary algorithms and other iterative optimization heuristics (IOHs) are classically introduced as frameworks within which a user can gather some modules to instantiate an algorithm. For instance, the *design* of an evolutionary algorithm requires to choose the population size, the variation and selection operators in use, the encoding structure, fitness function penalization weights, etc.

This highly flexible design of IOHs allows for efficient abstractions but comes at the burden of having to solve an additional (meta-)optimization problem. *Automated design* of heuristics aims at solving this problem by providing data-driven recommendations which IOH shall be employed for a given optimization problem and how it shall be configured. Automated IOH design has proven its promise in numerous applications, see [7,9,20,3,12] for examples and further references.

A common critique of machine-trained automated algorithm design is its potential lack of explainability. That is, the general fear is that by relying on automated design approaches, we may be loosing intuition for *why* certain recommendation are made – a key driver for the development of new optimization approaches. This fear is not without any reason: the vast majority of automated algorithm design studies fall short in this explainability aspect.

Our Contribution. Our work aims at providing paths to narrowing this important gap, by studying which information the trained models actually need to achieve convincing performance. As testbed we chose the automated classification of optimization problems through exploratory landscape analysis (ELA). We show that very small feature sets can suffice to reliably discriminate between various optimization problems and that these sets are robust with respect to the classifiers and function instances.

Apart from the explainability aspect, our findings have important consequences also for the efficiency of automated algorithm design: smaller feature sets are faster to compute and they can drastically reduce the time spent in the training phase. Another advantage of feature selection is that the classification or regression accuracy can *increase*.

Background and Motivation. ELA was introduced in [17] with the objective to gain insights about the properties of an unknown optimization problem. Instead of relying on expert knowledge, the keystone of ELA are computer-generated *features* that are based on sampling the decision space. With the purpose of enhancing the effectiveness of this approach, several additional features have been introduced since. A good selection of these features are automatically computed by the R package *flacco* [14], see Sec. 2 for more details.

We chose classification as task, because it offers a very clean setting in which the results are easily interpretable. Classification has a straightforward performance measure, the *classification accuracy*, i.e., the fraction of items that are classified correctly. Additionally, the classification accuracy is a good way of estimating the expressiveness of ELA feature sets, i.e., their ability to discriminate between different problems [26]. A proper classification furthermore plays an important role also in many other ML tasks, including the selection and configuration of algorithms, so that a good classification accuracy can be expected to provide good results also for these tasks.

Related Work. Given the mentioned speed-up and the better performance that one can expect from smaller feature sets, feature selection is not new, but rather standard in automated algorithm design. However, most related works still use a relatively large number of features, hindering explainability of the

trained models. Among the ELA-based applications in EC, the following ones have used the smallest feature portfolios.

Muñoz and Smith-Miles [19] compute the co-linearity between landscape features with the idea that if two features are strongly co-linear, they carry the same type of information about the landscape. Applying this procedure, nine features were kept for further analysis: the adjusted coefficient of determination of a linear regression model including interactions [17], the adjusted coefficient of determination of a quadratic regression model [17], the ratio between the minimum and maximum absolute values of the quadratic term coefficients in the quadratic model, the significance of D-th and first order [29], the skewness, kurtosis and entropy of the fitness function distribution [17], and the maximum information content [22].

Another method to perform feature selection is the use of search algorithms. In their work, Kerschke and Trautmann [12] compare four different algorithms, a greedy forward-backward selection, a greedy backward-forward selection, a $(10 + 5)$-GA and a $(10 + 50)$-GA. The smallest feature sets considered in their algorithm selection setting have a size of eight features: three features from the y-distribution feature set [17] (skewness, kurtosis, and number of peaks), one level set feature [17] (the ratio of mean misclassification errors when using a linear (LDA) and mixed discriminant analysis (MDA)), two information content features [22] (the maximum information content and the settling sensitivity), one cell mapping feature [13] (the standard deviation of the distances between each cell's center and worst observation), and one of the basic features (the best fitness value within the sample). This result is still considerably larger than the sets we will identify as promising in our work.

Saini *et al.* [28] and Lacroix and McCall [15] also use reduced feature sets, but do not expand on how these have been derived.

Availability of Our Data. All our project data is available at [27].

2 Problem Classification via Majority Judgment

Our primary objective is to analyze the number of features that are needed to correctly classify the 24 BBOB functions from the COCO benchmark environment and their robustness across several dimensions and sample sizes. We describe in this section the benchmark set, the experimental procedure, and the classification scheme.

The 24 BBOB Benchmark Problems. A standard benchmark environment for numerical black-box optimization is the COCO (**CO**mparing **C**ontinuous **O**ptimizers) platform [6]. From this environment, we consider the BBOB suite, a set of 24 noiseless problems. For each BBOB problem, several instances are available, which are obtained from a "base" function via translation, rotation and/or scaling transformations [6]. Each problem instances is a real-valued function $f : [-5, 5]^d \to \mathbb{R}$. Problems scale for arbitrary dimensions d. In our experiments, we consider six different dimensions, $d \in \{5, 10, 15, 20, 25, 30\}$,

and we focus on the first five instances of each problem (first instance in Sec. 3. In abuse of notation, we shall often identify the functions by their ID $1, \ldots, 24$.

Computation of Feature Values via flacco. For the feature value approximation, we sample for each of the 24 functions f a number n of points $x^{(1)}, \ldots, x^{(n)} \in [-5, 5]^d$, and we evaluate their function values $f(x^{(1)}), \ldots, f(x^{(n)})$. The set of pairs $\{(x^{(i)}, f(x^{(i)})) \mid i = 1, ..., n\}$ is then fed to the *flacco* package [14], which returns a vector of features. The *flacco* package covers a total number of 343 features [9], which are grouped into 17 feature sets. However, some of these features are often omitted in practice because they require adaptive sampling [2,12,18,24], while other features have previously been dismissed as non-informative for the BBOB functions [13,26]. After removing these sets from our test bed, we are left with six feature sets: *dispersion* (disp [16]), *information content* (ic [22]), *nearest better clustering* (nbc [10]), *meta model* (ela_meta [17]), *y-distribution* (ela_distr [17]), and *principal component analysis* (pca [14]). But even if this selection reduces the number of features to 46, a full enumeration of all subsets for all sizes $c \leq 46$ would still be computationally infeasible (since we need to train and test a classification model for each such set). We therefore need to reduce the set of eligible features further. To this end, we build on the work presented in [26], in which we studied the *expressiveness* of these 46 features. Based on this work we select four features. We add to this selection another six features, one per each of the feature set mentioned above (to ensure a broad diversity of features) and again giving preference to the most expressive ones and to features invariant to BBOB transformations [30]. This leaves us with the following ten features. We indicate in this list by ✓and - whether or not a feature is considered invariant under transformation according to [30] (first entry) and according to our data (second entry), respectively. Note here that the setting used in [30] is slightly different from the instances used in BBOB, mostly due to different ways to handle boundary constraints. The assessment can therefore differ.

1. **disp.ratio_mean_02** [✓,✓] (`disp`) computes the ratio of the pairwise distances of the points having the best 2% fitness values with the pairwise distances of all points in the design.
2. **ela_distr.skewness** [✓,✓] (`skew`) computes the skewness coefficient of the distribution of the fitness values. This coefficient is a measure of the asymmetry of a distribution around its mean.
3. **ela_meta.lin_simple.adj_r2** [✓,✓] (`lr2`), which computes the adjusted correlation coefficient R^2 of a linear model fitted to the data.
4. **ela_meta.lin_simple.intercept** [✓,-] (`int`), the intercept coefficient of the linear model.
5. **ela_meta.lin_simple.coef.max** [-,-] (`max`), the largest coefficient of the linear model that is not the intercept coefficient.
6. **ela_meta.quad_simple.adj_r2** [✓,✓] (`qr2`), the adjusted correlation coefficient R^2 of a quadratic model fitted to the data.
7. **ic.eps.ratio** [-,✓] ($\varepsilon_{\text{ratio}}$), the half partial information sensitivity.
8. **ic.eps.s** [-,✓] (ε_s), the settling sensitivity.

9. **nbc.nb_fitness.cor** [✓,✓] (`nbc`), the correlation between the fitness values of the search points and their indegree in the nearest-better point graph.
10. **pca.expl_var_PC1.cov_init** [✓,✓] (`pca`), which measures the importance of the first principal component of a Principal Component Analysis (PCA) over the sample points in the whole search space.

Normalization of Feature Values. The value of each feature is normalized between 0 and 1 where 0 (resp. 1) correspond to the smallest (resp. largest) value encountered in the approximated feature values. This normalization is performed independently for each dimension, each sample size, and each classifier used in this paper.

Sampling Strategy. Based on an extension of the preliminary experiments reported in [25] we use a quasi-random distribution to sample the points $x^{(1)}, \ldots, x^{(n)}$ from which the feature values are computed. More precisely, we use Sobol' sequences [32], which we obtain from the Python package *sobol_seq* (version 0.1.2), with randomly chosen initial seeds.

We sample a total number of 100 independent Sobol' designs, which leaves us with 100 feature value vectors per each function. Fig. 1 provides an impression of the distribution of these feature values. Plotted are here approximated values for the `lr2` feature. The comparison shows that the dispersion slightly decreases with the dimension, which is quite surprising in light of the lower density of the points in higher dimensions. We also see that the median values are not stable across dimensions. Some functions (F5 of course, which is correctly identified as a linear function, but also F16, F19, and F20, for example) show a high concentration of feature value approximations, whereas other functions show much larger dispersion within one dimension (e.g., F12, F15, F17, F18) or between different dimensions (F2, F11, F24).

Sample Size. To study the effect of the sample size on the number of features needed to correctly classify the 24 BBOB functions, we conduct experiments for seven different values of n, namely $n \in \{30d, 50d, 100d, 250d, 650d, 800d, 1000d\}$. We note here that a linear scaling of the sample size is the by far most common choice, see, for example, [3,11,12].

Feature Selection. We apply a *wrapper method*, i.e., we actually train a classifier for every considered subset of features. For a given sample size and a given dimension, we train and test all $\binom{10}{c}$ possible subsets of size c starting with $c = 1$. If none of these size-c subsets achieves our target accuracy, we move on to the size $c + 1$ subsets. As soon as a sufficiently qualified subset has been identified, we continue to evaluate all size-c subsets, but stop the selection process thereafter. This full enumeration of all possible feature combinations for a given size c allows us to investigate the *robustness* of the feature selection. Ideally, we would like to see that the feature sets achieving our 98% accuracy threshold (this will be introduced below) are stable across the different sample sizes. Robustness with respect to the dimension is much less of a concern to us, since the problem dimension is typically known and can be used for the choosing the feature ensemble that shall be applied to characterize the problem.

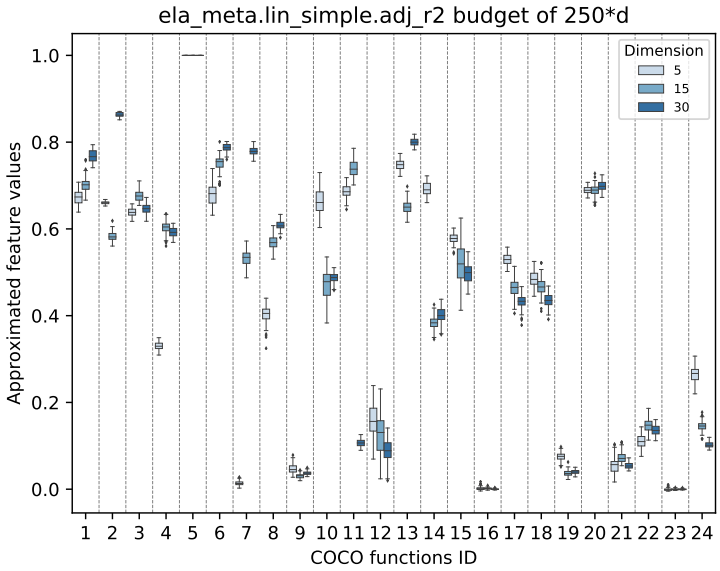


Fig. 1: Distribution of the feature values for the `lr2` feature for different dimensions. Each feature value is computed from $250 \times d$ samples and each boxplot represents results of 100 independent feature computations.

Validation Procedure and Target Classification Accuracy. In our experiments, we use 80 randomly chosen feature vectors (per function) to train a classification model, and we use the remaining $24 \times 20 = 480$ feature vectors for testing. For each of these 480 test cases we store the true function ID (i.e., the ID of the function that the feature value originates from) and we store the ID of the function that the classifier matches the feature vector to. From this data we compute the *overall classification accuracy*.

We repeat this procedure of splitting the set of all feature vectors into 80 training and 20 test instances 20 times; i.e., we repeat 20 times a **random sub-sampling validation**. We require that the overall classification accuracy for each of the 20 validations is **at least 98%**. That is, a feature set is eligible if, in each of the 20 random sub-sampling validation runs, it misclassifies at most 10 out of the 480 tested feature vectors. Feature combinations achieving a smaller classification accuracy in one of the validation runs are immediately discarded.

Classification Model. In the main part of this work, we use a Majority Judgment classifier [1]. A cross-validation with decision trees and KNN classifiers will be presented in Sec. 4.

The Majority Judgment classifier works as follows. Let $\Phi = \{\varphi_1, \ldots, \varphi_k\}$ be the set of features for which we want to know whether it achieves our 98% target precision requirement. We consider one of the independent subsampling

	Function ID (index j)			
	1	**2**	...	**24**
Feature φ_1	0	0.7	...	0.7
Feature φ_2	0.2	0.6	...	0.5
Feature φ_3	0.6	0.8	...	0.2
Median distance D_j	0.2	0.7	...	0.5

Table 1: Example for the Majority Judgment classification scheme with three features. The values in the table are the distances of the measured feature value ζ_i to the median feature values $M(i,j)$ of the training set. The median values are reported in the last line. The ID of the function minimizing this median distance D_j is the output of the Majority Judgment classifier.

validation runs. That is, for each function we randomly select 80 out of the 100 feature vectors. Denote by $\varphi_{i,j,r}$ the r-th estimated value for feature φ_i for the j-th BBOB function, the set $\{(\varphi_{i,j,r}, j) \mid i = 1,\dots,k, j = 1,\dots,24, r = 1,\dots,80\}$ describes the full set of training data. From this data we compute for each of the 24 functions $j = 1,\dots,24$ and for each feature $\varphi_i \in \Phi$ the median value

$$M(i,j) := \mathbb{M}\left(\{\varphi_{i,j,r} \mid r = 1,\dots,80\}\right).$$

This gives us a set of $24k$ values $M(i,j)$ and concludes the *training step*.

In the *testing step* we apply an *approval voting mechanism* [4] to each of the 480 test instances. Approval voting mechanisms are single-winner systems where the winner is the most-approved candidate among the voters. From this class of approval voting mechanism we choose *Majority Judgment* [1] —a voting techniques which ensures that the winner between three or more candidates has received an absolute majority of the scores given by the voters.

To apply Majority Judgment to our classification task, we do the following. We recall that the task of the classifier is to output, for a given feature vector $\zeta = (\zeta_i)_{i=1}^k$, the ID of the function that it believes this feature vector to belong to. To this end, it first computes for each of the k features i and for all 24 functions j the absolute distances $d_{i,j} := |\zeta_i - M(i,j)|$. Tab. 1 presents an example for what the distances may look like. We then compute for each function the median of these distances, by setting $D_j(\zeta) := \mathbb{M}\left(\{d_{i,j} \mid i = 1,\dots,k\}\right)$. The cells with these median values are highlighted with a blue background in Tab. 1, and the values $D_j(\zeta)$ are reported in the last line. The classifier outputs as predicted function ID the value j for which the distance $D_j(\zeta)$ is minimized. This cell is highlighted in yellow background color.

Computation Time. To give an impression of the computational resources required for our experiments, we report that the computation of the 100 5-dimensional feature vectors requires around 6 CPU hours, whereas the computation of the 25-dimensional feature vectors takes about 1221 CPU hours. Training and testing the classifier takes between 1 second and 3 hours, depending on the setting. In total, we have invested around 432 CPU days for computing the data presented in this work.

	Sample Size						
dimension	30d	50d	100d	250d	650d	800d	1000d
5	-	-	-	4	4	-	2
10	-	-	-	4	1	2	1
15	-	-	6	4	2	2	2
20	-	-	6	2	1	1	2
25	1	1	1	1	1	1	1
30	-	6	2	1	1	2	2

Table 2: Feature combination size achieving 98% classification accuracy in all 20 runs.

3 Feature Sets Achieving 98% Classification Accuracy

The portfolios of features for which we obtained the desired 98% classification accuracy for each of the 20 random sub-sampling validation runs are presented in Tab. 3. For convenience, their sizes are summarized in Tab. 2.

Our first, and most important, finding is that we can actually classify the BBOB functions with very few features. However, we also see that the existence of such portfolios requires a sufficient sample size. For $d \in \{5, 10, 15, 20\}$, none of the 2^{10} possible portfolios based on size-$30d$ and size-$50d$ feature approximations could achieve the 98% accuracy threshold.

We also see that, as expected, the size of the minimal portfolio achieving the target precision decreases with increasing sampling size. A few exceptions to this rule exist:

- No combination in $d = 5$ with $n = 800$ samples achieved the target precision.
- In $d = 10$ we see that a single feature, the intercept feature `int`, suffices to classify with 98% accuracy when the sampling size is $650d$ and $1000d$. For $800d$, however, this feature does not achieve the threshold. A detailed analysis of the classification accuracy achieved with this feature will be given in Fig. 2.
- In $d = 15$, the $\varepsilon_{\text{ratio}}$ information content feature classifies properly when the sample size equals $n = 800d$, but for $n = 1000d$, one additional feature is needed to pass the 98% accuracy threshold.
- In $d = 20$ a single feature suffices for $n = 650d$ and $n = 800d$, but for $n = 1,000d$ an additional feature is needed to achieve the target accuracy.

Overall, we see that for ten settings a single feature suffices for proper classification. An additional seven cases can be solved by a combination of two features. It seems counter-intuitive that in almost all cases the size of the smallest admissible portfolio decreases with increasing dimension. However, as already discussed in the context of Fig. 1, the dispersion of some feature values *decreases* with increasing dimension – an effect that is interesting in its own right. Without going into much detail here, we note that this effect is further intensified when using a properly scaled sampling size that maintains the same sampling density across dimensions.

		Feature									
d	n	int	lr2	qr2	max	ε_s	$\varepsilon_{\text{ratio}}$	disp	skew	pca	nbc
5	**30**d										
	50d										
	100d										
	250d		X	X			X				X
	650d		X	X			X				X
	800d										
	1000d		X				X				
10	**30**d										
	50d										
	100d										
	250d		XO	XO			XO			O	X
	650d	X									
	800d			X			X				
	1000d	X									
15	**30**d										
	50d										
	100d	X	X		X		X	X			X
	250d		X		X		X			X	
	650d		X		H	O	XH			O	
	800d						X				X
	1000d						XO	X		O	
20	**30**d										
	50d										
	100d	X	X		X		X	X			X
	250d			X			X				
	650d	X									
	800d						X				
	1000d			X			XO			O	
25	**30**d	X									
	50d	X									
	100d	X									
	250d	X									
	650d	X					O				
	800d	X									
	1000d	X						*M*			
30	**30**d										
	50d	X	X		X		X	X		X	
	100d		X				XO			O	
	250d	X									
	650d	X									
	800d		O	X			XO	*M*			
	1000d		O	X	H		XOHV	*M*			V

Table 3: Feature combinations achieving the 98% classification accuracy threshold in all 20 runs. Features with the same symbol (X,O,H,V) belong to the same combination. Results are grouped by dimension d and by the sample size n used to approximate the feature values. Blank rows are for (d,n) settings for which all 2^{10} feature sets failed. M = missing data (due to coronavirus measures in France, we have lost access to cluster and data.)

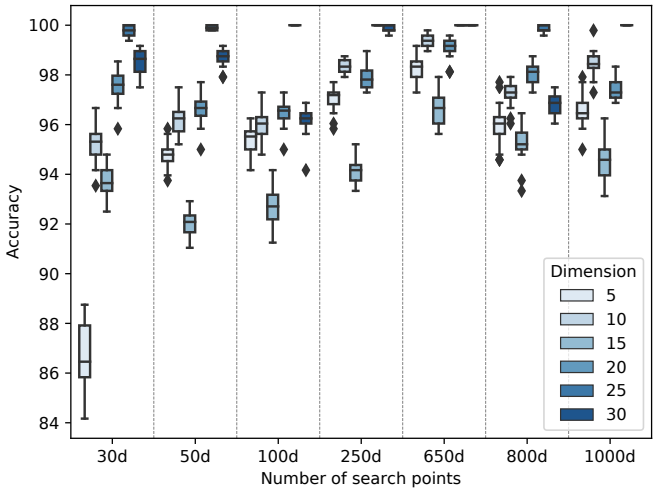


Fig. 2: Distributions of intercept feature accuracy by dimension and sample size

Robustness of the feature combinations with respect to dimension and sample size. Looking at the robustness of the selected combinations over the dimensions and the sample sizes, we observe the following.

One feature, the intercept feature `int`, is involved in 15 out of the 28 (d, n) pairs for which a successful feature portfolio could be found. This feature, in contrast, is rarely present in other combinations of size $|c| > 1$. To shed more light on its expressive power, we present in Fig. 2 the distributions of the classification accuracy for the various (d, n) combinations. Aggregated over all dimensions and all sample sizes, the median accuracy of the `int` feature is 96%. Even if the feature does not always reach our threshold of 98%, it is worth noting that its performances is almost always above 90%. Therefore, this feature is very expressive, and this across all tested dimension and sample sizes. Another interesting observation from Fig. 2 is that the classification accuracy is not monotonic in the dimension. In all but one case ($n = 30d$), the $d = 15$ results are worse than those for the other dimensions. As already seen in Tab. 3, for $n = 250 \times d$ we always have very good classification accuracy.

The most frequent feature is $\varepsilon_{\text{ratio}}$, which is present in almost all combinations of size $|c| \geq 2$. We count 21 successful combinations of size $|c| \geq 2$ and $\varepsilon_{\text{ratio}}$ appears in 20 of these combinations regardless of the dimension and the sample size. In total, it appears in successful portfolios for 17 out of the 28 (d, n) combinations for which a successful subset had been found. The $\varepsilon_{\text{ratio}}$ feature is very useful for our classification task.

The skewness feature *skew*, in contrast, does not appear in any of the portfolios of the smallest size.

Classification Accuracy When Using All flacco Features. We compare the results presented above with the classification accuracy achieved by the Majority Judgment voting scheme using the whole set of 46 fea-

tures described in Sec. 2. We perform the same sub-sampling validation as above. Interestingly, none of tests performed on the pairs (d,n) with $n \in \{30d, 50d, 100d, 250d, 650d, 1000d\}$ and $d \in \{5, 10, 15, 20, 25, 30\}$ met our required target precision of 98% for each of the 20 runs. We can thus conclude that, in addition to the gain in explainability, the selection of features for supervised-ELA approaches provide better performances, and – as we shall discuss below – also come at a much smaller computational cost.

4 Robustness with Respect to the Classifier

Having identified feature portfolios that reliably classify the BBOB functions with at least 98% accuracy when using Majority Judgment (MJ), we now investigate how robust this accuracy is with respect to the choice of the classifier. To this end, we apply the same classification routine as above, but now using *decision trees* (DT) and *K Nearest Neighbors (KNN)* as for classification. We use off-the-shelf implementations from the *scikit learn* Python package [23, we use version 0.21.3]. Our goal being in investigating robustness, we do not perform any hyper-parameter tuning for these two classifiers. For the KNN classifier we use $K = 5$. For all classifications with a reduced portfolio of features, if multiple combinations are available, only the one marked with X in Tab. 3 will be used.

Both KNN and decision trees perform as well as our classifier when trained and tested with the small portfolios from Tab. 3, i.e., they both reach at least 98% classification accuracy in every run except for the decision trees trained with only one feature, for which the accuracy drops to around 62% in every run. Fig. 3 summarizes the classification accuracy of the three classifiers for the case that features are based on $n = 250d$ samples, for the portfolios described in Tab. 3. Performance is indeed very robust with respect to the classification mechanism.

Running Time. While training and testing were made in around 4 seconds for the DT and for the MJ voting scheme, the KNN classifier needed around 12 seconds to complete the 20 sub-sampling validation runs.

Gain over Full Feature Set. We now study how much we gain in terms of computation time when we compute, train, and test the three classifiers (MJ, DT, and KNN) on the selected feature sets only.

To quantify this gain, we train all three classifiers with the full set of 46 features mentioned in Sec. 2. We first observe that the decision tree classifier has the best performances among the three classifiers in terms of accuracy. It achieves at least 99% classification accuracy. For KNN, in contrast, performances drops below our 98% threshold precision on several runs, resulting in a median classification accuracy (over all tests) of around 97%. The results for KNN align, as already briefly touched upon in Sec. 3, with those obtained using MJ, where none of the tests produced 20 runs in which the threshold was reached.

In terms of computation time, we observe significant differences between the small feature portfolios and the full *flacco* set. As already commented in Sec. 2, the computation of the feature values can be very time-consuming. Reducing the

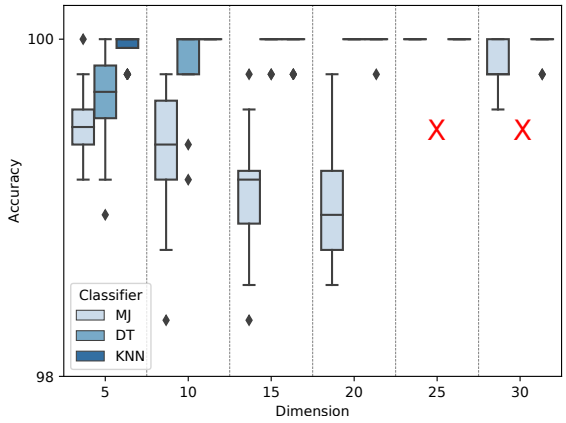


Fig. 3: Classification accuracy for the feature portfolios from Tab. 3 for budget $250d$. Results are sorted by dimension and classifier and are for 20 random sub-sampling validation runs. Training and testing is done on the first instance of each function only. The X corresponds to settings that did not achieve the 98% threshold.

number of features therefore reduces the running time of the feature extraction. However, the savings are even bigger when comparing the cost of training (and testing) the classifiers. For decision trees, the execution of the whole classification pipeline takes 3000 times longer than with the small portfolios – around 3 CPU hours instead of a few seconds. For KNN, the total cost is comparable, also around 3 CPU hours for training and testing the classifiers for the 20 sub-sampling validation runs. For the MJ classifier, the overall running time is only around 35 CPU minutes – which is still way above the time needed for the small portfolios.

Thus, overall, the reduced portfolios resulted not only in much faster computation times, but achieved also better classification accuracy.

5 Robustness with Respect to the Problem Instances

The discussion above focused on classifying the first instance of the BBOB functions, and we now investigate how robust the selection is with respect to different instances of the same problems. Concretely, we investigate classification accuracy when performing the same random sub-sampling validation routine as above to the set of features computed for the first five instances of the BBOB functions. In this experiment, we keep 80% of feature values for each instance for training the classifier, and we test on the remaining ones. In a second step we then test transferability, by performing a *leave-one-instance-out (LOIO) cross-validation.* In this setting, the classifiers are trained on four instances of each function and tested on the remaining one. We use the portfolios marked by an X in Tab. 3, and compare to classification accuracy when using all ten features. In the following,

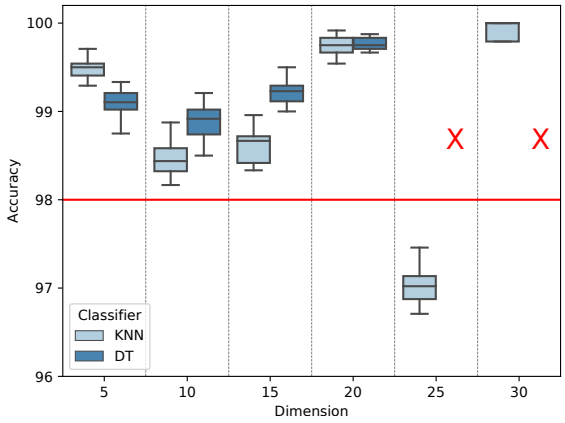
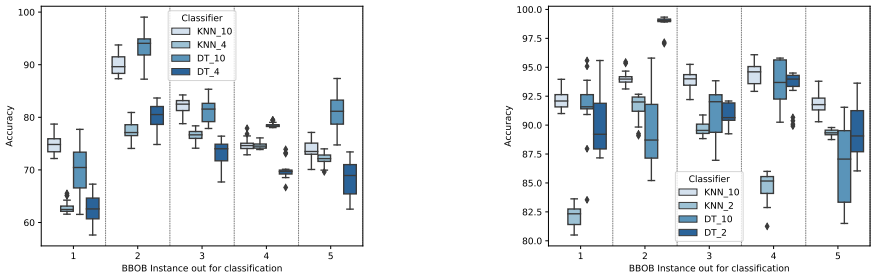


Fig. 4: Classification accuracy of DT and KNN classifiers when applied to the first five instances of the 24 BBOB functions. Feature values are computed from $250d$ samples, for the portfolios marked by an X in Tab. 3. Cases with poor performance are marked by a red X.

MJ voting is excluded as, by design, it is not suited to work with multiple distributions coming from different instances. Hence, only DT and KNN classifiers will be used in this section.

Fig. 4 aggregates the results obtained for the first classification task, where we take feature values from each or the first five instances. As in Fig. 3, DT performs badly in $d = 25$ and $d = 30$, where classification is only based on the intercept feature. For these cases, the median accuracy is 45% and 62%, respectively. Since the intercept feature is not invariant to fitness function transformations, the worsened performance is no surprise. In contrast, the median classification accuracy is above 98% for all portfolios with at least two features. We also note that KNN in dimension $d = 25$ does not reach our 98% threshold, but still achieves good performances with an average 97% accuracy.

Fig. 5 presents the classification accuracy achieved by KNN and DT in the LOIO setting. Fig. 5a is for features `lr2`, `qr2`, $\varepsilon_{\text{ratio}}$, and `nbc` computed from $650d$ samples in $d = 5$ and the Fig. 5b is for the two features `qr2` and $\varepsilon_{\text{ratio}}$ computed from $250d$ samples in $d = 20$. For comparison, we also plot the classification accuracy achieved when using all ten features listed in Sec. 2. For most settings, the accuracy obtained with the set of ten features is better than that achieved for the smaller portfolios. For the $650d$ setting, this is the case for all instances. For the $250d$ setting, DT performs better with the smaller portfolio when instance 1 or instance 3 is left out. The performance loss when using the reduced feature set is particularly drastic for KNN when instance 1 is left out (both cases), when instance 2 is left out ($650d$ case), and when instance 4 is left out ($250d$ case). Interestingly, for DT in the $650d$ setting, the largest performance losses occur when leaving instance 2 or 5 out. The average loss in classification accuracy is



(a) $650d$ samples, $d = 5$ (b) $250d$ samples, $d = 20$

Fig. 5: Classification accuracy of KNN and DT in the leave-one-instance-out setting. The subscripts _2, _4, and _10 refer to the size of the feature portfolio.

5% and 4% for KNN in the $650d$ and the $250d$ case, respectively. For DT, the average loss in the $650d$ case is 10% and the average gain in the $250d$ case is 2%.

We conclude that the feature selection is robust when studying different instances, except for those portfolios which consist only of a single feature. For the (arguably more interesting) LOIO setting, however, classification accuracy drops, but non-homogeneously for the different instances. We recommend using the larger feature portfolio in this case.

6 Conclusions

Our ambition to build small feature sets is driven by the desire to obtain models that are (at least to some degree) human-interpretable. While our study certainly has several limitations, as only one test bed is considered, it nevertheless shows that the number of features needed to successfully classify the BBOB functions is surprisingly low. Our main direction for future work is an application of the small feature sets to automated algorithm design tasks. [8] shows promising performance of the selected feature portfolio presented in Sec. 2 for automated performance regression and per-instance algorithm selection, results that we wish to detail further based on the results presented in Sec. 3. Our next important goal will then be to uncover *how* the performance of a given solver depends on the selected features, by taking a closer look at the trained regression models. With small feature sets, there is reasonable hope that we can identify meaningful correlations.

We are targeting, in the mid-term perspective, classifiers and automated algorithm design techniques that work well on highly constrained problems and which can cope with discontinuities. Extending the results of this work to such problems forms another important next step.

Other interesting directions for future work include the investigation of new features recently proposed in the literature, (such as, for example, the SOO-based features [5]). We also plan on a closer inspection of the classification results

presented above, particularly with respect to the mis-classifications. Functions that are wrongly classified more often than others (a preliminary investigation showed that these mis-classification rates depend on the dimension. In dimensions $d = 10$, for example, function 17 is confused with function 21 in 30% of the tests even when a sample size of $n = 10,000$ is used.) Such data can be used, in particular, for training set selection, but also for the *generation* of new problem instances for which the algorithms show some behavior not observable on other instances of the same collection [31,21].

Acknowledgments. We thank Cédric Buron, Claire Laudy, and Bruno Marcon for providing the implementation of the Majority Judgment classifier.

References

1. Balinski, M., Laraki, R.: Judge: Don't vote! Operations Research **62**(3), 483–511 (2014)
2. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Surrogate assisted feature computation for continuous problems. In: In: LION. pp. 17–31. Springer (2016)
3. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per instance algorithm configuration of CMA-ES with limited budget. In: GECCO. pp. 681–688. ACM (2017)
4. Brams, S., Fishburn, P.: Approval voting, 2nd edition. Springer (2007)
5. Derbel, B., Liefooghe, A., Vérel, S., Aguirre, H., Tanaka, K.: New features for continuous exploratory landscape analysis based on the SOO tree. In: FOGA. pp. 72–86. ACM (2019)
6. Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: a platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software pp. 1–31 (2020)
7. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning - Methods, Systems, Challenges. Springer (2019)
8. Jankovic, A., Doerr, C.: Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In: In: GECCO. pp. 841–849. ACM (2020)
9. Kerschke, P., Hoos, H., Neumann, F., Trautmann, H.: Automated Algorithm Selection: Survey and Perspectives. Evolutionary Computation **27**(1), 3–45 (Mar 2019)
10. Kerschke, P., Preuss, M., Wessing, S., Trautmann, H.: Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In: GECCO. pp. 265–272. ACM (2015)
11. Kerschke, P., Preuss, M., Wessing, S., Trautmann, H.: Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In: GECCO. pp. 229–236. ACM (2016)
12. Kerschke, P., Trautmann, H.: Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evolutionary Computation **27**(1), 99–127 (2019)
13. Kerschke, P., Preuss, M., Hernández Castellanos, C., Schütze, O., Sun, J.Q., Grimme, C., Rudolph, G., Bischl, B., Trautmann, H.: Cell mapping techniques for exploratory landscape analysis. Advances in Intelligent Systems and Computing **288**, 115–131 (2014)
14. Kerschke, P., Trautmann, H.: Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco. In:

Applications in Statistical Computing: From Music Data Analysis to Industrial Quality Improvement, pp. 93–123. Springer (2019)
15. Lacroix, B., McCall, J.A.W.: Limitations of benchmark sets and landscape features for algorithm selection and performance prediction. In: GECCO. pp. 261–262. ACM (2019)
16. Lunacek, M., Whitley, D.: The dispersion metric and the CMA evolution strategy. In: GECCO. p. 477. ACM (2006)
17. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory Landscape Analysis. In: GECCO. pp. 829–836. ACM (2011)
18. Morgan, R., Gallagher, M.: Sampling Techniques and Distance Metrics in High Dimensional Continuous Landscape Analysis: Limitations and Improvements. IEEE Transactions on Evolutionary Computation **18**(3), 456–461 (Jun 2014)
19. Muñoz, M.A., Smith-Miles, K.: Effects of function translation and dimensionality reduction on landscape analysis. In: IEEE CEC. pp. 1336–1342. IEEE (2015)
20. Muñoz, M.A., Sun, Y., Kirley, M., Halgamuge, S.K.: Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. Inf. Sci. **317**, 224–245 (2015)
21. Muñoz, M.A., Villanova, L., Baatar, D., Smith-Miles, K.: Instance spaces for machine learning classification. Machine Learning **107**(1), 109–147 (2018)
22. Muñoz, M., Kirley, M., Halgamuge, S.: Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content. IEEE Transactions on Evolutionary Computation **19**(1), 74–87 (Feb 2015)
23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. JMLR **12**, 2825–2830 (2011)
24. Pitra, Z., Repický, J., Holena, M.: Landscape analysis of Gaussian process surrogates for the covariance matrix adaptation evolution strategy. In: GECCO. pp. 691–699 (2019)
25. Renau, Q., Doerr, C., Dréo, J., Doerr, B.: Exploratory landscape analysis is strongly sensitive to the sampling strategy. In: PPSN. LNCS, vol. 12270, pp. 139–153. Springer (2020)
26. Renau, Q., Dreo, J., Doerr, C., Doerr, B.: Expressiveness and robustness of landscape features. In: GECCO (Companion). pp. 2048–2051. ACM (2019)
27. Renau, Q., Dreo, J., Doerr, C., Doerr, B.: Exploratory Landscape Analysis Feature Values for the 24 Noiseless BBOB Functions (2021). https://doi.org/10.5281/zenodo.4449934
28. Saini, B., López-Ibáñez, M., Miettinen, K.: Automatic surrogate modelling technique selection based on features of optimization problems. In: GECCO (Companion). pp. 1765–1772 (2019)
29. Seo, D., Moon, B.R.: An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. Evol. Comput. **15**(2), 169–198 (2007)
30. Skvorc, U., Eftimov, T., Korosec, P.: Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. Appl. Soft Comput. **90**, 106138 (2020)
31. Smith-Miles, K., Bowly, S.: Generating new test instances by evolving in instance space. Computers & OR **63**, 102–113 (2015)
32. Sobol', I.: On the distribution of points in a cube and the approximate evaluation of integrals. USSR Computational Mathematics and Mathematical Physics **7**(4), 86–112 (Jan 1967)