Performance Analysis of Continuous Black-Box Optimization Algorithms via Footprints in Instance Space

Mario A. Muñoz mario.munoz@monash.edu
School of Mathematical Sciences, Monash University, Clayton, Victoria 3800 Australia

Kate A. Smith-Miles kate.smith-miles@monash.edu
School of Mathematical Sciences, Monash University, Clayton, Victoria 3800 Australia

doi:10.1162/EVCO_a_00194

Abstract

This article presents a method for the objective assessment of an algorithm's strengths and weaknesses. Instead of examining the performance of only one or more algorithms on a benchmark set, or generating custom problems that maximize the performance difference between two algorithms, our method quantifies both the nature of the test instances and the algorithm performance. Our aim is to gather information about possible phase transitions in performance, that is, the points in which a small change in problem structure produces algorithm failure. The method is based on the accurate estimation and characterization of the algorithm footprints, that is, the regions of instance space in which good or exceptional performance is expected from an algorithm. A footprint can be estimated for each algorithm and for the overall portfolio. Therefore, we select a set of features to generate a common instance space, which we validate by constructing a sufficiently accurate prediction model. We characterize the footprints by their area and density. Our method identifies complementary performance between algorithms, quantifies the common features of hard problems, and locates regions where a phase transition may lie.

Keywords

Algorithm selection, black-box continuous optimization, exploratory landscape analysis, footprint analysis, performance prediction.

1 Introduction

The goal in a continuous optimization problem is to maximize or minimize a function whose input and output variables are real numbers, subject to some constraints. Often, these problems lack an algebraic expression, have noncalculable derivatives, exhibit uncertainty or noise, or may involve time-consuming simulations or experiments. Topologically, these problems may present local and global optima, large fluctuations in output value between similar inputs, and interdependencies between input variables. However, these problem features are unknown. All that is known is a black-box output response to a given input. Therefore, sampling the inputs and obtaining the output response is the only way to acquire information about the problem. This approach, known as black-box optimization, does not consider the internal workings of the function. As such, there is no guarantee of finding the exact global optimum, although this is often unnecessary in practice. More often, a target output is defined, which represents a reasonable improvement over current practice.

Manuscript received: May 21, 2015; revised: June 20, 2016 and August 17, 2016; accepted: September 2, 2016.

Having to solve these problems with limited resources and knowledge, as well as their profuse nature in practical fields (Lozano et al., 2011), has led to an increased number of reported algorithms for continuous black-box optimization over the last decades. This algorithmic diversity eclipses significant algorithmic innovations (Sörensen, 2015) and fosters the recycling of ideas (Weyland, 2010; Črepinšek et al., 2012; De Corte and Sörensen, 2013; Piotrowski et al., 2014). Thus, the algorithmic diversity compromises our ability to master—or even be familiar with—all algorithms (Hough and Williams, 2006). This would be irrelevant if any algorithm performed well on any problem. Unfortunately, the No Free Lunch Theorems (NFLT) prove that, even though all algorithms perform equally on average across the complete space of problems (Wolpert and Macready, 1997), some algorithms are preferable to others on specific problems (Schumacher et al., 2001; Langdon and Poli, 2007; Auger and Teytaud, 2010). While NFLTs were proven on deterministic algorithms and finite search domains, more recent work has shown their applicability to randomized algorithms in very large domains, such as those found in continuous optimization (Rowe et al., 2009; Lockett and Miikkulainen, 2016).

Selecting an appropriate algorithm for a given problem is at best cumbersome (Tang et al., 2014), even with expert knowledge on search algorithms, and skills in algorithm engineering and statistics (Blum et al., 2011). To facilitate this task, it is fundamental to have an objective assessment of the performance of a diverse set of algorithms. That is, we need to have knowledge of each algorithm's strengths and weaknesses (Langdon and Poli, 2007). However, our theoretical understanding of the performance of most algorithms on real-world problems is still limited, even after significant advances (Auger and Doerr, 2011). Therefore, the conventional approach to this issue is to analyze the performance of one or more algorithms on a suite of benchmark problems (Langdon and Poli, 2007). This approach is not issue-free given that no test suite can cover the whole range of difficulties encountered in black-box optimization (Ros, 2009b). A limited test suite may result in small differences in performance between algorithms, which obfuscates the source and nature of the differences (Langdon and Poli, 2007). Alternatively, custom problems can be generated such that they maximize the performance difference between two algorithms; hence, exaggerating their comparative strengths and weaknesses (Langdon and Poli, 2007). However, these comparisons explain only extreme cases, and tell us little about possible phase transitions in performance, that is, the points in which a small change in problem structure elicits poor algorithm performance. To deeply understand the link between problems and algorithms, an analysis of experimental performance should be linked to an assessment of similarities and differences between problems, which can be achieved only by characterizing their structure (Malan and Engelbrecht, 2014).

The extensive and successful work on algorithm selection for discrete and analytical problems (Smith-Miles, 2009; Hutter et al., 2014; Smith-Miles et al., 2014; Kotthoff, 2014) demonstrates the power of linking problem characteristics to the performance of algorithms. Most of this work is based on the framework by Rice (1976), which identifies four interrelated spaces.[1] First is the ill-defined problem space, $\mathcal{F}$, which contains all the relevant problems to be solved (i.e., continuous black-box functions to be minimized in this article). Second is the algorithm space, $\mathcal{A}$, which is composed of all

[1]We have modified Rice's notation to use f as function. In the original paper, x is a problem, $f(x)$ is a feature vector, A is an algorithm, $p(A, x)$ is a performance measure, $\mathcal{P}$ is the problem space, and $\mathcal{F}$ is the feature/characteristics space.

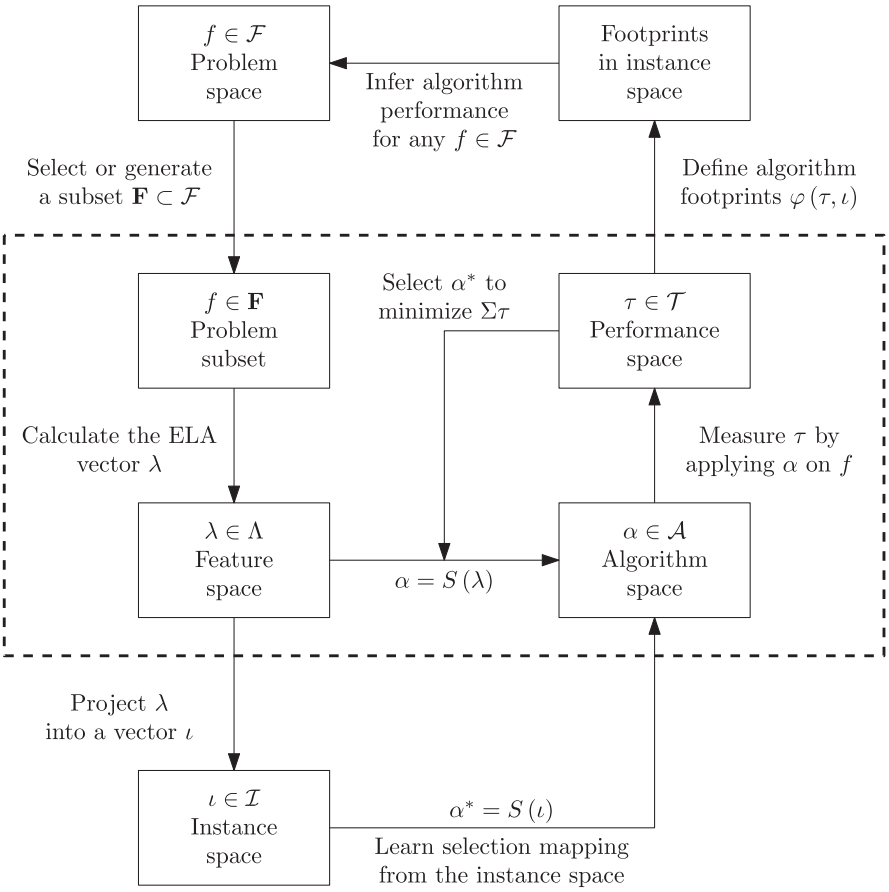

Figure 1: Algorithm selection framework by Smith-Miles et al. (2014), which extends the original of Rice (1976), shown in the box. The nomenclature has been adjusted for continuous optimization.

the algorithms applicable to the problems in $\mathcal{F}$, regardless whether they are successful or not. Third is the performance space, $\mathcal{T}$, which is the set of feasible values of τ, a score that measures the cost of using an algorithm $\alpha \in \mathcal{A}$ to solve a problem $f \in \mathcal{F}$. Fourth is the feature space, $\mathbf{\Lambda}$, which is defined by a set of measurable features that expose the complexities of the problems; hence, $\mathbf{\Lambda}$ is the key element in the implementation of the framework as it provides order to $\mathcal{F}$. However, $\mathbf{\Lambda}$ is high-dimensional; hence, hard to analyze. Therefore, Smith-Miles et al. (2014) extended Rice's framework by projecting the feature vector defining an instance into a two-dimensional vector, ι, in the instance space, $\mathcal{I} \subset \mathbb{R}^2$, for visualization. Then, τ is employed to identify the regions of $\mathcal{I}$ in which good or exceptional performance is expected from an algorithm. These regions, known as algorithm footprints (Corne and Reynolds, 2010), not only illustrate the links between problems and algorithms, but also provide information on the phase transitions. The extended framework is depicted in Figure 1.

This article provides a method for the objective assessment of a black-box optimization algorithm's strengths and weaknesses. Unlike related work on algorithm

assessment and comparison (Hansen et al., 2011; El-Abd, 2012; Rios and Sahinidis, 2013; Liao et al., 2015), the method quantifies both the nature of the test instances and the algorithm performance. The method is based on the accurate estimation and characterization of each algorithm's footprint—a region in the space of all possible instances of the problem where an algorithm is expected to perform well based on inference from empirical performance analysis. Therefore, the method is also useful to predict the likelihood of an algorithm being successful on a new problem, such as the work by Malan and Engelbrecht (2014), although, it also provides a visual representation that generalizes performance across areas of the problem space. The method follows an adaptation of Rice's framework originally proposed by Smith-Miles et al. (2014) for combinatorial optimization problems, which we have refined for its use with continuous black-box optimization problems. The framework has been successful in identifying complementary and individual strengths of several algorithms in other problem domains such as traveling salesman problem (Smith-Miles and Tan, 2012) and graph coloring (Smith-Miles et al., 2014). However, it has not been yet adopted and applied to continuous black-box optimization.

To construct the feature space, $\mathbf{\Lambda}$, we use Exploratory Landscape Analysis (ELA) measures, as evidence suggests that they can be effective predictors of algorithm performance (Mersmann et al., 2011; Bischl et al., 2012; Muñoz et al., 2012; Malan and Engelbrecht, 2014). To estimate the footprints, we select a set of features to generate a common instance space, which we validate by constructing sufficiently accurate prediction models of algorithm performance across the instance space. Finally, we propose a new method to construct and characterize the footprints, which we apply for each algorithm and the overall portfolio of state-of-the-art algorithms. The results reveal unique strengths and weaknesses, and demonstrate our methodology's ability to quantify common features of hard problems, and locate regions where a phase transition may lie.

The remainder of this article is as follows: In Section 2, we introduce the metadata employed to generate a reliable representation of the problem space, and to measure the power of a continuous black-box optimization algorithm. We then train and validate prediction models in the feature space in Section 3, and we construct the instance space used to visualize the algorithm footprints in Section 4. In Section 5, we measure the relative power of each algorithm to gain insights into the conditions under which each algorithm has demonstrable strengths and weaknesses. Finally, we discuss the main implications of the results and outline avenues for further research in Section 6.

2 Experimental Metadata

Before describing the experimental metadata, let us define our notation. Without losing generality over maximization, we assume minimization throughout this article. The goal in a black-box optimization problem is to minimize a cost function $f : \mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{X} \subset \mathbb{R}^D$ is the input space, $\mathcal{Y} \subset \mathbb{R}$ is the output space, and $D \in \mathbb{N}^*$ is the dimensionality of the problem. A candidate solution $\mathbf{x} \in \mathcal{X}$ is a D-dimensional vector, and $y \in \mathcal{Y}$ is the candidate's cost. A target cost value, $y_t \in \mathcal{Y}$, defines the upper bound of a satisfactory minimization performance from an algorithm.

2.1 Black-Box Continuous Optimization Instances

As a representative subset of the problem space, $\mathbf{F}$, we use the noiseless benchmark set from the Comparing Continuous Optimizers (COCO) package (Hansen et al., 2014). Problem instances are generated by scaling and transforming 24 basis functions, which

are classified into five categories: Separable ($f_1 - f_5$), low or moderately conditioned ($f_6 - f_9$), unimodal with high conditioning ($f_{10} - f_{14}$), multimodal with adequate global structure ($f_{15} - f_{29}$), and multimodal with weak global structure ($f_{20} - f_{24}$). Transformations include linear translations and rotations, and symmetry breaking through oscillations around the identity. Each instance is uniquely identified by an index. For each basis function, we generate 15 transformed instances for each dimension $D = \{2, 5, 10, 20\}$, resulting in a total of 1440 problem instances. A detailed qualitative description of each basis function is proposed by Mersmann et al. (2015).

2.2 Exploratory Landscape Analysis Measures

Exploratory landscape analysis (ELA) methods produce one or more measures related to the features of a problem instance (Mersmann et al., 2011). There are several methods developed for continuous optimization problems, most of which have been adapted from combinatorial optimization (Pitzer and Affenzeller, 2012; Malan and Engelbrecht, 2013). For this work, we implemented the methods summarized in Table 1, as they are quick and simple to calculate. Furthermore, these methods can share a sample obtained through a Latin hypercube design (LHD), guaranteeing that the differences observed between measures depend only on the instance, and not on sample size or sampling method. For example, the convexity and local search methods described by Mersmann et al. (2011) use independent sampling approaches; hence, they cannot share a sample between them nor with the methods in Table 1. The convexity method takes two candidates, $\{\mathbf{x}_i, \mathbf{x}_j\}$, and forms a linear combination with random weights, $\mathbf{x}_k$. Then, the difference between y_k and the convex combination of y_i and y_j is computed. The result is the number of iterations out of 1000 in which the difference is less than a threshold. Meanwhile, the local search method uses the Nelder–Mead algorithm, starting from 50 random points. The solutions are hierarchically clustered in order to identify the local optima of the function. The basin of attraction of a local optima, $\mathbf{x}_l$, that is, the subset from $\mathcal{X}$ from which a local search converges to $\mathbf{x}_l$, is approximated by the number of algorithm runs that terminate at each $\mathbf{x}_l$. Both sampling approaches do not guarantee the resulting sample is unbiased; hence, reusable. Besides ensuring a fair comparison, sharing a LHD sample reduces the overall computational cost, as no new candidates must be taken from the space.

We generate an input sample, $\mathbf{X}$, for each considered dimension of each function using LHD. The size of the sample is $1000 \times D$. The output sample, $\mathbf{Y}$, is generated by evaluating $\mathbf{X}$ on each instance from the COCO benchmark of a given dimension. As D increases, the size of $\mathbf{X}$ is relatively smaller, affecting the ability of the features to accurately summarize the properties of the instance. Since accurate features can only be calculated by increasing the sample size exponentially (Jansen, 1999), a trade-off in accuracy is made to calculate the features in polynomial time (He et al., 2007). This is an unavoidable limitation due to the curse of dimensionality. Furthermore, each feature was normalized employing the techniques described in Table 1.

2.3 Algorithms

We used the algorithms listed in Table 2 as a representative subset of the algorithm space, $\mathbf{A} \subset \mathcal{A}$. The algorithms were automatically selected using ICARUS (Muñoz and Kirley, 2016), a method that uses uncovered sets to identify complementary algorithms. Also known as Landau or Fishburn sets, uncovered sets are used in voting systems theory to identify the winners of an election (Penn, 2006). The selected algorithms were tested using COCO at the Black-Box Optimization Benchmarking (BBOB) sessions at

Table 1: Summary of the exploratory landscape analysis features employed in this article. Each was normalized using the transformations listed.

Method	Feature	Description	Transformations	Reference
Correlation	FDC	Fitness distance correlation	$\tanh^{-1}$	(Jones and Forrest, 1995)
Average distances	$DISP_{1\%}$	Dispersion of sample points within 1% of $\mathbf{y}_t$	Unit scaling	(Lunacek and Whitley, 2006)
Surrogate models	$\bar{R}^2_L$	Adjusted coefficient of determination of a linear regression model	Unit scaling	(Mersmann et al., 2011)
	$\bar{R}^2_{LI}$	Adjusted coefficient of determination of a linear regression model including variable interactions	Unit scaling	
	$\bar{R}^2_Q$	Adjusted coefficient of determination of a purely quadratic regression model	Unit scaling	
	$\bar{R}^2_{QI}$	Adjusted coefficient of determination of a quadratic regression model including variable interactions	Unit scaling	
	$\min(\beta_L)$	Minimum of the absolute value of the linear model coefficients	$\log_{10}$, z-score	
	$\max(\beta_L)$	Maximum of the absolute value of the linear model coefficients	$\log_{10}$, z-score	
	CN	Ratio between the minimum and the maximum absolute values of the quadratic term coefficients in the purely quadratic model	Unit scaling	
Entropic Significance	$\xi^{(D)}$	Significance of D-th order	z-score, tanh	(Seo and Moon, 2007)
	$\xi^{(1)}$	Significance of first order	z-score, tanh	
	$\xi^{(2)}$	Significance of second order	z-score, tanh	
Cost distribution	$\gamma(\mathbf{Y})$	Skewness of the cost distribution	z-score, tanh	(Mersmann et al., 2011; Marin, 2012)
	$\kappa(\mathbf{Y})$	Kurtosis of the cost distribution	$\log_{10}$, z-score	
	$H(\mathbf{Y})$	Entropy of the cost distribution	$\log_{10}$, z-score	
Fitness sequences	$H_{\max}$	Maximum information content	z-score	(Muñoz, Kirley, and Halgamuge, 2015)
	ϵ_S	Settling sensitivity	$\log_{10}$, z-score	
	M_0	Initial partial information	z-score	

Table 2: Summary of the algorithms employed in this article, which were selected using ICARUS (Muñoz and Kirley, 2016) and the publicly available results from the BBOB sessions at the 2009 and 2010 GECCO Conferences. Algorithm names are as used for the dataset descriptions available at http://coco.gforge.inria.fr/doku.php?id=algorithms .

Algorithm	Description	Reference
BFGS	The MATLAB implementation (`fminunc`) of this quasi-Newton method, which is randomly restarted whenever a numerical error occurs. The Hessian matrix is iteratively approximated using forward finite differences, with a step size equal to the square root of the machine precision. Other than the default parameters, the function and step tolerances were set to 10^{-11} and 0 respectively.	(Ros, 2009a)
BIPOP-CMA-ES	A multistart CMA-ES variant with equal budgets for two interlaced restart strategies. After completing a first run with a population of size $\lambda_{\text{def}} = 4 + \lfloor 3 + \ln D \rfloor$, the first strategy doubles the population size; while the second one keeps a small population given by $\lambda_s = \left\lfloor \lambda_{\text{def}} \left(\frac{1}{2} \frac{\lambda_\ell}{\lambda_{\text{def}}} \right)^{\mathcal{U}[0,1]^2} \right\rfloor$, where λ_ℓ is the latest population size from the first strategy, λ, and $\mathcal{U}[0, 1]$ is an independent uniformly distributed random number. Therefore, $\lambda_s \in [\lambda_{\text{def}}, \lambda/2]$. All other parameters are at default values.	(Hansen, 2009)
LSstep	An axis parallel line search method effective only on separable functions. To find a new solution, it optimizes over each variable independently, keeping every other variable fixed. The STEP version of this method uses interval division, i.e., it starts from an interval corresponding to the upper and lower bounds of a variable, which is divided by half at each iteration. The next sampled interval is based on its "difficulty," i.e., by its belief of how hard it would be to improve the best-so-far solution by sampling from the respective interval. The measure of difficulty is the coefficient a from a quadratic function $f(x) = ax^2 + bx + c$, which must go through the both interval boundary points.	(Pošík and Huyer, 2012)
Nelder–Doerr	A version of the Nelder–Mead algorithm, that uses random restarts, resizing and half-runs. In a resizing step, the current simplex is replaced by a "fat" simplex, which maintains the best vertex, but relocates the remaining ones such that they have the same average distance to the center of the simplex. Such steps are performed every 1000 algorithm iterations. In a half-run, the algorithm is stopped after $30 \times D$ interations, with only the most promising half-runs being allowed to continue.	(Doerr et al., 2009)
$(1 + 1)$CMA-ES	A simple CMA-ES variant, with one parent and one offspring, elitist selection and random restarts. All other parameters are set to the defaults.	(Auger and Hansen, 2009)

the 2009 and 2010 Genetic and Evolutionary Computation Conferences (GECCO). The results are publicly available at the COCO website.[2] The results are stored as raw data files which contain the numerical output of an algorithm run on a given problem. A thorough description of the files' contents is made by Hansen et al. (2014). From the files, we collected the number of function evaluations required to reach the provided y_t within 10^{-8} in at least one of 15 runs, as defined by the BBOB sessions' rules (Hansen et al., 2014). Each algorithm run had a budget of $10^4 \times D$ function evaluations. On the 2009 session, three runs were carried out on each of the first five instances, whereas on the 2010 session, one run was carried out on each of the first fifteen instances. Despite this difference, the experiments are considered equivalent to averaging across random restarts. In both scenarios, the computation of the average is somewhat dominated by the most difficult starting points or instances, which reduces the number of effective runs and increases the purely random variation in the results. Making several repetitions over all instances is a valid method to counteract this variation.

2.4 Algorithm Performance Score

Although there are several performance measures reported in the literature (Bartz-Beielstein, 2006; Malan and Engelbrecht, 2014), we use the expected running time, $\hat{t}$, as it is simple, interpretable, relevant to practice, and the measure of choice on most benchmark comparisons (Suganthan et al., 2005; Hansen et al., 2014). It is defined as the number of function evaluations required to reach y_t within a target precision for the first time over a number of runs. The result is normalized over the dimension, log-transformed, and compared to a threshold. The resulting binary performance measure, τ, describes whether an algorithm solves or does not solve the instance. An instance is solvable and labeled with $\tau = 0$ if the algorithm reaches y_t in at least one run within the budget. Otherwise it is unsolvable and labeled with $\tau = 1$. An algorithm is the best performing on an instance if it minimizes $\hat{t}$. Furthermore, we define an overall performance measure for the portfolio, and we identify the best algorithm for each instance. An instance is β-easy for the portfolio and labeled '0' if at least $100 \times \beta$% of the algorithms are able to solve it. Otherwise the instance is β-hard and labeled '1.' For this article, we define $\beta = 0.5$.

2.5 Summary of Experimental Metadata

To summarize, we collected the following metadata:

1. The problem subset **F** is composed of 1440 black-box continuous optimization instances from the COCO benchmark set, with $\{2, 5, 10, 20\}$ dimensions, as described in Section 2.1.
2. The features space, $\mathbf{\Lambda}$, is defined as the 18 ELA measures listed in Table 1.
3. An algorithm subset **A** comprises the five methods as described in Table 2.
4. The binary performance score τ, which labels an instance as solvable (0) or unsolvable (1) by an algorithm, using the performance criteria described in Section 2.4 as $\tau = 0$ if $\hat{t} < 10^4 \times D$, or 1 otherwise.

[2] http://coco.gforge.inria.fr/doku.php

3 Feature Selection and Performance Prediction

The ELA methods listed in Section 2.2 transform the sample data from the input and output space, $(\mathbf{X}, \mathbf{Y})$, to a point in the feature space, $\mathbf{\Lambda}$. While there is some good evidence to support the relationship between the selected features and problem difficulty (Bischl et al., 2012; Muñoz et al., 2012; Malan and Engelbrecht, 2014), it is likely that the number of candidate features may be infinite. Nevertheless, not all of them may be necessary to obtain an accurate performance prediction, nor to generate a viable instance space. For example, some features may be colinear, providing marginal additional information. Others may measure characteristics unrelated to the difficulties for a specific algorithm. Therefore, we use a feature selection procedure to identify the subset of features that allow us to make the most accurate predictions.

The first step in this procedure is to test for colinearities using Pearson correlation. A feature is discarded if its correlation coefficient, $r_{X,Y}$, is greater than 0.7 and its correlated feature is computationally simpler. We also discard those features highly correlated with the dimensionality of the problem, D. Therefore, these nine features were discarded: FDC is colinear with $\bar{R}^2_Q$ ($r_{X,Y} = 0.714$), $DISP_{1\%}$ is colinear with D ($r_{X,Y} = 0.752$), $\bar{R}^2_L$ is colinear with $\bar{R}^2_{LI}$ ($r_{X,Y} = 0.860$), $\bar{R}^2_{QI}$ is colinear with $\bar{R}^2_Q$ ($r_{X,Y} = 0.871$), $\min(\beta_L)$ is colinear with $H(\mathbf{Y})$ ($r_{X,Y} = 0.865$), $\max(\beta_L)$ is colinear with $H(\mathbf{Y})$ ($r_{X,Y} = 0.915$), $\xi^{(2)}$ is colinear with $\xi^{(D)}$ ($r_{X,Y} = 0.860$), ϵ_S is colinear with $H(\mathbf{Y})$ ($r_{X,Y} = 0.884$), and M_0 is colinear with $H_{\max}$ ($r_{X,Y} = 0.809$).

The second step is to measure the classification test error of support vector machine (SVM) models using as inputs a subset of at least three features. We fit three types of models depending on their output variable. The first type uses the binary performance measure, τ, as the output variable. Therefore, this model type predicts whether an instance is solvable or not for a given algorithm. The second type uses as the output variable an index representing the best algorithm for a given instance. Finally, the third type predicts whether an instance is β-easy or β-hard for the portfolio. One model of the first type for each algorithm in Table 2, and one model of the third type were fitted. On the other hand, the second type requires a multiclass model which is implemented using binary classifiers in a ranking by pairwise comparisons (Hüllermeier et al., 2008) architecture. We used LIBSVM as implementation of the SVM classifiers (Chang and Lin, 2011) and the procedure described below to fine-tune them and select the best feature subset.

Since the instances generated in the COCO benchmark are based on transformations of 24 basis functions, the instances may share similarities between each other, potentially affecting accuracy of the error estimation. Furthermore, the SVM classification model has two parameters, C and γ, which must be finely tuned. Therefore, we implement a combination of the hold-one-problem-out (HOPO) and hold-one-instance-out (HOIO) approaches proposed by Bischl et al. (2012) instead of a simple k-fold cross-validation to estimate the error and fine tune the SVMs. For all dimensions under analysis, instances 1 to 10 from each basis function correspond to the training set and instances 11 to 15 correspond to the test set. The training set is separated into six randomly partitioned groups of four basis functions as follows: $\{1, 7, 13, 19\}$, $\{2, 8, 14, 20\}$, $\{3, 9, 15, 21\}$, $\{4, 10, 16, 22\}$, $\{5, 11, 17, 23\}$, $\{6, 12, 18, 24\}$. A tuning grid for C and γ is constructed using a LHD of 30 points between -5 and 15, which then are used as powers of two. For each point in the parameter grid, a SVM model is cross-validated across function groups. To avoid over-fitting due to differences in the size of a class, a weight, ω_i, is given to each class equal to the percentage of instances not belonging to that class. The

Table 3: Selected features by the two-stage procedure described in Section 3 for predicting solvability.

	$\bar{R}^2_{LI}$	$\bar{R}^2_{Q}$	CN	$\xi^{(D)}$	$\xi^{(1)}$	$H_{\max}$	$\gamma(\mathbf{Y})$	$\kappa(\mathbf{Y})$	$H(\mathbf{Y})$
BFGS	√			√	√	√	√	√	√
BIPOP-CMA-ES	√			√					√
LSstep		√		√	√		√	√	
Nelder–Doerr		√		√			√		
(1 + 1)CMA-ES		√	√	√	√				√
Portfolio	√	√		√	√	√			√
β-Hard		√		√			√		

Table 4: Performance of the algorithm performance model on the test set, where $\%S$ is the percentage of solvable instances and $\%\hat{S}$ is the model forecast. The remaining columns measure the performance of the model. In boldface are values above 90% for ACC, SPC, TPR, NPV, and PPV, representing good model performance, and values above 10% of FNR and FPR, representing poor model performance.

Algorithm	$\%S$	$\%\hat{S}$	ACC	SPC	FNR	FPR	TPR	NPV	PPV
BFGS	43.8%	43.8%	**98.8%**	**98.6%**	1.1%	1.4%	**98.9%**	**98.6%**	**98.9%**
BIPOP-CMA-ES	95.8%	95.6%	**99.4%**	**99.6%**	5.0%	0.4%	**95.0%**	**99.8%**	**90.5%**
LSstep	30.2%	31.0%	**96.3%**	**95.2%**	3.3%	4.8%	**96.7%**	**92.6%**	**97.9%**
Nelder–Doerr	59.4%	58.3%	**93.1%**	**93.3%**	7.2%	6.7%	**92.8%**	**95.0%**	**90.5%**
(1 + 1)CMA-ES	65.6%	62.5%	**90.2%**	**90.2%**	9.7%	9.8%	**90.3%**	**94.7%**	82.8%

procedure is repeated using all groups and all points in the tuning grid for a given subset of at least three features. Therefore, one function from each class defined by Hansen et al. (2014) has been removed from the training set. The mean cross-validation error across all groups is used to determine the best parameters for the SVM for a given subset of features. Then, a SVM is fitted using the complete training set and the best parameters. The classification test error is then used to select the best subset of features.

The results of this step are presented in Table 3. Using the nine remaining features, 466 feature combinations of minimum three and maximum of nine features are possible. We used exhaustive enumeration to select the features marked in Table 3 that provided the best model. The table shows that there are different selected features for each model.

Table 4 shows the test performance of the first type of model, which forecasts whether an instance is solvable or not for a given algorithm. In the table, $\%S$ is the percentage of solvable instances and $\%\hat{S}$ is the model forecast. Performance is quantified by the model's accuracy (ACC), specificity (SPC), false negative rate (FNR), false positive rate (FPR), true positive rate (TPR), negative predictive value (NPV), and positive predictive value (PPV). In boldface are values above 90% for ACC, SPC, TPR, NPV, and PPV, representing good model performance, and values above 10% of FNR and FPR, representing poor model performance. The results show that the prediction models are effective in discriminating solvable and unsolvable instances, as they have similar

Table 5: Performance of the algorithm selection model on the test set, where $\%B$ is the percentage of instances for which an algorithm is best and $\%\hat{B}$ is the model forecast. The remaining columns represent the confusion matrix of the model. In boldface are those values in the diagonal above 90%, which represent excellent model performance, and off the diagonal above 10%, which represent poor model performance.

Algorithm	$\%B$	$\%\hat{B}$	BFGS	BIPOP-CMA-ES	LSstep	Nelder–Doerr	$(1+1)$CMA-ES
BFGS	15.6%	15.8%	**100.0%**	0.0%	2.2%	0.0%	0.0%
BIPOP-CMA-ES	42.7%	42.3%	0.0%	**99.0%**	0.0%	0.0%	0.0%
LSstep	9.4%	9.4%	0.0%	0.0%	**97.8%**	0.8%	0.0%
Nelder–Doerr	25.0%	25.2%	0.0%	1.0%	0.0%	**99.2%**	0.0%
$(1+1)$CMA-ES	7.3%	7.3%	0.0%	0.0%	0.0%	0.0%	**100.0%**

Table 6: Performance of the portfolio hardness model with $\beta = 0.5$, where $\%C$ is the percentage of instances on each class and $\%\hat{C}$ is the model forecast. The remaining columns contain the confusion matrix for the prediction models. In boldface are those values in the diagonal above 90%, representing good model performance, and off the diagonal above 10%, representing poor model performance.

	$\%C$	$\%\hat{C}$	β-easy	β-hard
β-easy	59.4%	58.3%	**93.3%**	7.2%
β-hard	40.6%	41.7%	6.7%	**92.8%**

values of SPC and TPR. The least reliable of the models was fitted for $(1+1)$CMA-ES, which has a tendency to misclassify unsolvable problems (NPV > PPV, PPV = 82.8%). A similar problem is present in the BIPOP-CMA-ES model (FNR > FPR), due to the severe class imbalance ($\%S = 95.8\%$).

Table 5 shows the test performance of the second type of model, which forecasts the best algorithm for a given instance. In the table, $\%B$ is the percentage of instances for which an algorithm is best and $\%\hat{B}$ is the model forecast. The remaining columns indicate the best algorithm, while the rows indicate the model selection. In other words, these columns represent the confusion matrix of the model. In boldface are those values in the diagonal above 90%, which represent excellent model performance, and off the diagonal above 10%, which represent poor model performance. The results show that LSstep and $(1+1)$CMA-ES are specialized algorithms, as each one of them is the best solver for less than 10% of the instances. As observed in Table 5, the model is highly accurate with on-diagonal performance above 97.0%. Errors are occasionally made by selecting Nelder–Doerr over BIPOP-CMA-ES (1.0%), BFGS over LSstep (2.2%) and LSstep over Nelder–Doerr (0.8%). With these choices though, all the instances would still be solved by the recommended algorithm.

Table 6 shows the test performance of the third type of model, which forecasts whether an instance is β-easy or β-hard for the portfolio, with $\beta = 0.5$. In the table, $\%C$ is the percentage of instances in each class and $\%\hat{C}$ is the model forecast. The remaining columns contain the confusion matrix. In boldface are those values in the diagonal

Table 7: Selected features by the two-stage procedure described in Section 3. The selected features for the common instance space were those that produced a stable two-dimensional representation while minimizing the average test error.

	$\bar{R}^2_{LI}$	$\bar{R}^2_{Q}$	CN	$\xi^{(D)}$	$\xi^{(1)}$	$H_{\max}$	$\gamma(\mathbf{Y})$	$\kappa(\mathbf{Y})$	$H(\mathbf{Y})$
BFGS	√	√			√		√		√
BIPOP-CMA-ES			√	√	√			√	√
LSstep		√		√	√				
Nelder–Doerr	√	√		√					
$(1+1)$CMA-ES	√			√	√				
Portfolio		√		√	√				
β-Hard	√	√		√					
Common space	√	√		√	√				√

above 90%, representing good model performance, and off the diagonal above 10%, representing poor model performance. The results show that the prediction model retains accuracy above 90%.

To summarize, the nine selected ELA features have effectively enabled highly accurate prediction of algorithm performance, identifying whether each algorithm can solve an instance, which algorithm from the chosen portfolio solves each instance fastest, and how the entire portfolio considers the difficulty of each instance. The relationship between the ELA features and the performance of algorithms necessary to tackle the algorithm selection problem shown in Figure 1 has therefore been established.

4 Instance Space Generation through Dimension Reduction

We now shift our focus to using the ELA features to construct an instance space—enabling each instance to be represented as a point in a metric space—which will enable us to visualize the similarities and differences between test instances (functions). The instance space will also enable us to characterize and quantify the region of the space where we can expect good-performance of an algorithm. We call such a region the algorithm footprint (Corne and Reynolds, 2010). For this purpose, a dimensionality reduction method is required to project the m-dimensional feature space into $\mathbb{R}^2$ for ease of visualization. We use principal component analysis (PCA), which rotates the data to a new coordinate system in $\mathbb{R}^m$, with the axes defined by m new features which are linear combinations of the original ones. The new axes are calculated as the eigenvectors of the $m \times m$ covariance matrix. We then project the instances on the two principal eigenvectors corresponding to the two largest eigenvalues of the covariance matrix.

Just as we did for the prediction models described in Section 3, we use a selection process to identify which features generate a viable instance space, that is, the one that allows the best separation between classes of instances. The procedure is mostly the same as described in Section 3, with the difference that the subset of features is projected onto two variables (coordinates along the top two PCA axes), which are then used as inputs to the SVMs. The three types of models described here were also trained in the two-dimensional instance space. The selected features for each model are presented in Table 7. Each fitted model selected a different subset of features, which precludes

a fair comparison between algorithms, as the projected instance space is not common. Therefore, a sacrifice in individual model accuracy must be made in order to have a common space for all algorithms, facilitating a fair comparison. Such common space could be constructed by selecting the subset of features which minimizes the average error across all the models, that is, $\left\{\bar{R}^2_Q, \xi^{(D)}, \xi^{(1)}\right\}$. However, this subset resulted in too many instances being mapped to the same point in the instance space, causing numerical instability on subsequent analyses. With an increase on average error of 2.90%, the best set of features that produced a stable instance space was $\left\{\bar{R}^2_{LI}, \bar{R}^2_Q, \xi^{(D)}, \xi^{(1)}, H(\mathbf{Y})\right\}$. Although the instance space could be constructed using all the features, this would result in an increased error of 10.07%. The instance space, which explains 89.45% of the variance in the data, is described by the equation:

$$\begin{bmatrix} PC_1 \\ PC_2 \end{bmatrix} = \begin{bmatrix} 0.1499 & 0.1513 & 0.1738 & 0.1444 & 0.9506 \\ 0.1167 & 0.2722 & 0.4599 & 0.7936 & -0.2663 \end{bmatrix} \begin{bmatrix} \bar{R}^2_{LI} \\ \bar{R}^2_Q \\ \xi^{(D)} \\ \xi^{(1)} \\ H(\mathbf{Y}) \end{bmatrix}^\top \tag{1}$$

Figure 2 illustrates the common instance space, where each point represents an instance, plotted separately by the qualitative groups described in Section 2.1 and by dimensionality. In the instance space, problem dimensionality is inversely proportional to the PC_2 ($r_{X,Y} = -0.5923$). Since $\approx 60\%$ of the instances have $PC_2 < 0$, with a minimum being -1.1902, the instances are closer to each other as the dimension increases. This demonstrates that the features lose capacity to discriminate between the functions at higher dimensions as expected, given the sampling dependency on the feature calculations. Besides, conditioning decreases with the PC_1. This can be explained by the strong correlation between PC_1 and $H(\mathbf{Y})$ ($r_{X,Y} = 0.9882$), with the latter known to relate to function conditioning (Muñoz, Kirley, and Halgamuge, 2015). Furthermore, multimodal functions are located toward the lower-left side of the instance space, with f_{22} and f_{21} constructing the tail of the space. We can observe a number of gaps in the instance space, that might accommodate the generation of new test functions.

Figure 3 illustrates the value of each feature in Table 3 throughout the instance space, with each one of them linearly scaled in the $[0, 1]$ range. Moderate correlations are observed between PC_1 and $\bar{R}^2_{LI}$ ($r_{X,Y} = 0.5037$), $\bar{R}^2_Q$ ($r_{X,Y} = 0.4842$) and $\xi^{(D)}$ ($r_{X,Y} = 0.5265$). Moderate to high correlations are observed between PC_2 and $\bar{R}^2_Q$ ($r_{X,Y} = 0.4588$), $\xi^{(D)}$ ($r_{X,Y} = 0.7337$), $\xi^{(1)}$ ($r_{X,Y} = 0.9176$). These features are related with the problem separability (Seo and Moon, 2007), modality and scaling (Mersmann et al., 2011). Hence, they confirm that smoother, unimodal problems likely to be separable and ill-conditioned, will have positive PC_1 and PC_2. Unselected features have moderate to low correlation with the projected coordinates. For example, $H_{\max}$ —a feature related with modality (Muñoz, Kirley, and Halgamuge, 2015)—increases toward the lower-left side of the space ($PC_1 : r_{X,Y} = -0.3738$, $PC_2 : r_{X,Y} = -0.4048$), where multimodal functions are located. On the other hand, CN lacks of an obvious gradient on the instance space ($PC_1 : r_{X,Y} = -0.2690$, $PC_2 : r_{X,Y} = 0.1487$).

Within this 2-D instance space, it is interesting to see if the location of an instance enables algorithm performance to be predicted as well as when we used all nine features to summarize an instance. We regenerate the models from Section 3, now using only the 2-D coordinates as the input vector.

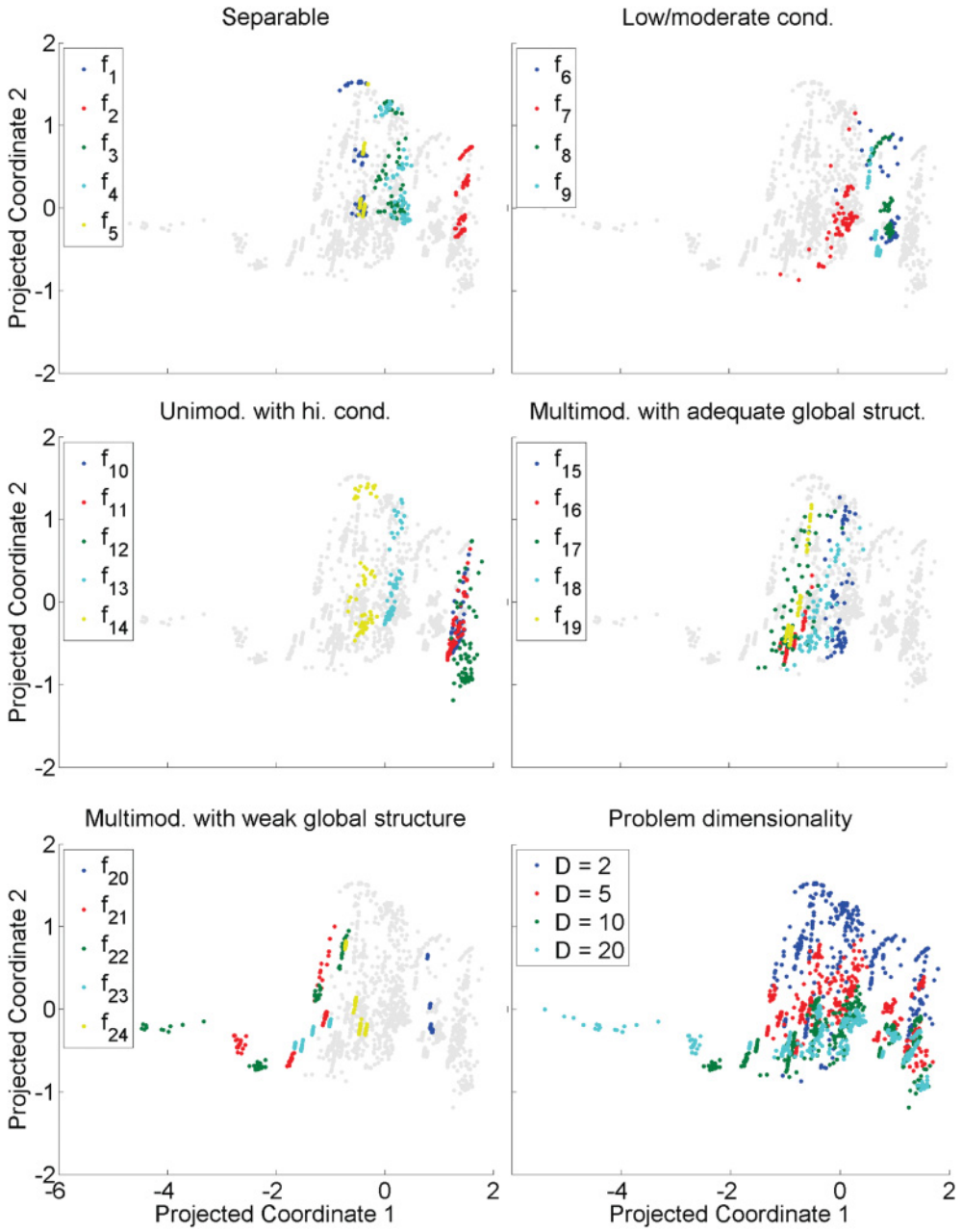

Figure 2: Functions from the COCO benchmark set and their location in the common instance space, with coordinates given by Equation (1).

Table 8 shows the test performance of the first type of model, which forecasts whether an instance is solvable or not for a given algorithm. The columns and rows follow the conventions of Table 4. The results show that the prediction models in Section 3 are more accurate than the ones built for visualization as expected, with the LSstep

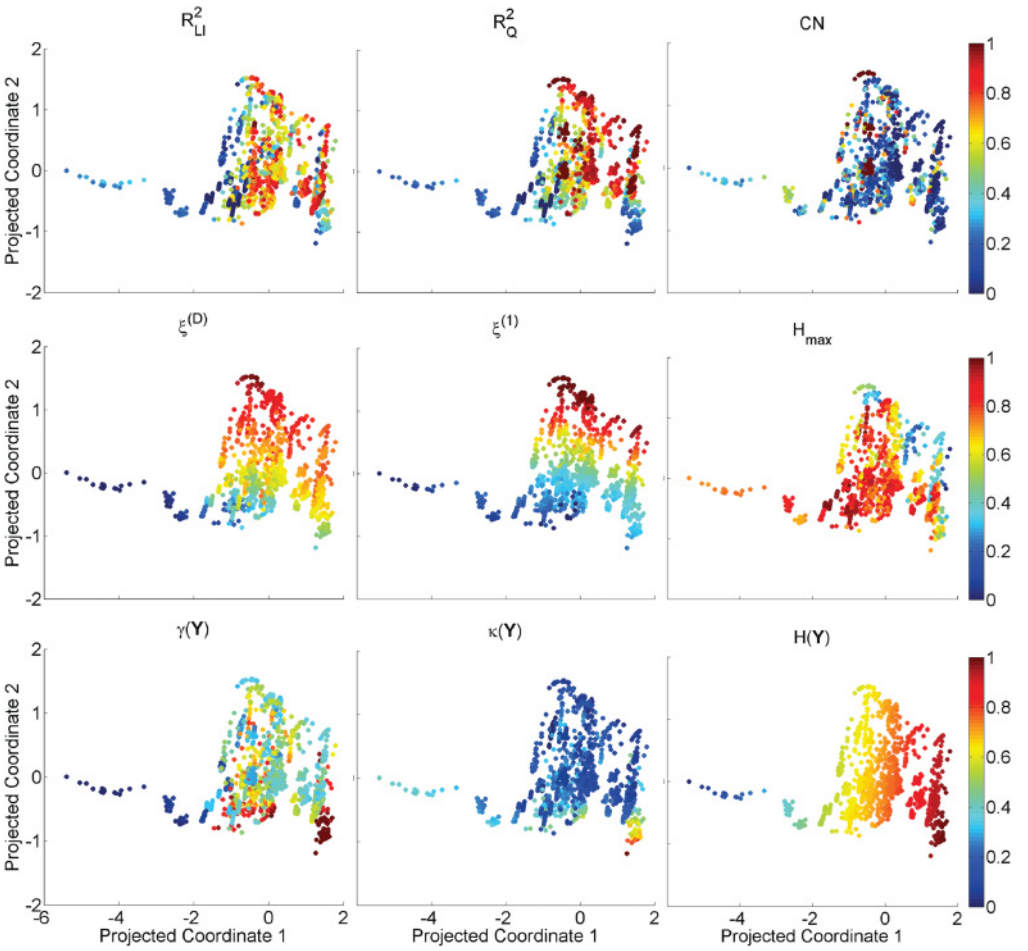

Figure 3: Feature distribution across the common instance space, with each one of them linearly scaled in [0, 1] range.

and (1 + 1)CMA-ES models falling under 90% accuracy. Furthermore, the visualization models perform less uniformly, presenting specific advantages and disadvantages. For example, the BFGS and LSstep models are more accurate at detecting unsolvable problems (TPR > SPC), although they can be deemed as conservative (FPR > FNR, BFGS:FPR = 12.4%, LSstep:FPR = 27.6%). On the contrary, the BIPOP-CMA-ES model is optimistic (FNR ≫ FPR, FNR = 55.0%), which means that the model oversimplifies the results. As it was the case for prediction, the (1 + 1)CMA-ES model is the least reliable, with a tendency to misclassify unsolvable problems (NPV > PPV, PPV = 70.3%). These results indicate that the areas of the instance space in which a problem is solvable are noncontiguous.

Table 9 shows the test performance of the second type of model, which forecasts the best algorithm for a given instance. The rows and columns follow the conventions of Table 5. The results show that the model has lost accuracy, with only BIPOP-CMA-ES retaining on-diagonal performance above 90.0%. Errors are made by selecting Nelder–Doerr over BFGS (10.7%), BIPOP-CMA-ES over LSstep (26.7%), and BFGS over

Table 8: Performance of the algorithm performance model on the test set, where $\%S$ is the percentage of solvable instances and $\%\hat{S}$ is the model forecast. The remaining columns measure the performance of the model. In boldface are values above 90% for ACC, SPC, TPR, NPV, and PPV, representing good model performance, and values above 10% of FNR and FPR, representing poor model performance.

Algorithm	$\%S$	$\%\hat{S}$	ACC	SPC	FNR	FPR	TPR	NPV	PPV
BFGS	43.8%	42.3%	**90.6%**	87.6%	7.0%	**12.4%**	**93.0%**	**90.6%**	**90.6%**
BIPOP-CMA-ES	95.8%	97.7%	**97.3%**	**99.6%**	**55.0%**	0.4%	45.0%	**97.7%**	81.8%
LSstep	30.2%	25.8%	87.7%	72.4%	5.7%	**27.6%**	**94.3%**	84.7%	88.8%
Nelder–Doerr	59.4%	62.9%	**91.9%**	**96.1%**	**14.4%**	3.9%	85.6%	**90.7%**	**93.8%**
$(1+1)$CMA-ES	65.6%	59.4%	82.1%	81.6%	**17.0%**	**18.4%**	83.0%	**90.2%**	70.3%

Table 9: Performance of the algorithm selection model on the test set, where $\%B$ is the percentage of instances for which an algorithm is best and $\%\hat{B}$ is the model forecast. The remaining columns represent the confusion matrix of the model. In boldface are those values in the diagonal above 90%, which represent excellent model performance, and off the diagonal above 10%, which represent poor model performance.

Algorithm	$\%B$	$\%\hat{B}$	BFGS	BIPOP-CMA-ES	LSstep	Nelder–Doerr	$(1+1)$CMA-ES
BFGS	15.6%	14.0%	77.3%	0.5%	0.0%	3.3%	**11.4%**
BIPOP-CMA-ES	42.7%	45.0%	5.3%	**92.2%**	**26.7%**	6.7%	8.6%
LSstep	9.4%	8.3%	0.0%	2.9%	66.7%	3.3%	0.0%
Nelder–Doerr	25.0%	25.8%	**10.7%**	3.9%	6.7%	86.7%	2.9%
$(1+1)$CMA-ES	7.3%	6.9%	6.7%	0.5%	0.0%	0.0%	77.1%

Table 10: Performance of the portfolio hardness model with $\beta = 0.5$, where $\%C$ is the percentage of instances on each class and $\%\hat{C}$ is the model forecast. The remaining columns contain the confusion matrix for the visualization models. In boldface are those values in the diagonal above 90%, representing good model performance, and off the diagonal above 10%, representing poor model performance.

	$\%C$	$\%\hat{C}$	β-easy	β-hard
β-easy	59.4%	62.9%	**96.1%**	**14.4%**
β-hard	40.6%	37.1%	3.9%	85.6%

$(1+1)$CMA-ES (11.4%). These errors would result on 2.4% of the instances to remain unsolved through an incorrect choice of algorithms.

Table 10 shows the test performance of the third type of model, which forecasts whether an instance is β-easy or β-hard for the portfolio, with $\beta = 0.5$. The rows and columns and rows follow the conventions of Table 6. The results show that only β-easy instances maintain accuracy above 95%, while 14.4% of β-hard instances are classified

as β-easy. Of these falsely easy instances, 82.8% can only be solved by a single algorithm. On the other hand, 72.7% of the falsely hard instances can be solved by three algorithms. Clearly, a desire to visualize the algorithm performance in two-dimensions has cost some accuracy. Nevertheless, since the accuracy across models is still above 80%, we conclude that the instance space representation is reliable enough, and could no doubt be improved with the addition of more features that better explain problem difficulty. Visualization in 3-D would also lessen some damage caused by dimension reduction, as the first three eigenvalues explain 96.6% of the variance, while the first two explain 89.4%.

5 Analysis of Algorithm Power

Using predictions by the models described in Section 4, we define two footprints for each algorithm. The first one, called the *solution footprint*, indicates the area of the space in which any instance is predicted to be solvable by the corresponding algorithm, even if those instances are not in the training subset. The second one, called the *absolute footprint*, indicates the area of the space in which the algorithm will be the best performing for any instance. To characterize a footprint, we measure its relative size, density (number of instances inside the footprint area), and purity (the percentage of solvable or best performing instances within the footprint), which provide us with an objective measurement of the relative strength of the algorithm across the instance space. Furthermore, these measurements give us information on where within the instance space the algorithm is strong. The accuracy of the footprints is tightly linked to the predictive models fitted in Section 4; therefore, the results in Tables 8 to 10 serve as a measure of confidence for each footprint (Smith-Miles et al., 2014), and should be reported simultaneously.

Methods for calculation and analysis of algorithm footprints have been reported in previous work. For example, Smith-Miles and Tan (2012) used the area of the convex hull created by the points where good performance was observed. To find the convex hull, we calculated a Delaunay triangulation, which was then pruned by removing the triangles whose edges exceeded a threshold. This is a top-down approach for the footprint calculation. Smith-Miles et al. (2014) generated unconnected triangles by finding the nearest neighbors and maintaining a taboo list. The triangles were merged together if the resulting region fulfilled the density and purity requirements, which can be thought of as a bottom-up approach. There are limitations to both approaches. On the former, there is insufficient evidence that the remaining triangles belong to the footprint. On the latter, the random elements on the triangle construction lead to a number of them being exceedingly large.

We combine the strengths of both approaches in Algorithm 1, aiming to mitigate their limitations. In its first step, the algorithm measures the distances between two instances, and marks one of them for elimination if the distance is less than a threshold, δ. This is done to avoid numerical instability with the triangulation algorithm, which is applied as the second step. As a third step, the algorithm finds the concave hull, by removing any triangle with edges larger than another threshold, Δ. As a fourth and final step, the algorithm calculates the density and purity of each triangle in the concave hull. If a triangle does not fulfill the density and purity limits, it is removed from the hull. The parameters for Algorithm 1 are the lower and upper distance thresholds, $\{\delta, \Delta\}$, set to 1% and 25% of the maximum distance, respectively. The density threshold, ρ, is set to 10, and the purity threshold, π, is set to 75%.

Algorithm 1: Calculation of an algorithm footprint using concave hulls with minimum density and purity requirements. A triangle $\mathbf{t}$ is defined by a set of vertices $\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c\} \in \mathbf{X}_{\text{TRI}}$.

Input: A set of instances in the instance space, $\mathbf{X}$, with their performance labels, $\mathbf{y}$, and the purity, π, density, ρ, and distance, $\{\delta, \Delta\}$, thresholds.
Output: An algorithm footprint composed of a triangulation, $\mathbf{T}$, and a set of areas for each triangle $\mathbf{t} \in \mathbf{T}$, $\mathbf{a}$.

```
// Generate a candidate list for triangulation
```
$\mathbf{X}_{\text{GOOD}} = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}, y_i = \text{GOOD}\}$, $N = |\mathbf{X}_{\text{GOOD}}|$, $\mathbf{e} = \{\text{FALSE}\}^N$, $\mathbf{a} = \{\emptyset\}$;
for $i \leftarrow 1$ **to** N **do** // Remove a candidate if $\|\mathbf{x}_i - \mathbf{x}_j\| < \delta$
  for $j \leftarrow 1$ **to** N **do**
    if $\|\mathbf{x}_i - \mathbf{x}_j\| < \delta$ **then**
      if $e_i = \text{FALSE}$ **then** $e_j = \text{TRUE}$ **else** $e_j = \text{FALSE}$;
    end
  end
end
$\mathbf{X}_{\text{TRI}} \leftarrow \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}_{\text{GOOD}}, e_i = \text{FALSE}\}$;
```
// Calculate the concave hull
```
$\mathbf{T} \leftarrow$ `Delaunay` $(\mathbf{X}_{\text{TRI}})$; `// Find the Delaunay triangulation for` $\mathbf{X}_{\text{GOOD}}$
for $i \leftarrow 1$ **to** $|\mathbf{T}|$ **do** `// Remove any triangle with a side greater than` Δ
  if $\|\mathbf{x}_a - \mathbf{x}_b\| > \Delta \vee \|\mathbf{x}_b - \mathbf{x}_c\| > \Delta \vee \|\mathbf{x}_c - \mathbf{x}_a\| > \Delta$ **then** $\mathbf{T} \leftarrow \mathbf{T} - \{\mathbf{t}_i\}$;
end
for $i \leftarrow 1$ **to** $|\mathbf{T}|$ **do** `// Calculate the purity, density and area of each triangle`
  $a_i =$ `AreaOfTriangle` $(\mathbf{t}_i)$;
  $d_i =$ `InstancesInTriangle` $(\mathbf{t}_i, \mathbf{X})\,/a_i$;
  $p_i =$ `InstancesInTriangle` $(\mathbf{t}_i, \mathbf{X}_{\text{GOOD}})/$`InstancesInTriangle` $(\mathbf{t}_i, \mathbf{X})$;
  `// Remove the triangle if it does not meet the density and purity requirements`
  if $d_i < \rho \vee p_i < \pi$ **then** $\mathbf{T} \leftarrow \mathbf{T} - \{\mathbf{t}_i\}$ **else** $\mathbf{a} \leftarrow \mathbf{a} + a_i$;
end

When comparing two algorithms to decide which one is the best for a given area of the instance space, contradictions may appear between absolute footprints calculated with Algorithm 1. To eliminate such contradictions and increase the footprint's purity, we estimate the area lost by the "base" footprint if the contradicting sections are removed. The method is described in Algorithm 2. Assuming that absolute footprints with larger areas imply a higher algorithmic power, we remove the contradicting triangles in the base footprint if their area is less than the total area of the contradicting triangles in the "test" footprint. The density and purity of the resulting footprint are measured to guarantee that they meet the limits. This algorithm may also be used to clarify the edge between the solution footprint and the remaining instance space. In this case, we treat the unsolved instances as belonging to a second algorithm.

Figure 4 illustrates the solution footprints, that is, the performance is defined as the algorithm's ability to solve a problem within a budget, as predicted by the models validated in Section 4. Blue marks represent the solved problems and red marks the unsolved ones. The *known region* of the instance space, that is, the concave hull defined by all the instances, has a total area of 6.0835 units, with a density of 233.7 instances/unit. If we focus on the BIPOP-CMA-ES footprint, illustrated in Figure 4b, we observe that its unsolved region closely matches parts of LSstep's solution region, illustrated in Figure 4c. Comparing this results with Figure 2, we observe that those regions correspond to $\{f_3, f_4\}$. This implies a certain level of complementariness between these two

Algorithm 2: Footprint contradiction detection using polygon intersection. The algorithm tests whether a base and a test footprint contradict each other. Then, it removes the contradicting sections on the base footprint depending on the size of the area of contradiction, that is, the area lost by the test footprint when the contradicting triangles are removed.

Input: A base and test footprints, $\{\mathbf{T}_{\text{base}}, \mathbf{a}_{\text{base}}\}$ and $\{\mathbf{T}_{\text{test}}, \mathbf{a}_{\text{test}}\}$.
Output: A recalculated base footprint.

$\boldsymbol{\alpha}_{|\mathbf{T}_{\text{base}}|\times 1} = \mathbf{0}$; `// Area of contradiction`
`// Determine if a base triangle is intercepted and measure its area of contradiction`
for $i \leftarrow 1$ **to** $|\mathbf{T}_{base}|$ **do**
  for $j \leftarrow 1$ **to** $|\mathbf{T}_{test}|$ **do**
    if `PolygonIntersection` $(\mathbf{t}_{\text{base},i}, \mathbf{t}_{\text{test},j})$ **is** TRUE **then** $\alpha_i = \alpha_i + a_{\text{test},j}$;
  end
end
`// Remove a base triangle if its area of contradiction is larger than its area`
for $i \leftarrow 1$ **to** $|\mathbf{T}_{base}|$ **do**
  if $\alpha_i > a_{base,i}$ **then** $\mathbf{T}_{\text{base}} \leftarrow \mathbf{T}_{\text{base}} - \{\mathbf{t}_{\text{base},i}\}$, $\mathbf{a}_{\text{base}} \leftarrow \mathbf{a}_{\text{base}} - \{a_{\text{base},i}\}$;
end

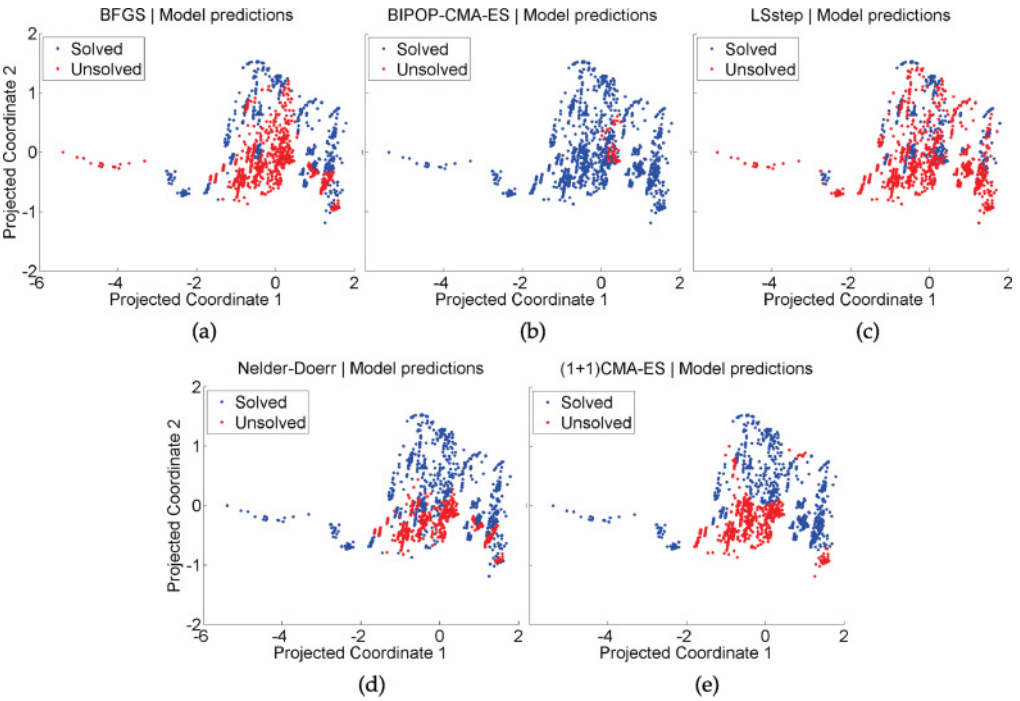

Figure 4: Solution footprints of (a) BFGS, (b) BIPOP-CMA-ES, (c) LSstep, (d) Nelder–Doerr, and (e) $(1 + 1)$CMA-ES, as predicted by the models validated in Section 4.

algorithms. We also can notice that BIPOP-CMA-ES can solve problems roughly located at $PC_1 \in [-2, 0]$, $PC_2 \in [-1, 0]$. According to Figure 2, this region mostly corresponds to multimodal problems with adequate global structure, which no other algorithm in

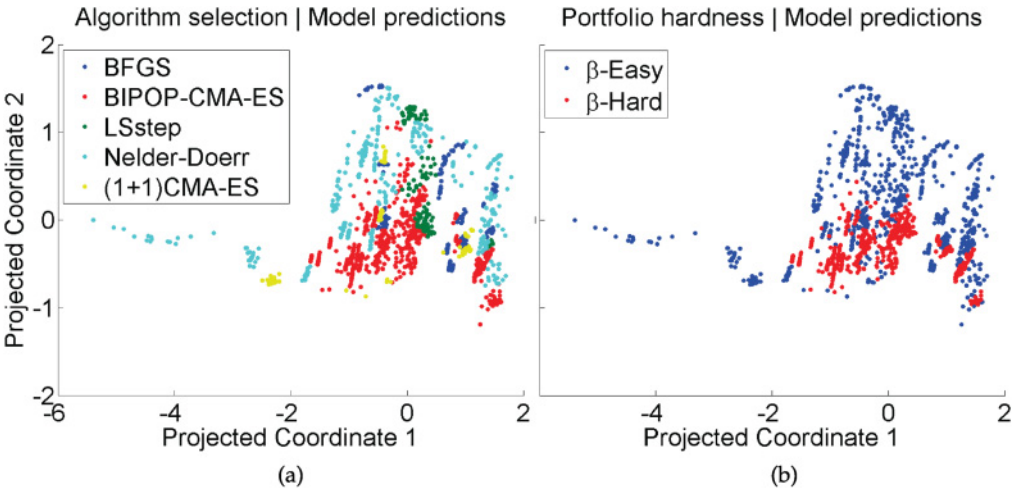

Figure 5: Portfolio footprints as predicted by the models validated in Section 4, where (a) shows the absolute footprints for each algorithm, and (b) shows the solution footprint for the portfolio based on the β-hardness criteria.

the portfolio is able to solve. This result matches our expectations, as BIPOP-CMA-ES is designed to exploit strong global structures. The easiest problems seem to be located on the top right region of the space, as most of the algorithms are able to solve them. By comparing these results with Figure 2, we observe that the easier problems have $D = 2$, while the harder ones have $D = \{10, 20\}$. The tail-like region located at $PC_1 \in [-6, -2]$, which is mostly unsolvable to BFGS and LSstep, is composed of high dimensional, multimodal problems with weak structure.

Figure 5a illustrates the absolute footprints as predicted by the model validated in Section 4, with the color of the marks indicating the best algorithm. Figure 5b shows the predicted solution footprint for the portfolio, based on the β-hardness criteria, with the blue marks representing the β-easy problems and red marks the β-hard ones. Figure 5 confirms the observations made in Figure 4. For example, the β-hard regions for the algorithm are those covered by BIPOP-CMA-ES and, to a lesser extent, LSstep. On the other hand, the Nelder–Doerr algorithm covers most of the β-easy regions.

To quantify these observations, we calculate the area, a, density, d, and purity, p, of the algorithm footprints. Table 11 shows the results based on the predictions of the models validated in Section 4 and their difference with the results based on experimental data. The normalized area and density, denoted with the subscript N, were calculated using the values from the known region of the instance space as references, that is, $a = 6.0835$ and $d = 233.7463$. In boldface are the differences that exceed 10%. Table 11 also shows the accuracy of the models in which the footprints are based, extracted from Tables 8 to 10.

As observed in Figure 4, BIPOP-CMA-ES has the largest solution footprint (ACC $=$ 90.6%, $a_N = 97.4\%$, $d_N = 98.3\%$, $p = 99.4\%$) followed by Nelder–Doerr and $(1 + 1)$CMA-ES, with densities between 80% and 90% and purities above 99%. Larger footprints imply higher algorithmic power, while higher density provides larger sample sizes to support the conclusion of good performance. On the other hand, LSstep has the smallest and most dense solution footprint (ACC $= 87.7\%$, $a_N = 3.5\%$, $d_N = 511.6\%$, $p = 99.6\%$).

Table 11: Analysis of algorithm power through the predicted accuracy, ACC, area, a, density, d, and purity, p, of the solution and absolute footprints as predicted by the models validated in Section 4, and their difference with the results based on experimental data. The area and density are normalized using the values from the known region of the instance space as references, i.e., $a = 6.0835$ and $d = 233.7463$. In boldface are the differences that exceed 10%.

		Solution footprint				Absolute footprint			
	Algorithm	ACC	a_N	d_N	p	ACC	a_N	d_N	p
Predicted	BFGS	90.6%	44.1%	91.4%	99.5%	77.3%	2.7%	514.7%	99.0%
	BIPOP-CMA-ES	97.3%	97.4%	98.3%	99.4%	92.2%	23.8%	176.0%	98.7%
	LSstep	87.7%	3.5%	511.6%	99.6%	66.7%	3.5%	241.2%	100.0%
	Nelder–Doerr	91.9%	62.8%	85.8%	99.3%	86.7%	36.7%	67.8%	97.2%
	$(1+1)$CMA-ES	82.1%	60.1%	97.6%	99.8%	77.1%	0.8%	787.4%	100.0%
	β-easy					96.1%	62.8%	85.8%	99.3%
	β-hard					85.6%	22.5%	177.9%	94.6%
Difference	BFGS		3.3%	1.0%	0.6%		0.5%	**109.6%**	0.6%
	BIPOP-CMA-ES		0.9%	0.8%	0.1%		4.8%	**16.3%**	1.5%
	LSstep		7.1%	**321.5%**	1.3%		0.5%	**47.5%**	8.1%
	Nelder–Doerr		2.4%	0.6%	0.2%		5.1%	5.6%	6.9%
	$(1+1)$CMA-ES		3.2%	0.6%	0.4%		0.3%	**244.6%**	2.4%
	β-easy						2.4%	0.6%	0.2%
	β-hard						1.7%	5.1%	0.1%

A small footprint implies specialization, while high density indicates strong similarities between the covered instances. However, smaller footprints are harder to predict accurately, due to the minority class distribution.

If we focus on the absolute footprints, Nelder–Doerr and BIPOP-CMA-ES cover the largest area, with 36.7% and 23.8%, respectively, of the valid instance space covered. Nelder–Doerr has also the least dense footprint (ACC = 86.7%, $d_N = 67.8\%$). As observed in Figures 2 and 4d, this algorithm is best for lower dimensional problems, which happen to cover a larger area. The remaining algorithms have footprints that cover less than 10% of the instance space. In particular $(1+1)$CMA-ES is the weakest one (ACC = 77.1%, $a_N = 0.8\%$, $d_N = 787.4\%$, $p = 100.0\%$), implying that it has strong performance throughout the instance space, and is excellent for some problems located at $PC_2 \in [-1, -0]$, most of which are multimodal with adequate global structure and $D = 10$, according to Figure 2. In overall, its performance is unremarkable compared with others but sufficient to make it a good off-the-shelf algorithm. In total, the absolute footprints cover 67.5% of the space, while the remaining 32.5% is disputed as no decision can be made due to contradictions and lack of information.

To round off the results, Table 11 also shows the area, density, and purity of the β-easy and hard regions of the space. With ACC = 96.1%, $a_N = 62.8\%$ and $d_N = 85.8\%$, the β-easy region is the largest but also the least dense, which is explained by its coverage of most low-dimensional problems according to Figure 2. Although its area is close to that of the combined absolute footprints, it is worth noting that the β-easy region

corresponds to the intersection of at least two solution footprints. Hence, these two areas are not expected to be equal. On the other hand, the β-hard region covers 22.5% of the space with $d_N = 177.9\%$, covering most of the higher dimensional problems. The remaining 15.4% is disputed.

6 Conclusion

At the beginning of this article, we aimed to provide a method for the objective assessment of the strengths and weaknesses of black-box continuous optimization algorithms, considering both the nature of the test instances and a measure of performance. This method is based on the accurate estimation and characterization of the algorithm footprints, that is, the regions in the instance space in which the algorithm is the best or it is expected to perform well. To estimate the footprints, we selected a set of features to generate a common instance space. We validated the features by demonstrating their ability to predict algorithm performance in Section 3. An instance space was generated using the selected features in Section 4, enabling the strengths and weaknesses of each algorithm to be visualized. Finally, we characterized the solution and absolute footprints for each algorithm, and the β-easy regions for the overall portfolio in Section 5, with β-easy understood as an instance for which at least $100 \times \beta\%$ of the algorithms are able to provide a solution. We now discuss the implications of this work.

The strong links between some features resulted in nine of them being discarded due to high correlation. Another four were removed as they increased the error of the predictive models. All the retained features are invariant to translations on the output space, $\mathcal{Y}$, but none is invariant to translations on the input space, $\mathcal{X}$, for all instances and functions (Muñoz and Smith-Miles, 2015). This explains why two other features were critical to generate a stable instance space, while producing models with accuracy above 80%. Although it has been shown that both single features and large feature sets can fail as performance predictors (Malan and Engelbrecht, 2014), it is also true that problems with similar features may result in different performance if we don't measure sufficient features that affect performance. By keeping these two features, we also retained some information about the phase transitions, that is, the points in which a small change in problem structure produces algorithm failure. There is no doubt that new and adequate features could sharpen the phase transitions; however, we should aim to exploit the data already collected as much as possible so we do not incur additional costs (Muñoz, Sun, Kirley, and Halgamuge, 2015).

The footprint analysis revealed a complementary relationship between BIPOP-CMA-ES, an excellent off-the-shelf algorithm with a large footprint, and LSstep, a specialized one with a footprint that covers the BIPOP-CMA-ES gaps. These two algorithms covered most of the β-hard instances. To build a proficient portfolio, a third algorithm could be selected between BFGS, Nelder–Doerr and $(1 + 1)$CMA-ES. Given the smaller size of BFGS footprint, the similarities between $(1 + 1)$CMA-ES to BIPOP-CMA-ES, and the speed in which Nelder–Doerr works on lower dimensions (Hansen et al., 2011) and its larger footprint, we conclude that Nelder–Doerr offers the best complement.

As a consequence of the footprint analysis, we estimate that nearly 30% of the known region of the instance space is disputed; that is, no single algorithm could be considered the best. Furthermore, for nearly 15% of the known region there is no consensus as of whether the problems are β-easy or hard. This may be rectified through additional features, but we also need to consider the choice of test instances. Since the bounds of the known region are determined by the instances in the problem subset, $\mathbf{F}$, the results suggest a lack of instances within the disputed areas, and outside the

bounds of the known regions. For example, the tail-like region located at $PC_1 \in [-6, -2]$ and $PC_2 \in [-1, 0]$ is mostly isolated from the rest of the instances. Currently, we lack instances located in the empty space above or below this region; that is, $PC_1 \in [-6, -2]$ and $PC_2 \notin [-1, 0]$, and we ignore the structure that such instances may have. These disputed regions are perhaps the most interesting areas of the instance space, as they represent the phase transitions for each algorithm. Our next objective is to generate test functions that fill the gaps between footprints and also push the bounds of the instance space.

Our analysis is dependent on the quality of the meta-data described in Section 2. For example, our future work focuses on generating a more comprehensive set of test instances, which densely samples the instance space. We expect that these new test instances will alter the overall performance of each algorithm, as it was observed for graph coloring problems (Smith-Miles and Bowly, 2015). Moreover, it is possible that new data modifies the correlations between features, or that algorithm performance is better described by a different set of features. Although a new instance space must be calculated in light of this new evidence, the methodology proposed in this article can be quickly reapplied to updated meta-data.

Other current areas of inquiry are the analysis of the effects that sample size and randomness have on the ELA measures. Broadening the scope of this work, we are adapting these methods to other problems besides black-box continuous optimization, for example, machine learning, time series modeling, and combinatorial optimization. We are also working on building freely accessible web-tools that carry out the footprint analysis automatically. Such tools will be at *www.monash.edu/matilda* in the future.

Acknowledgments

This work is funded by the Australian Research Council through grant No. DP120103678 and Australian Laureate Fellowship FL140100012. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research, and N. Hansen for his clarification on the rationale behind the number of trials and calculation of the ERT during the BBOB sessions.

References

Auger, A., and Doerr, B. (2011). *Theory of randomized search heuristics*. Singapore: World Scientific.

Auger, A., and Hansen, N. (2009). Benchmarking the (1+1)-CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '09)*, pp. 2459–2466.

Auger, A., and Teytaud, O. (2010). Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, 57(1):121–146.

Bartz-Beielstein, T. (2006). *Experimental research in evolutionary computation*. New York: Springer.

Bischl, B., Mersmann, O., Trautmann, H., and Preuß, M. (2012). Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '12)*, pp. 313–320.

Blum, C., Puchinger, J., Raidl, G., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151.

Chang, C., and Lin, C. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems Technology*, 2:27:1–27:27.

Corne, D., and Reynolds, A. (2010). Optimisation and generalisation: Footprints in instance space. In *Parallel Problem Solving from Nature*, pp. 22–31. Lecture Notes in Computer Science, vol. 6238.

Črepinšek, M., Liu, S., and Mernik, L. (2012). A note on teaching–learning-based optimization algorithm. *Information Sciences*, 212:79–93.

De Corte, A., and Sörensen, K. (2013). Optimisation of gravity-fed water distribution network design: A critical review. *European Journal of Operational Research*, 228(1):1–10.

Doerr, B., Fouz, M., Schmidt, M., and Wahlstrom, M. (2009). BBOB: Nelder–Mead with resize and halfruns. In *Proceedings of The Genetic Evolutionary Computation Conference (GECCO '09)*, pp. 2239–2246.

El-Abd, M. (2012). Performance assessment of foraging algorithms vs. evolutionary algorithms. *Information Sciences*, 182(1):243–263.

Hansen, N. (2009). Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of The Genetic Evolutionary Computation Conference (GECCO '09)*, pp. 2389–2396.

Hansen, N., Auger, A., Finck, S., and Ros, R. (2014). Real-parameter black-box optimization benchmarking BBOB-2010: Experimental setup. Technical Report RR-7215, INRIA.

Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2011). Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of The Genetic Evolutionary Computation Conference (GECCO '11)*, pp. 1689–1696.

He, J., Reeves, C., Witt, C., and Yao, X. (2007). A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. *Evolutionary Computation*, 15(4):435–443.

Hough, P., and Williams, P. (2006). Modern machine learning for automatic optimization algorithm selection. In *Proceedings of the INFORMS Artificial Intelligence and Data Mining Workshop*, pp. 1–6.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise differences. *Artificial Intelligence*, 172:1897–1916.

Hutter, F., Xu, L., Hoos, H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

Jansen, T. (1999). On classifications of fitness functions. Technical Report CI-76/99, University of Dortmund.

Jones, T., and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 184–192.

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60.

Langdon, W., and Poli, R. (2007). Evolving problems to learn about particle swarm optimizers and other search algorithms. *IEEE Transactions on Evolutionary Computation*, 11(5):561–578.

Liao, T., Molina, D., and Stützle, T. (2015). Performance evaluation of automatically tuned continuous optimizers on different benchmark sets. *Applied Soft Computing*, 27:490–503.

Lockett, A., and Miikkulainen, R. (in press). A probabilistic re-formulation of no free lunch: Continuous lunches are not free. *Evolutionary Computation*.

Lozano, M., Molina, D., and Herrera, F. (2011). Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computation*, 15:2085–2087.

Lunacek, M., and Whitley, D. (2006). The dispersion metric and the CMA evolution strategy. In *Proceedings of The Genetic Evolutionary Computation Conference (GECCO '06)*, pp. 477–484.

Malan, K., and Engelbrecht, A. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163.

Malan, K., and Engelbrecht, A. (2014). Characterising the searchability of continuous optimisation problems for PSO. *Swarm Intelligence*, 8(4):1–28.

Marin, J. (2012). How landscape ruggedness influences the performance of real-coded algorithms: A comparative study. *Soft Computation*, 16(4):683–698.

Mersmann, O., Bischl, B., Trautmann, H., Preuß, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proceedings of The Genetic Evolutionary Computation Conference (GECCO '11)*, pp. 829–836.

Mersmann, O., Preuß, M., Trautmann, H., Bischl, B., and Weihs, C. (2015). Analyzing the BBOB results by means of benchmarking concepts. *Evolutionary Computation*, 23(1):161–185.

Muñoz, M., and Kirley, M. (2016). ICARUS: Identification of Complementary AlgoRithms by Uncovered Sets. In *IEEE CEC '16*, pp. 2427–2432.

Muñoz, M., Kirley, M., and Halgamuge, S. (2012). A meta-learning prediction model of algorithm performance for continuous optimization problems. In *Parallel Problem Solving from Nature*, pp. 226–235. Lecture Notes in Computer Science, vol. 7941.

Muñoz, M., Kirley, M., and Halgamuge, S. (2015). Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87.

Muñoz, M., and Smith-Miles, K. (2015). Effects of function translation and dimensionality reduction on landscape analysis. In *IEEE CEC '15*, pp. 1336–1342.

Muñoz, M., Sun, Y., Kirley, M., and Halgamuge, S. (2015). Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245.

Penn, E. (2006). Alternate definitions of the uncovered set and their implications. *Social Choice and Welfare*, 27:83–87.

Piotrowski, A., Napiorkowski, J., and Rowinski, P. (2014). How novel is the "novel" black hole optimization approach? *Information Sciences*, 267:191–200.

Pitzer, E., and Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, pp. 161–191. Vol. 378.

Pošík, P., and Huyer, W. (2012). Restarted local search algorithms for continuous black box optimization. *Evolutionary Computation*, 20(4):575–607.

Rice, J. (1976). The algorithm selection problem. In *Advances in Computers*, pp. 65–118. Vol. 15.

Rios, L. M., and Sahinidis, N. V. (2013). Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

Ros, R. (2009a). Benchmarking the BFGS algorithm on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '09)*, pp. 2409–2414.

Ros, R. (2009b). Real-parameter black-box optimisation: Benchmarking and designing algorithms. PhD thesis, Universite Paris-Sud.

Rowe, J., Vose, M., and Wright, A. (2009). Reinterpreting no free lunch. *Evolutionary Computation*, 17(1):117–129.

Schumacher, C., Vose, M., and Whitley, L. (2001). The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '01)*, pp. 565–570.

Seo, D., and Moon, B. (2007). An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. *Evolutionary Computation*, 15(2):169–198.

Smith-Miles, K. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25.

Smith-Miles, K., Baatar, D., Wreford, B., and Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24.

Smith-Miles, K., and Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113.

Smith-Miles, K., and Tan, T. (2012). Measuring algorithm footprints in instance space. In *IEEE 2012 Congress on Evolutionary Computation*, pp. 3446–3453.

Sörensen, K. (2015). Metaheuristics—The metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18.

Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, NTU, Singapore and IIT, Kanpur.

Tang, K., Peng, F., Chen, G., and Yao, X. (2014). Population-based algorithm portfolios with automated constituent algorithms selection. *Information Sciences*, 279:94–104.

Weyland, D. (2010). A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology. *International Journal of Metaheuristic Computing*, 1(2):50–60.

Wolpert, D., and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.