Generating New Space-Filling Test Instances for Continuous Black-Box Optimization

Mario A. Muñoz munoz.m@unimelb.edu.au
Kate Smith-Miles smith-miles@unimelb.edu.au
School of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria 3010 Australia

https://doi.org/10.1162/evco_a_00262

Abstract

This article presents a method to generate diverse and challenging new test instances for continuous black-box optimization. Each instance is represented as a feature vector of exploratory landscape analysis measures. By projecting the features into a two-dimensional instance space, the location of existing test instances can be visualized, and their similarities and differences revealed. New instances are generated through genetic programming which evolves functions with controllable characteristics. Convergence to selected target points in the instance space is used to drive the evolutionary process, such that the new instances span the entire space more comprehensively. We demonstrate the method by generating two-dimensional functions to visualize its success, and ten-dimensional functions to test its scalability. We show that the method can recreate existing test functions when target points are co-located with existing functions, and can generate new functions with entirely different characteristics when target points are located in empty regions of the instance space. Moreover, we test the effectiveness of three state-of-the-art algorithms on the new set of instances. The results demonstrate that the new set is not only more diverse than a well-known benchmark set, but also more challenging for the tested algorithms. Hence, the method opens up a new avenue for developing test instances with controllable characteristics, necessary to expose the strengths and weaknesses of algorithms, and drive algorithm development.

Keywords

Algorithm selection, benchmarking, black-box continuous optimization, exploratory landscape analysis, instance generator.

1 Introduction

In continuous black-box optimization (BBO), as in other optimization fields, the performance of algorithms is assessed experimentally using a collection of well-studied benchmark instances (Whitley et al., 1996). There is often an eager belief that these benchmarks are diverse and representative enough, such that the conclusions drawn from experimentation hold for future unobserved instances. It is also expected that the benchmarks reveal the unique strengths and weaknesses of various algorithms. However, strong evidence suggests that benchmark selection has considerable impact on the algorithm assessment (Liao et al., 2015). Since the current benchmarks may not be sufficient, this article proposes a method to redress the situation. This method enables existing benchmarks to be better understood in the context of the broadest possible instance space. Moreover, the method also generates new test instances with controllable

Manuscript received: 14 August 2018; revised: 28 May 2019 and 5 July 2019; accepted: 8 July 2019.

characteristics, ensuring the diversity we need for insightful algorithm performance analysis.

Careful synthetic generation of benchmark instances for continuous BBO is a necessity, since real-world continuous BBO problems are often ill-defined or involve time-consuming simulations or experiments, thus limiting the number of realistic instances available for testing. Furthermore, synthetic instances can exhibit different properties and structure compared to real-world ones, which merely reflect the kinds of problems from current and past applications, and do not help us prepare for the possibilities of future real-world problems. Synthetic generation of instances with controllable properties is therefore essential to augment algorithm testing beyond a limited set of real-world problems.

The plethora of algorithms that have been proposed for continuous BBO makes rigorous algorithm stress testing all the more critical. The abundance of algorithms is well explained by the fact that BBO problems can be found in diverse practical fields, and that solutions are sometimes urgently needed. Consequently, new methods are often proposed (and sometimes reinvented) by researchers and practitioners from fields outside the core optimization community of applied mathematics and computer science (Sörensen, 2015). However, the resulting algorithmic diversity has compromised our ability to understand the uniqueness, strengths, and weaknesses of most of them (Hough and Williams, 2006), leading to lax research practices, concealment of significant algorithmic innovations (Sörensen, 2015) and recycling of ideas (Weyland, 2010; Črepinšek et al., 2012; Piotrowski et al., 2014). Given that some algorithms are likely to be preferable to others for a given instance (Langdon and Poli, 2007), selecting a good algorithm requires expert knowledge in a broad spectrum of candidates, as well as skills in algorithm engineering and statistics. However, this does not guarantee success, and algorithm selection remains at best cumbersome.

To develop an empirically based theory for algorithm selection that holds to rigorous scrutiny, a robust experimentation protocol should be followed (Hooker, 1995; Smith-Miles and Bowly, 2015). For example, statistical analysis must be used to evaluate the results, as required by most evaluation frameworks (Hansen et al., 2014; Liang et al., 2014); and experimental design methods should be used to develop computational tests (Hooker, 1994), including the generation and selection of a representative set of benchmark instances. However, this latter task is full of challenges. For example, randomly generated instances are insufficient as they lack diversity and rarely resemble real-world ones (Hooker, 1995). Furthermore, limited diversity only highlights small differences in performance between algorithms, making it difficult to discern the source and nature of the differences (Langdon and Poli, 2007). In the long term, new algorithms may be over-fitted to the benchmark set (Hooker, 1995), as the current publication culture seems to encourage the presentation of algorithms that outperform previous approaches, rather than reporting any weaknesses that might be revealed on other test instances (Smith-Miles and Bowly, 2015). Alternatively, custom instances can be generated such that they maximize the performance difference between two algorithms; hence, exaggerating their comparative strengths and weaknesses (Langdon and Poli, 2007). This approach gives us a precise picture of when an algorithm will significantly outperform another; however, this approach cannot generate instances where the difference of algorithm performances is not maximal. Moreover, using algorithm performance as a fitness function to generate harder instances may lead to computationally intractable running times of the generation process (van Hemert, 2006). Therefore, it is desirable to have a method that provides a finer control on the instance structure, while maintaining reasonable computational requirements.

Even if we currently have a representative instance set, it may be difficult to recognize it as such because the space of relevant instances is unknown (Hooker, 1995). However, the extensive work on algorithm selection in recent years (Smith-Miles, 2009; Hutter et al., 2014; Smith-Miles et al., 2014) provides a framework to construct a model of this space; hence, a path to evaluate the representativeness of a set, or to construct a new one if necessary. The key element in the framework is the assessment of the similarities and differences between instances, which is achieved by calculating features that discriminate algorithms by performance or reflect properties of real-world instances (Smith-Miles and Bowly, 2015); hence, the features provide order to the previously ill-defined space of relevant instances. Exploring this space will likely result in atypical instances; that is, they do not resemble any current real-world problems (Hooker, 1995), nor they retain analytic properties that makes convergence analysis possible. This experimental approach complements theory-based ones, while it also gives us an opportunity to future-proof our methods to new challenges arising, as it expands our knowledge of the broadest possible instance space, and the performance of algorithms across it.

This article is the second in a series that develops a methodology for the objective assessment of the strengths and weaknesses of continuous BBO algorithms, based on the algorithm selection framework (Rice, 1976) and design of experiments techniques. In the first article (Muñoz and Smith-Miles, 2017), we estimated and tested a set of features shown to be good predictors of algorithm performance. Then, we built a model of the space of instances based on these features, and used it to identify the algorithm footprints—the regions in which good or exceptional performance is expected—for a group of five algorithms. The methods proposed allowed the visualization of complementary performance between algorithms, quantified the common features of hard problems, and identified the regions where a phase transition may lie. More importantly, the article showed that the benchmark instances employed are clustered in certain areas of the space; hence, they may not be representative of the broadest possible set of test instances. This lack of diversity severely restricts the insights we can gain into unique algorithm strengths and weaknesses. In this article, we extend the methodology to generate new test instances based on target locations in both the feature and instance spaces; hence, achieving a finer control in the instance structure. We illustrate the method using two- and ten-dimensional functions, but it generalizes to other dimensions as well. Using three state-of-the-art algorithms, we demonstrate that the newly evolved test instances are not only more diverse than existing benchmark functions, but some are also more challenging, albeit atypical.

The article continues as follows: Section 2 describes the algorithm selection framework, and the existing benchmark test instances considered and features employed to create a BBO instance space. Section 3 presents our method to fill this instance space with diverse instances evolved via genetic programming, and its results. Section 4 contrasts our method with other instance generation approaches and compares the performance of three algorithms on the new instances and the existing test set. We show that, while not being our explicit aim, the method is able to generate instances that elicit unique performance characteristics from the algorithms. Finally, Section 5 discusses the implications of this work and outlines avenues for further research.

2 Creating a BBO Instance Space

This section summarizes the methodology described in Muñoz and Smith-Miles (2017) used to construct and validate an instance space. The results presented in this article

now include a larger set of benchmark test instances and more features. Aside from these differences, we refer the reader to Muñoz and Smith-Miles (2017) for additional methodological details omitted from this summary.

2.1 The Algorithm Selection Framework

The algorithm selection framework (Rice, 1976) links problem characteristics to algorithm performance, by defining four spaces.[1] First is the ill-defined problem space, $\mathcal{F}$, which contains all the relevant problems (functions) to be solved (minimized). Second is the algorithm space, $\mathcal{A}$, which is composed of all the algorithms applicable to the problems in $\mathcal{F}$, regardless of 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 problem instances; 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) extend the framework by projecting $\mathbf{\Lambda}$ into a two-dimensional instance space, $\mathcal{I} \subset \mathbb{R}^2$, for ease of 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 (Smith-Miles et al., 2014), 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.

For this article, and without loss of generality for maximization, the problems in $\mathcal{F}$ are continuous black-box functions to be minimized, specifically $f : \mathcal{X} \to \mathcal{Y}$, 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 objective or cost value. A target cost value, $\mathbf{y}_t \in \mathcal{Y}$, defines the upper bound of a satisfactory minimization performance from an algorithm. The expected running time, $\hat{t}$, measures the number of function evaluations required to reach y_t within a target precision, e_t, for the first time over a number of runs. The result is normalized over the dimension and $\log_{10}$-transformed. When some runs are unsuccessful, $\hat{t}$ depends on the termination criteria of the algorithm. An algorithm is the best performing on an instance if it minimizes $\hat{t}$ compared to other algorithms.

2.2 Test Instances

We use the noiseless benchmark set from the Comparing Continuous Optimizers (COCO) software (Hansen et al., 2014) as a baseline subset of $\mathcal{F}$. This is a well-studied set composed of 24 basis functions defined within $\mathcal{X} = [-5, 5]^D$ divided 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 ($f_{15} - f_{19}$), and multimodal with weak global structure ($f_{20} - f_{24}$). Qualitative descriptions of each basis function are available in Mersmann et al. (2015).

The COCO software generates new instances by scaling and transforming these basis functions. Transformations include linear translations and rotations, and symmetry

[1] We use the notation by Muñoz and Smith-Miles (2017), where f is a function. In Rice (1976), 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.

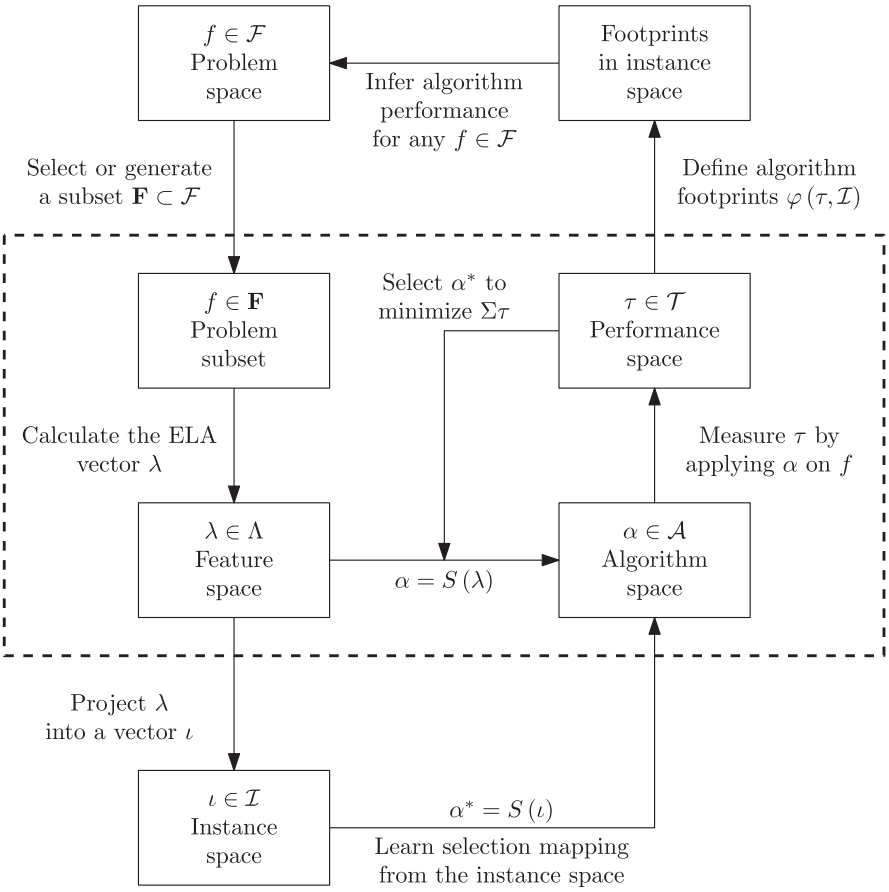


Figure 1: Extended algorithm selection framework by Smith-Miles et al. (2014). The nomenclature has been adjusted for continuous optimization.

breaking through oscillations about the identity. For example, let $f(\mathbf{x}) = \|\mathbf{R}(\mathbf{x} - \mathbf{x}_o)\|^2 + y_t$ be one of the basis functions, where $\mathbf{R}$ is an orthogonal rotation matrix, and $\mathbf{x}_o$ and y_t cause translational shifts on the input and output space respectively. A function instance is generated by providing values for $\mathbf{R}$, $\mathbf{x}_o$, and y_t. To allow replicable experiments, the COCO software uniquely identifies each instance using an index. We generated instances $[1, \ldots, 30]$ for $D = \{2, 3, 5, 8, 10, 20, 40, 60, 100\}$, resulting in 6480 problem instances, from which we generate a preliminary instance space.

2.3 Exploratory Landscape Analysis Features

Exploratory Landscape Analysis (ELA), also known as Fitness Landscape Analysis, is an umbrella term for sample-based methods that produce one or more features related to the characteristics of the cost function (Mersmann et al., 2011). For this work we employed 33 ELA features listed in Table 1, which have shown to be good predictors of algorithm performance (Bischl et al., 2012; Muñoz and Smith-Miles, 2017). To calculate each of them, we generate an input sample, $\mathbf{X}$, of size $D \times 10^3$ candidates using Latin hypercube design (LHD). We deem this sample size to be sufficient considering the

Table 1: List of the Calculated Exploratory Landscape Analysis (ELA) Features.

Correlations (Jones and Forrest, 1995)			
FDC	Fitness distance correlation		
Average distances (Lunacek and Whitley, 2006)			
$DISP_{1\%}$	Dispersion of points within 1% of $\mathbf{y}_t$		
Surrogate models (Mersmann et al., 2011)			
$\bar{R}^2_L$	Adjusted coefficient of determination of a linear model	$\bar{R}^2_{LI}$	Adjusted coefficient of determination of a linear model including interactions
$\bar{R}^2_Q$	Adjusted coefficient of determination of a purely quadratic model	$\bar{R}^2_{QI}$	Adjusted coefficient of determination of a quadratic model including interactions
$\beta_{\min}$	Minimum of the absolute value of the linear model coefficients	$\beta_{\max}$	Maximum of the absolute value of the linear model coefficients
CN	Ratio between the minimum and the maximum absolute values of the quadratic term coefficients in the purely quadratic model	EL_{10}	Mean cross-validation accuracy (MCVA) of a Linear Discriminant (LDA) at 10%
EQ_{10}	MCVA of a Quadratic Discriminant (QDA) at 10%	ET_{10}	MCVA of a Classification and Regression Tree (CART) at 10%
LQ_{10}	The ratio between EL_{10} and EQ_{10}	EL_{25}	MCVA of a LDA at 25%
EQ_{25}	MCVA of a QDA at 25%	ET_{25}	MCVA of a CART at 25%
LQ_{25}	The ratio between EL_{25} and EQ_{25}	EL_{50}	MCVA of a LDA at 50%
EQ_{50}	MCVA of a QDA at 50%	ET_{50}	MCVA of a CART at 50%
LQ_{50}	The ratio between the EL_{50} and EQ_{50}		
Entropic Significance (Seo and Moon, 2007)			
$\xi^{(D)}$	Significance of D-th order	$\xi^{(1)}$	Significance of first order
$\sigma^{(1)}$	Standard deviation of the significance of first order	$\xi^{(1)}$	Significance of second order
$\sigma^{(2)}$	Standard deviation of the significance of second order		

Table 1: Continued.

Cost distribution (Mersmann et al., 2011; Marin, 2012)			
$\gamma(\mathbf{Y})$	Skewness of the cost distribution	$\kappa(\mathbf{Y})$	Kurtosis of the cost distribution
$H(\mathbf{Y})$	Entropy of the cost distribution	PKS	Number of peaks of the cost distribution
Fitness sequences (Muñoz, Kirley et al., 2015)			
$H_{\max}$	Maximum information content	ϵ_S	Settling sensitivity
M_0	Initial partial information		

Table 2: Average silhouette value for each feature cluster obtained using correlation as the dissimilarity measure for k-means clustering.

1.000	CN								
0.907	$H(Y)$	$\beta_{\min}$	$\beta_{\max}$	$\epsilon_{\max}$					
0.534	R^2_Q	R^2_L	R^2_{LI}	R^2_{QI}	EQ_{10}	EQ_{25}	EL_{50}	EQ_{50}	FDC
0.525	LQ_{25}	LQ_{10}	$DISP_{1\%}$	$H_{\max}$	M_0				
0.514	$\gamma(Y)$	$\kappa(Y)$							
0.385	$\xi^{(1)}$	$\xi^{(2)}$	$\xi^{(N)}$						
0.149	EL_{25}	EL_{10}	LQ_{50}						
0.146	PKS	$\sigma^{(1)}$	$\sigma^{(2)}$	ET_{10}	ET_{25}	ET_{50}			

evidence by Kerschke et al. (2016). The output sample, **Y**, is generated by evaluating **X** on each instance from the COCO benchmark. By sharing **X** across instances, we guarantee that the differences observed in the features are not due to sample size or sampling method. Moreover, sharing **X** reduces the overall computational cost, as no new candidates must be taken from the space. All features were scaled to zero mean and unit standard deviation.

Nevertheless, computing 33 features repeatedly during the instance generation process described in Section 3 could be time-consuming. Therefore, we employed feature selection to reduce this set to a more manageable one. Using the data from the 6480 COCO instances, we define a dissimilarity matrix as $1 - \left|\rho\left(\lambda_i, \lambda_j\right)\right|$, where ρ is the Pearson correlation between features λ_i and λ_j. Then, we use this dissimilarity matrix as input to a k-means clustering algorithm, such that similar features are clustered together. To determine the number of clusters, $k = 8$, we use silhouette analysis. The results are shown in Table 2. We leverage our knowledge of the features to select the most suitable ones. For example, CN and R^2_Q can be computed from the same model, while EL_{25} is required to calculate LQ_{25}. Moreover, $H(Y)$ has proven to be an effective predictor of ill-conditioning (Muñoz, Kirley et al., 2015). The resulting feature vector used to summarize each instance is $\lambda = \left[R^2_Q\ CN\ H(Y)\ \xi^{(1)}\ \gamma(Y)\ EL_{25}\ LQ_{25}\ PKS\right]^T$.

To provide evidence that this subset is accurate in its representation of algorithm performance and problem difficulty, we fitted a classification model to predict whether an algorithm can reach the optimum within a target precision of 10^{-8} with a total budget of $D \times 10^6$ function evaluations. The performance of three state-of-the-art algorithms is modeled: BIPOP-CMA-ES (Hansen, 2009), BFGS (Broyden, 1970; Ros, 2009), and Nelder-Mead with Resize and Halfruns (Doerr et al., 2009). We use Random Forests with 100 trees as the classification model, with an out-of-the-bag error of {2.92%, 3.61%, 3.13%} for each algorithm. These results give us confidence that the subset of features is accurate.

2.4 A Preliminary BBO Instance Space

We estimate a preliminary model for the instance space, by projecting into two dimensions the feature vector λ resulting from the COCO benchmark. We use Principal Component Analysis (PCA) to project, and select the two most significant components to generate $\iota \in \mathcal{I}$, a two-dimensional point in the instance space. These two principal components retain 51% of the variance in the data, and define the two axes of the instance

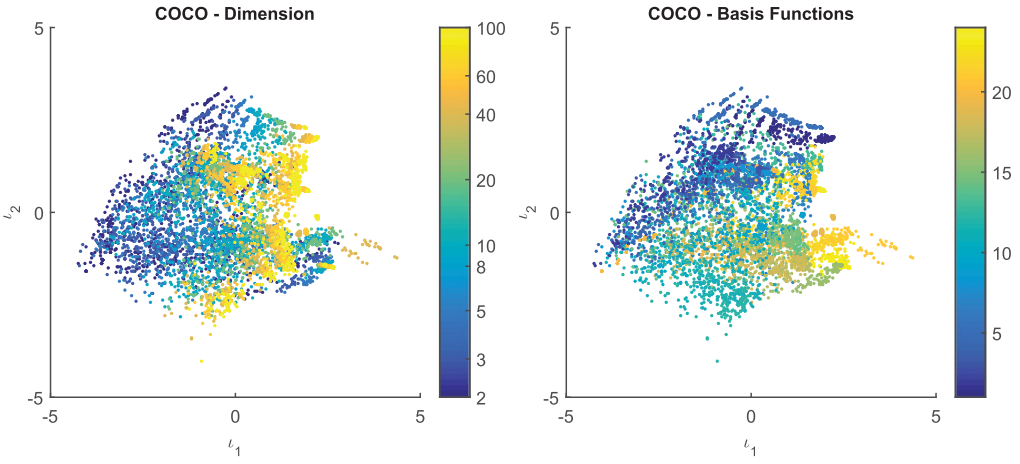


Figure 2: Two-dimensional instance space resulting from projecting the COCO data set using PCA. Instances are colored by (a) dimension, and (b) basis function.

space as linear combinations of the features with coefficients given by Equation (1).

$$\iota = \begin{bmatrix} -0.28454 & 0.61038 \\ 0.20991 & 0.16267 \\ -0.31959 & -0.02927 \\ -0.46389 & 0.26739 \\ -0.30095 & -0.43889 \\ 0.02979 & 0.53365 \\ 0.53101 & 0.01772 \\ -0.42920 & -0.22560 \end{bmatrix}^{\top} \begin{bmatrix} R^2_Q \\ CN \\ H(Y) \\ \xi^{(1)} \\ \gamma(Y) \\ EL_{25} \\ LQ_{25} \\ PKS \end{bmatrix}. \tag{1}$$

The two-dimensional projection is presented in Figure 2, where in general, the dimensionality of the problem increases from left to right. Moreover, the basis functions follow a pattern from top right to bottom left. That is, separable functions ($f_1 - f_5$) are located in the upper left part of the space, whereas multimodal with weak global structure functions ($f_{20} - f_{24}$) are located in the lower right. More importantly, Figure 2 shows limits to the COCO set, and questions whether these functions are sufficiently diverse given the large space to the bounds of $\mathbf{\Lambda}$. Therefore, we will explore this new territory in the instance space, with the aim of understanding the diversity described by the features, and produce a more comprehensive set of test instances.

It is worth acknowledging that the COCO benchmark is not necessarily representative of a broader population of optimization problems. Hence, this preliminary space may be overfitted to it. Moreover, our choice of features might also need to be updated after introducing additional instances, such that we can explain the performance on them. Therefore, the process to generate an instance space can be summarized as follows:

1. Select an instance subset deemed representative of the broad space.
2. Select a feature set and assess its ability to accurately predict performance.
3. Generate an instance space and evaluate the diversity of the instances.

4. If the instances are inadequate, discard those that are easy for all algorithms and generate challenging new instances, and then return to Step 2; otherwise, stop.

Of course, this instance space is dependent on the choice of instances, algorithms, and selected features. Hence, we do not claim that the resulting space is definitive. Nevertheless, it affords an opportunity to gain insights into algorithm strengths and weaknesses, and to identify areas where diversity of benchmarks could be increased to support this quest. The instance space generation is therefore an iterative process, which would end when the most informative subset of features and instances is obtained (Muñoz et al., 2018).

3 Generating New Test Instances Through Genetic Programming

To generate new instances, we use Genetic Programming (GP), a biologically inspired method that evolves computer programs to perform a task. Each program is represented by a binary tree whose leaves are variables or constants, and its nodes are operations such as addition and multiplication. For this article, the task is to generate a program that represents a function, such that when the input sample, $\mathbf{X}$, is evaluated, the position of its feature vector in the instance space is equal to that of a user-defined target. This task is similar to a symbolic regression problem, in which the parameters and structure of the model are adjusted automatically, unlike a traditional regression analysis, in which the user must specify the structure of the mathematical model, for example, linear or quadratic. We use GPTIPS v2.0 (Searson et al., 2010), a MATLAB implementation of GP designed for symbolic regression, and defined the fitness function $J = \|\mathbf{v}_T - \mathbf{v}_G\| + 2^{D_T - D_G}$, where $\mathbf{v}_T$ and D_T are the feature vector and the dimensionality of the target function, and $\mathbf{v}_G$ and D_G are the feature vector and the dimensionality of the generated function. Therefore, the GP routine is allowed to create functions without using all the available variables as allowed by Langdon and Poli (2007), but penalized for it.

The GP routine is run 10 times of 100 generations each using a population of size 400, with a maximum tree depth of 10. Lexicographic tournament selection is employed with ten individuals per tournament, while the best 10% of the population is kept as an elite set. The probability of a mutation, crossover and direct transfer were set to 30%, 60%, and 10%, respectively. We set as stopping criteria a fitness value of 10^{-3}. Constants are generated from a uniform distribution in the $[-100, 100]$ range. The vocabulary of operations used by the GP routine is $\{\times, +, -, x^2, \sin, \cos, \tanh, e^{-x}, e^x, \lfloor x \rceil\}$. We generate two- and ten-dimensional functions, following three strategies to select target points. The first is to validate the method by testing if existing benchmark functions can be recreated, while the second and third enable new test functions to be created in previously unexplored regions of the instance space. Given that the features represent the algorithm performance, our expectation is that by exploring these regions, we can obtain instances with different algorithm performance.

Strategy 1 (S1) We attempt to generate an instance with a similar feature vector as one of the instances from the COCO benchmarks. We randomly select five instances from each basis function, resulting in 120 targets. To account for any random variation, we attempt this experiment five times for the two-dimensional functions but only once for the ten-dimensional functions, due to computational limitations. This results in 600 two-dimensional and 120 ten-dimensional functions. The objective of this strategy is to test whether the feature vector gives us the finer control

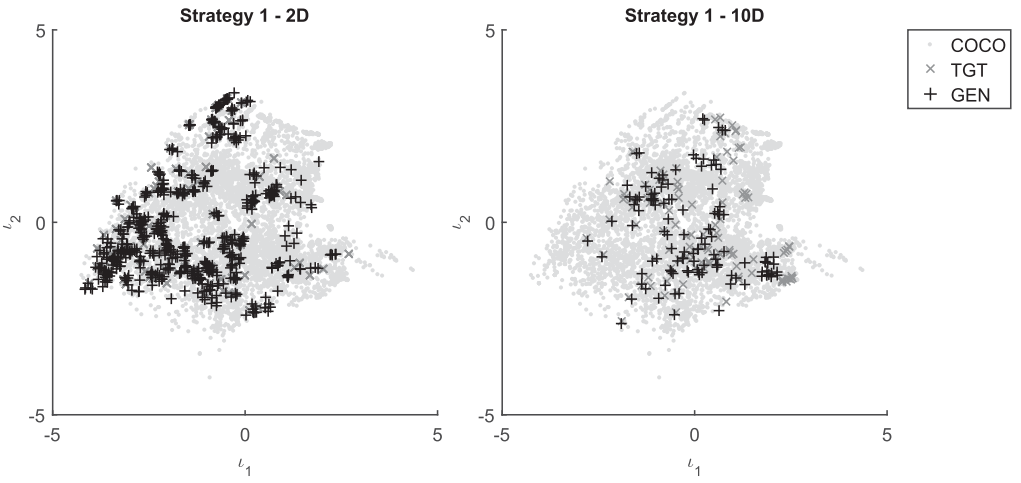


Figure 3: Location of the targets (+) and the generated instances (·) for S1. The location of all COCO instances is provided by the gray background dots. (a) shows the two-dimensional functions, while (b) shows the ten-dimensional functions.

we seek over the instance structure, while pinpointing possible limitations in our approach.

Strategy 2 (S2) We use a LHD sample of 10^2 target points across the eight-dimensional feature space, $\mathbf{\Lambda}$. The bounds of the LHD are defined by the maximum and the minimum feature values from the COCO benchmarks for the given dimensionality. As in S1, we repeat this experiment five times for the two-dimensional functions, resulting in 500 two-dimensional and 100 ten-dimensional functions. The objective of this strategy is to explore new feature combinations, and the structure of the instances that reside in areas of the space not currently covered by COCO.

Strategy 3 (S3) We use a LHD sample of 10^2 target points across the two-dimensional instance space, $\mathcal{I}$. The bounds of the LHD are defined between $[-10,\ 10]^2$, which are larger than the known bounds shown in Figure 2. The two-dimensional projection allows some flexibility in the feature combinations. Therefore, this strategy allows us to test the limits of the instance space, while observing the appearance of functions with feature values beyond the known bounds. Unlike the previous strategies, we repeat this experiment once for both dimensions, resulting in 100 two-dimensional and 100 ten-dimensional functions. Targets are shared for both dimensions.

3.1 Strategy 1: Recreating an Existing Function

Figure 3 illustrates the location of the targets (shown as cross marks) and the generated instances (shown as plus marks) for S1. The two-dimensional functions are presented in Figure 3a, with median RMSE per feature, $\overline{RMSE} = 0.0390$ and Interquartile Range, $IQR = 0.0305$; while ten-dimensional functions are presented in Figure 3b, with $\overline{RMSE} = 0.0847$ and $IQR = 0.1217$. These results demonstrate that the GP is able

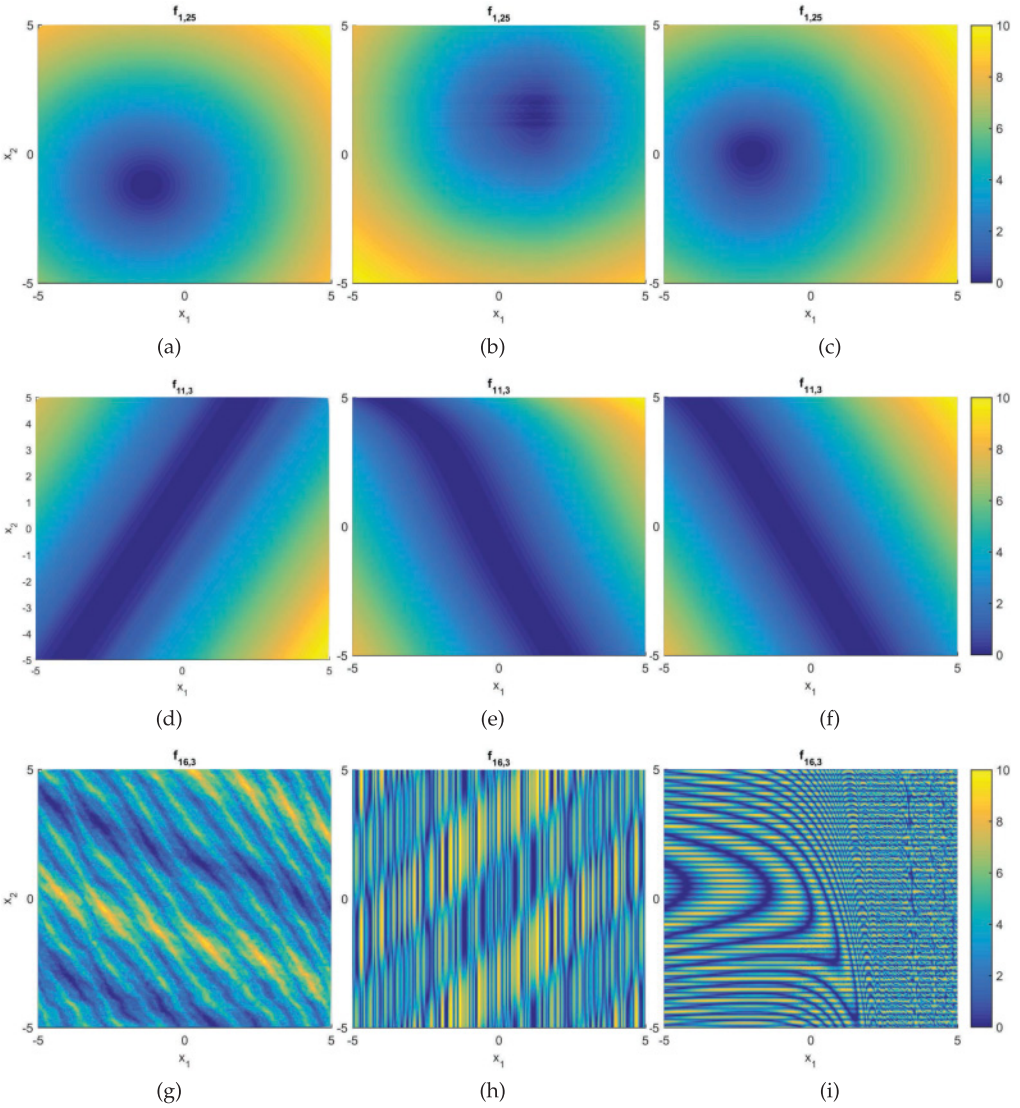


Figure 4: Generated instances following S1. For each of three existing functions (in rows), the leftmost column shows the target instance, the central column shows the generated instance with the lowest error, and the rightmost column shows the generated instance with the highest error. (a) to (c) show the results for $f_{1,25}$, an instance from sphere function; (d) to (f) show the results for $f_{11,3}$, an instance from the Discus function; and (g) to (i) show the results for $f_{16,3}$, an instance from the Weierstrass function.

to find functions with similar feature vectors as the COCO benchmarks in these two dimensionalities.

Figure 4 shows surface plots from three of the 120 randomly selected two-dimensional target instances and the GPs best and worst attempts to evolve functions with similar feature vectors. The leftmost column shows the target instances from the

COCO benchmark set: $\{f_{1,25}, f_{11,3}, f_{16,3}\}$; the central and rightmost column shows the generated instances with the lowest and highest error to target from the five attempts respectively. The shading color represents the objective or cost values, which have been log-normalized to the [0, 10] range, with dark blue representing minimal cost and yellow representing maximal cost.

Figure 4a shows $f_{1,25}$, an instance from the sphere function lying at $\iota = [-1.2330\ 1.0549]$ in $\mathcal{I}$ presented in Figure 2. Because of its symmetry, two instances from the sphere function will have the same features if their global minimum are located at reciprocal locations; that is, $\mathbf{x}_1^\star = |\mathbf{x}_2^\star|$ (Muñoz and Smith-Miles, 2015), as it is the case for the generated instance with the lowest error, illustrated in Figure 4b. On the other hand, the instance with the highest error, illustrated in Figure 4c, is no longer a pure sphere; however, it has an optimum close to the target. Figure 4d shows $f_{11,3}$, an instance from the discus function lying at $\iota = [-1.9480\ \ -0.8734]$ in $\mathcal{I}$. We are able to replicate the poor conditioning of the function, although other structures have appeared. In both generated instances, illustrated in Figures 4e and 4f, respectively, the valley presents as slight bent. Diversions from the target area will result in instances with characteristics of other functions (Muñoz and Smith-Miles, 2015). Figure 4g shows $f_{16,3}$, an instance from the Weierstrass function, which is highly multimodal with a periodic structure lying at $\iota = [0.3365\ \ -2.2125]$ in $\mathcal{I}$. Both generated instances, illustrated in Figures 4h and 4i, attempt to replicate the multimodal structure, with different levels of success. Visual inspection indicates that the best trial closely resembles the target, while the worst has characteristics not observable in the original target function, which may affect algorithm performance. Finally, the following equations show the expressions for the best trials. Equation (2) corresponds to Figure 4b, Equation (3) to Figure 4e, and Equation (4) to Figure 4h. These equations demonstrate that the GP: (a) displaces the global optimum by adding linear terms; (b) introduces coefficients different from one; (c) produces interaction terms between variables; and (d) generates periodic structures through trigonometric terms.

$$y = x_1^2 + x_2^2 - 2x_1 - 3x_2 - 183.2 - \cos\left(\sin\left(e^{x_2^2} - e^{-e^{-x_2{}^4}}\right)\right) - e^{-\cos^2 x_1} - e^{-e^{\sin(\sin x_2)}} \tag{2}$$

$$y = \left(1031.0x_1 + 439.1x_2 - \sin\left(x_2^2\right) + \mathrm{e}^{-x_2} + 17.91\mathrm{e}^{x_2} + 3x_1x_2 - x_2\mathrm{e}^{x_2} - 19.91x_2^2 + 502.0\right)^2 \tag{3}$$

$$y = \left(\left(\cos\left(\mathrm{e}^{7.497-\cos x_1}\right) + \tanh\left(\tanh\left(\cos\left(x_1 - x_2\right)\right)\right)\right)\left(\mathrm{e}^{\cos(\tanh(\cos x_1))} + 7.578\right) - 2.672\right)^2. \tag{4}$$

3.2 Strategy 2: Generating Functions with Novel Feature Combinations

Knowing that we can recreate existing functions, which means that we have finer control over the instance structure, we now proceed to generate new functions that are located in unexplored regions of the feature ($8D$) and instance ($2D$) spaces. Figure 5 illustrates the location of the targets and the generated instances for S2, where target points are selected in the $8D$ feature space and then projected to $2D$ using the same projection as Equation (1). Figure 5a shows the two-dimensional functions, with $\overline{RMSE} = 0.1414$ and $IQR = 0.1134$, while Figure 5b shows the ten-dimensional functions, with $\overline{RMSE} = 0.2360$ and $IQR = 0.1718$. The larger errors than S1 indicate that

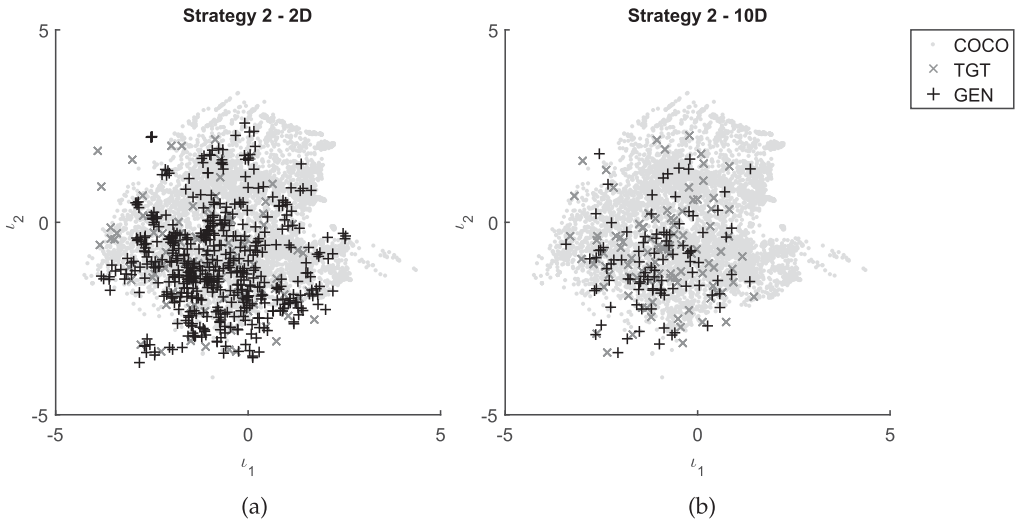


Figure 5: Location of the targets (+) and the generated instances (·) for S2. The location of all COCO instances is provided by the gray background dots. (a) shows the two-dimensional functions, while (b) shows the ten-dimensional functions.

some feature combinations are harder to reach. It is not clear if this is due to computational limitations of the GP implementation, or because the target function locations fall outside the theoretical bounds of what might be possible. It is clear however that the target points selected using S2 do not push the boundaries of the existing benchmarks very far. Hence, we will use S3 to find instances at the bounds of the instance space.

Figure 6 shows contour plot surfaces from a selection of nine of the 500 generated instances using S2. These instances are: $f_{S2,53}$ located at $\iota = [-2.8077\ \ -1.1642]$, an ill-conditioned function, with local optima located in a bent valley; $f_{S2,55}$ located at $\iota = [-1.5798\ \ -0.6928]$, which has two symmetric valleys within a ill-conditioned funnel and a neutral structure at the edge; $f_{S2,126}$ located at $\iota = [-1.5752\ \ -0.4412]$, which has a multimodal grid structure within a sinusoidal valley; $f_{S2,135}$ located at $\iota = [-1.0916\ \ -0.4387]$, which has a large neutral area and a large multimodal area; $f_{S2,215}$ located at $\iota = [-0.6462\ \ -1.9377]$, a multimodal function with periodic structure; $f_{S2,256}$ located at $\iota = [-2.0375\ \ -0.3621]$, which has a single optimum shaped as a triangle bend; $f_{S2,300}$ located at $\iota = [0.2867\ \ -3.3695]$, a mostly flat instance with an optimum located at a narrow valley; $f_{S2,364}$ located at $\iota = [-1.9374\ \ -0.5165]$, which has a ridge separating a higher cost area from the global optima; and $f_{S2,443}$ located at $\iota = [-1.4225\ \ -1.4472]$, which has a unimodal structure, with a large valley leading to the optima at the edge. While some of these instances share high-level characteristics with COCO, i.e., $\{f_{S2,53}, f_{S2,256}, f_{S2,443}\}$ are unimodal, ill-conditioned, with strong global structure (Mersmann et al., 2015), they are not equal. This is because high-level characteristics are ambiguous; that is, there are degrees of modality, conditioning, and global structure, which influence algorithm performance.

3.3 Strategy 3: Pushing the Boundaries of the Instance Space

Figure 7 illustrates the location of the targets and the generated instances for S3. Figure 7a shows the two-dimensional functions, with $\overline{RMSE} = 0.0038$ and $IQR = 0.1710$,

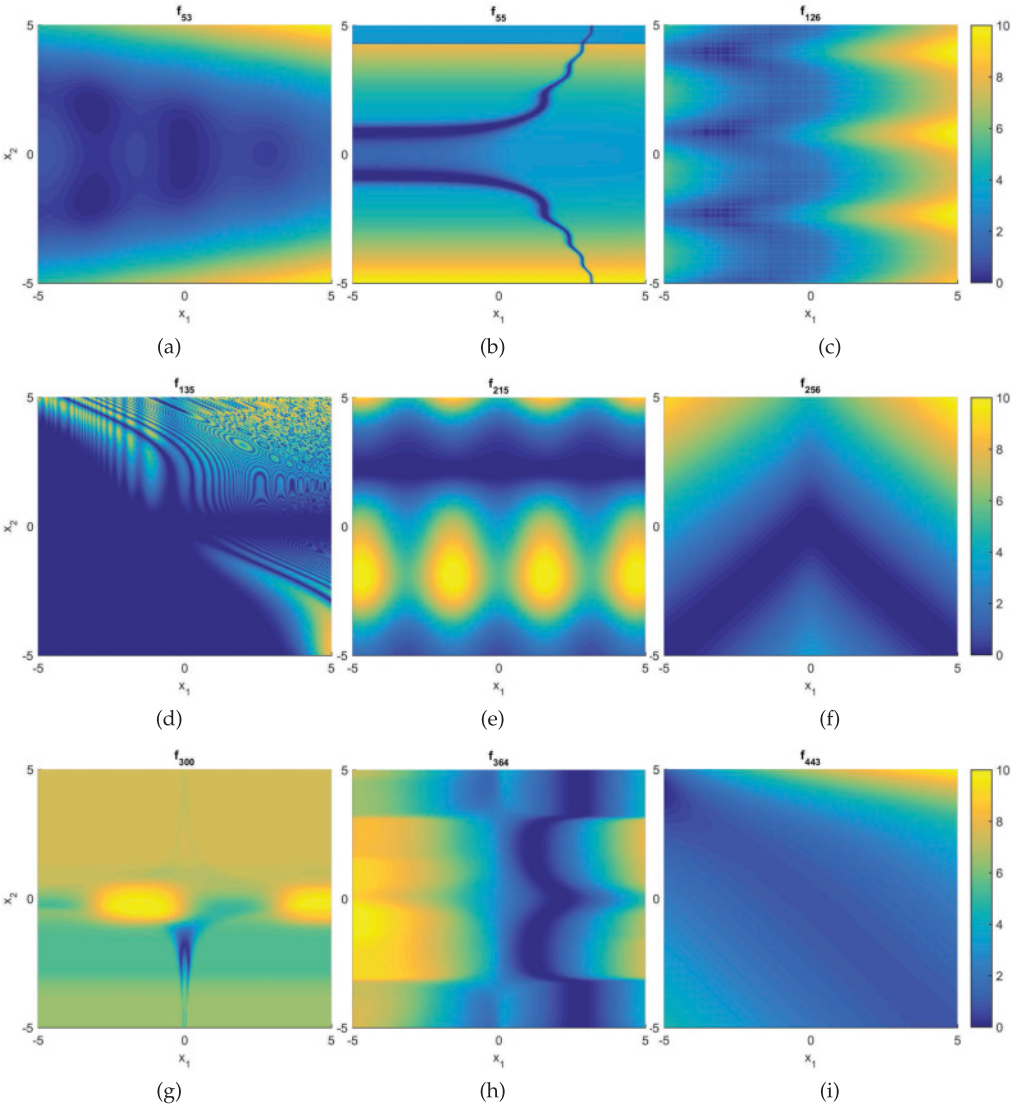


Figure 6: Generated instances following S2. These fall close to the COCO instances in the instance space, but are not colocated with any, thus sharing similar high-level characteristics that are expressed in unique ways.

while Figure 7b shows the ten-dimensional functions, with $\overline{RMSE} = 0.0770$ and $IQR = 0.1892$. Unlike Strategies 1 and 2, the errors are significantly lower. This is due to the targets being defined in $\mathcal{I}$, where an infinite number of feature combinations can produce such locations. Therefore, the GP needs to find only one that it is sufficiently good.

Figure 8 shows surface plots from three of the 100 two-dimensional generated instances, drawn from the convex hull boundary of the 100 locations. Therefore, these instances are some extremes reached by the GP. These instances are: $f_{S3,10}$ located at the top right, $\iota = [9.7466\ 4.3967]$, which has two large neutral areas separated by an

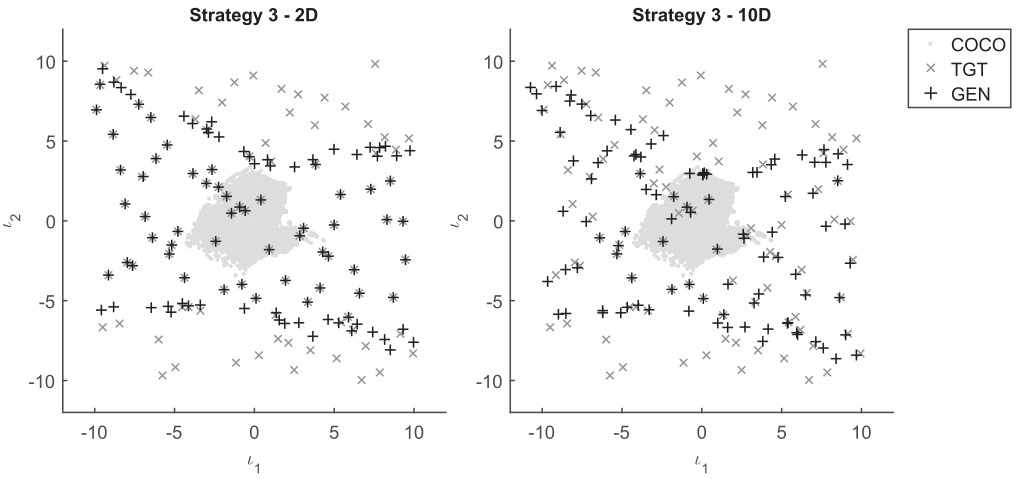


Figure 7: Location of the targets (+) and the generated instances (·) for S3. The location of all COCO instances is provided by the gray background dots. (a) shows the two-dimensional functions, while (b) shows the ten-dimensional functions.

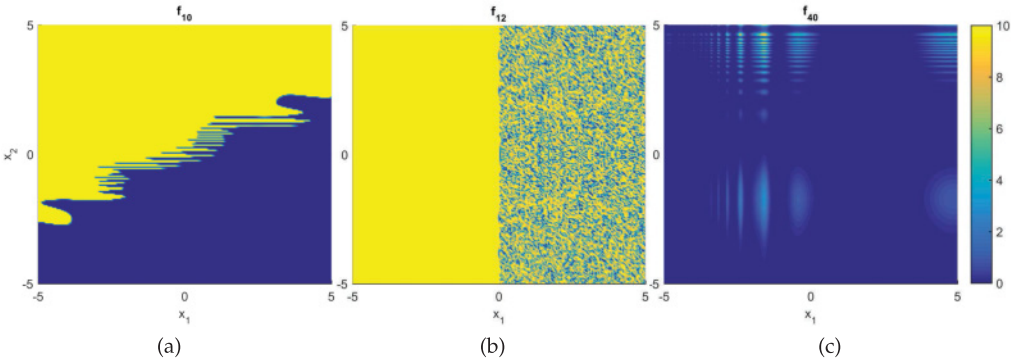


Figure 8: A selection of generated instances following S3, drawn from the convex hull boundary of the 100 locations. The figure demonstrates that at the extremes, one or more features dominate the generation process, leading to mostly neutral functions and without observable structure.

irregular cliff; $f_{S3,12}$ located at bottom right, $\iota = [9.9698\ -7.6042]$, which has one neutral area and one highly rugged area; and $f_{S3,40}$ located at the bottom left, $\iota = [-9.5859\ -5.5881]$, a mostly neutral function with some periodic peaks. These results demonstrate that at the extremes, one or more features dominate the generation process, leading to functions where neutrality dominates. Therefore, it is worthwhile to examine the instances located between the convex hull boundary and the COCO benchmarks, such that we could identify regions where new instances with more diverse characteristics may lie.

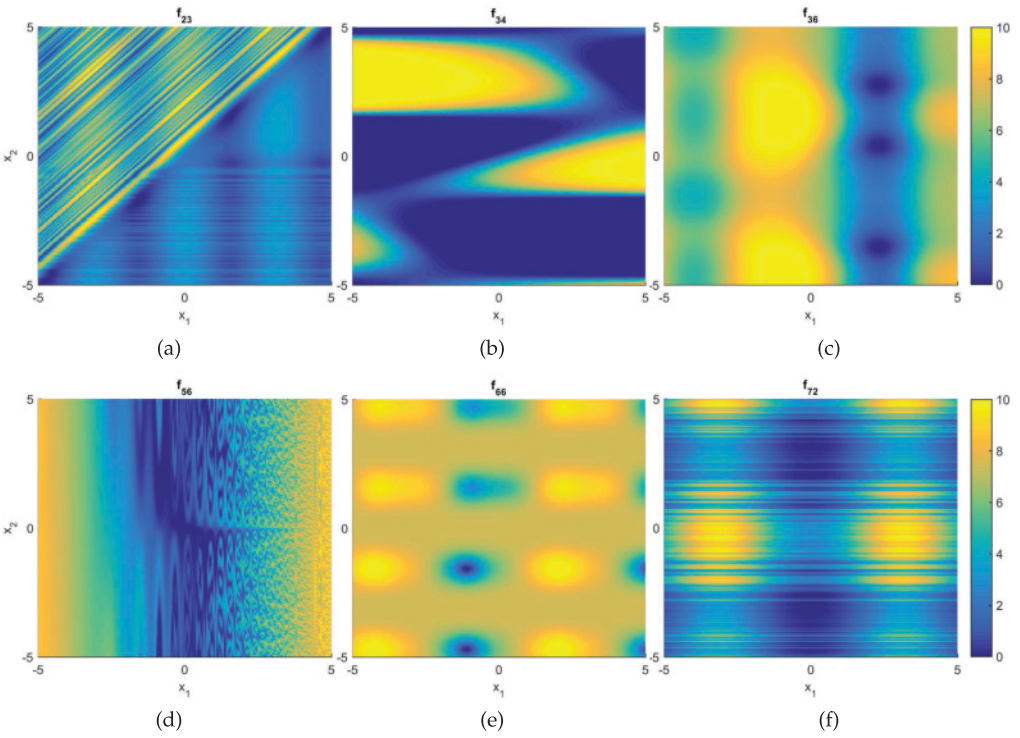


Figure 9: A selection of generated instances following S3, drawn from locations where no COCO instances are available. These functions are close to the convex hull boundary, which indicates that there is a large space between the COCO benchmarks and these functions that could be explored for new challenging instances.

Figure 9 shows surface plots from six of the 100 two-dimensional generated instances, drawn from locations where no COCO instances are available. These instances are: $f_{S3,23}$ located at $\iota = [8.1595 \ -7.4299]$, which has two highly multimodal areas in different directions; $f_{S3,34}$ located at $\iota = [-9.1426 \ -3.3966]$, which has three pronounced ridges; $f_{S3,36}$ located at $\iota = [6.4388 \ -6.4693]$, three deep basins of attraction; $f_{S3,56}$ located at $\iota = [-0.9244 \ 0.8668]$, which is symmetrically multimodal; $f_{S3,66}$ located at $\iota = [9.3112 \ -6.7882]$, which has periodic optima distributed in a grid structure; and $f_{S3,72}$ located at $\iota = [-5.3305 \ -2.0755]$, which, besides a grid structure, has a periodical ruggedness. These functions are not at the convex hull boundary as those in Figure 8, which indicates that there is a large space between the COCO benchmarks and these functions that could be explored for new challenging instances.

4 Analysis

In this section, we analyze the instances generated. First, we compare our results with random generation, from clustering problems (Gallagher, 2016), and evolved using the performance differential of two algorithms as the GP fitness function (Langdon and Poli, 2007) in Section 4.1. Then, we evaluate the probability distribution of performance for BIPOP-CMA-ES, BFGS, and Nelder-Mead in Section 4.2. Finally, we explore whether there are instances that uniquely easy or hard in Section 4.3.

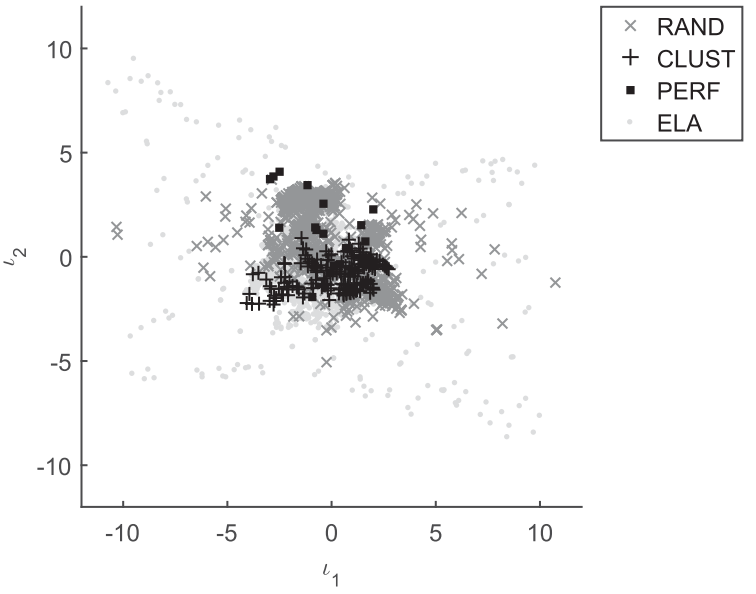
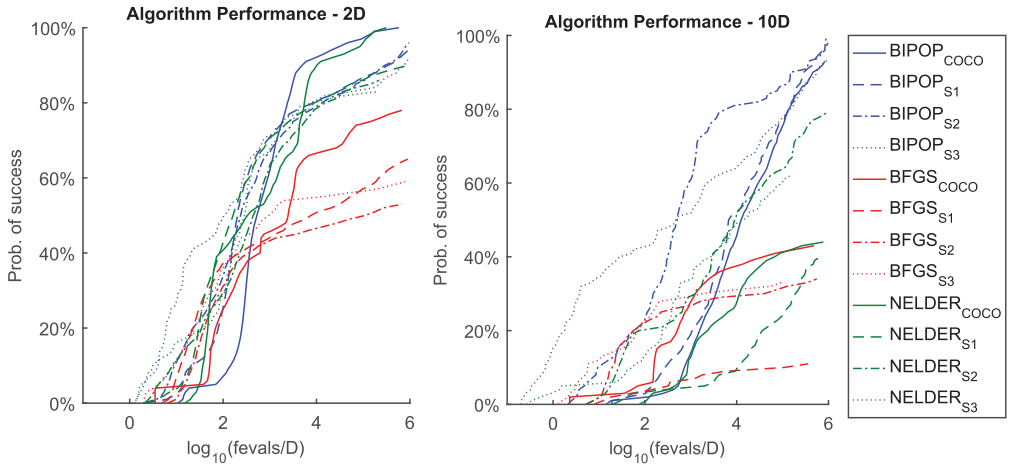


Figure 10: Location of the generated instances using feature targets (ELA), against three other instance generation approaches: random (RAND), derived from clustering problems (CLUST), and instances evolved using the performance differential of two algorithms (PERF).

4.1 Comparison with Other Generation Approaches

Figure 10 shows the location of our generated instances with all of our strategies, against three other instance generation approaches: randomly generated instances from the first generation of the GP, instances derived from clustering problems (Gallagher, 2016) as a type of real-world BBO problem, and instances evolved using the performance differential of two algorithms as the fitness function (Langdon and Poli, 2007). The clustering instances were obtained by setting the number of clusters in the $[2, \ldots, 10]$ range, with the following 18 UCI datasets: (a) Abalone, (b) Balance Scale, (c) Banknote authentication, (d) Blood transfusion, (e) E-Coli, (f) Energy efficiency, (g) German towns, (h) Haberman's survival, (i) Istanbul stock exchange, (j) Iris, (k) Pima Indians' diabetes, (l) Ruspini, (m) Seeds, (n) Stone flakes, (o) User Knowledge Modeling (Test), (p) User Knowledge Modeling (Train), (q) Vertebral column, (r) Wholesale customers data, and (s) Yeast.

Figure 10 demonstrates that random instances span a wider area of the instance space compared to COCO benchmarks; however, this method lacks the control offered through measuring characteristics, and cannot achieve the same reach as the evolved instances. Clustering instances have several advantages as a source of BBO test instances: they are easily scalable, there is certain control over the global optimum, and they have a clear real-world application. They do not appear to exhibit very different features from the current COCO benchmarks though. Finally, instances that have been evolved to accentuate performance differences using the method of Langdon and Poli (2007) are mostly concentrated in the upper-central areas of the current COCO benchmarks and do not give us the reach across the whole instance space we seek, to understand strengths and weaknesses of algorithms in the broadest sense. It is clear



(a) (b)

Figure 11: Probability of success for BIPOP-CMA-ES, BFGS and Nelder-Mead on the COCO benchmark set and the generated instances of (a) two dimensions and (b) ten dimensions.

that our proposed approach to evolving diverse test instances across a given instance space (even if the features that define the space are updated in subsequent iterations of the method) is most effective.

4.2 Probability Distribution of Algorithm Performance

We now evaluate the probability distribution of performance for three state-of-the-art algorithms: BIPOP-CMA-ES, BFGS, and Nelder-Mead on the functions generated in Section 3. The results on the COCO benchmarks indicate that these three algorithms are some of the best performing, and their search mechanisms are significantly different from each other (Hansen et al., 2011).

Unlike the COCO benchmark, we do not know the optimum of the generated functions. Therefore, we cannot specify a target solution, y_t, nor calculate the expected running time, $\hat{t}$. Given that, we set up the following experiment to obtain an estimate. We run all three algorithms with default parameters, bounds to $[-5, 5]$, set an optimal target value of $-\infty$, a total budget of $D \times 10^6$ function evaluations. We allow all algorithms to randomly restart until the budget is exhausted. During the run, we record all the fitness values and define the best one found by any of the three algorithms as the "experimental" optimum. We then set target precisions, e_t, equal to $\left[10^{-3}\ 10^{-4}\ 10^{-5}\ 10^{-6}\ 10^{-7}\ 10^{-8}\right]$. If an algorithm is capable of finding the experimental optimum within a given precision, then the algorithm is successful on that instance. To make a fair comparison, we also run these three algorithms with the same experimental conditions on the 1440 COCO instances, that is, indexes $[1, \ldots, 30]$ at $D = \{2, 10\}$.

Figure 11 shows the probability of success for each of these algorithms. The horizontal axis represents the log-scaled number of function evaluations divided by the dimension, whereas the vertical axis represents the probability of success within the given budget. Figure 11a shows the results for the two-dimensional functions, while Figure 11b shows the results for the ten-dimensional functions. The results show that,

Table 3: Instances from each strategy group that elicit unique performance characteristics from the three algorithms under analysis.

		BIPOP-CMA-ES				BFGS				NELDER			
	D	Easy		Hard		Easy		Hard		Easy		Hard	
COCO	2	0	(0%)	9	(1%)	3	(0%)	79	(11%)	1	(0%)	1	(0%)
	10	0	(0%)	6	(1%)	20	(3%)	0	(0%)	0	(0%)	14	(2%)
S1	2	1	(0%)	5	(1%)	0	(0%)	150	(25%)	1	(0%)	1	(0%)
	10	0	(0%)	0	(0%)	0	(0%)	0	(0%)	0	(0%)	1	(1%)
S2	2	4	(1%)	5	(1%)	0	(0%)	97	(19%)	0	(0%)	4	(1%)
	10	1	(1%)	1	(1%)	0	(0%)	2	(2%)	0	(0%)	1	(1%)
S3	2	0	(0%)	0	(0%)	1	(1%)	19	(19%)	0	(0%)	8	(8%)
	10	10	(10%)	0	(0%)	0	(0%)	7	(7%)	0	(0%)	14	(14%)
Total Generated		16		11		1		275		1		29	

in general, there is a lower probability of success for the newly generated instances, and so they are more challenging. The only exception is for BIPOP-CMA-ES for the S3 ten-dimensional instances, which tend to be solved faster than similar COCO functions.

4.3 Instances with Unique Performance Characteristics

To gather further insight into the performance results, we explore whether there are instances that uniquely easy or hard. Given that there is always an algorithm that finds the experimental optimum within the full budget of $D \times 10^6$, we need strict definitions of what constitutes uniquely easy or hard:

Uniquely easy is an instance that the tested algorithm can reach the tightest precision of 10^{-8} within an average budget of less than $D \times 10^3$, while all comparison algorithms would take an average budget at least one order of magnitude larger.

Uniquely hard is an instance that all comparison algorithms can reach the loosest precision of 10^{-3} within an average budget less than $D \times 10^3$, while the tested algorithm would take an average budget at least one order of magnitude larger.

Table 3 shows the number of instances from each strategy group that are uniquely easy or hard. From a total of 333 generated instances with unique performance characteristics, 275 of them are uniquely hard for BFGS. This does not come as a surprise, as BFGS tends to exploit gradients, which the GP is not explicitly told to generate. For example, the COCO instances marked as easy for BFGS all correspond to the Sphere function, f_1. The location of all of these instances in the space is illustrated in Figure 12, where we observe that several hard instances correspond to S3. While not being an explicit objective of our method, this result show that exploring areas outside the bounds given by COCO can yield instances that improve our understanding of the strengths and weaknesses of each algorithm.

Figure 13 shows some of these instances. Figure 13a shows a uniquely easy instance for BIPOP-CMA-ES, located at $\iota = [1.1970\ 1.3683]$ on the central area of the space. This instance has structure that leads to the upper area, which is mostly neutral and low cost. Figure 13b shows a uniquely easy instance for BFGS, located at $\iota = [5.8986\ \ -6.0352]$ on the lower-right corner of the space. This instance is dominated by an oscillation on x_1.

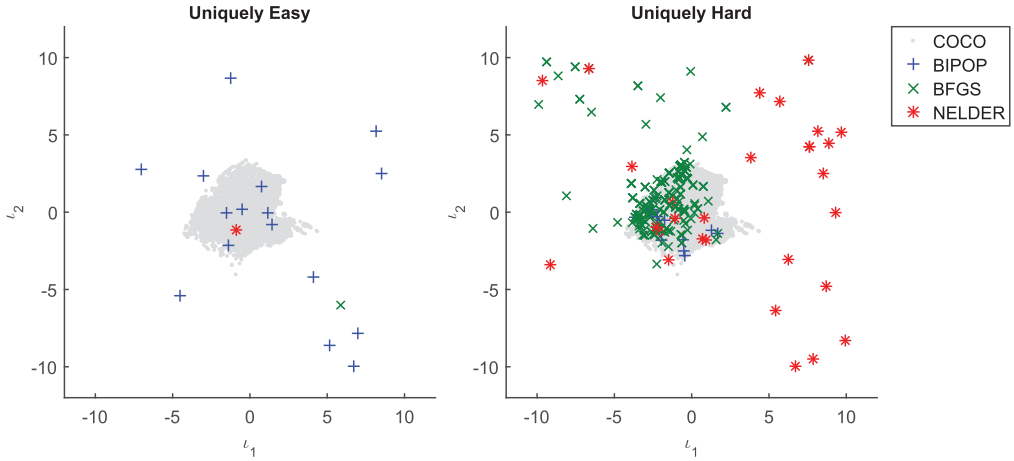


Figure 12: Location of the generated instances that are uniquely (a) easy or (b) hard.

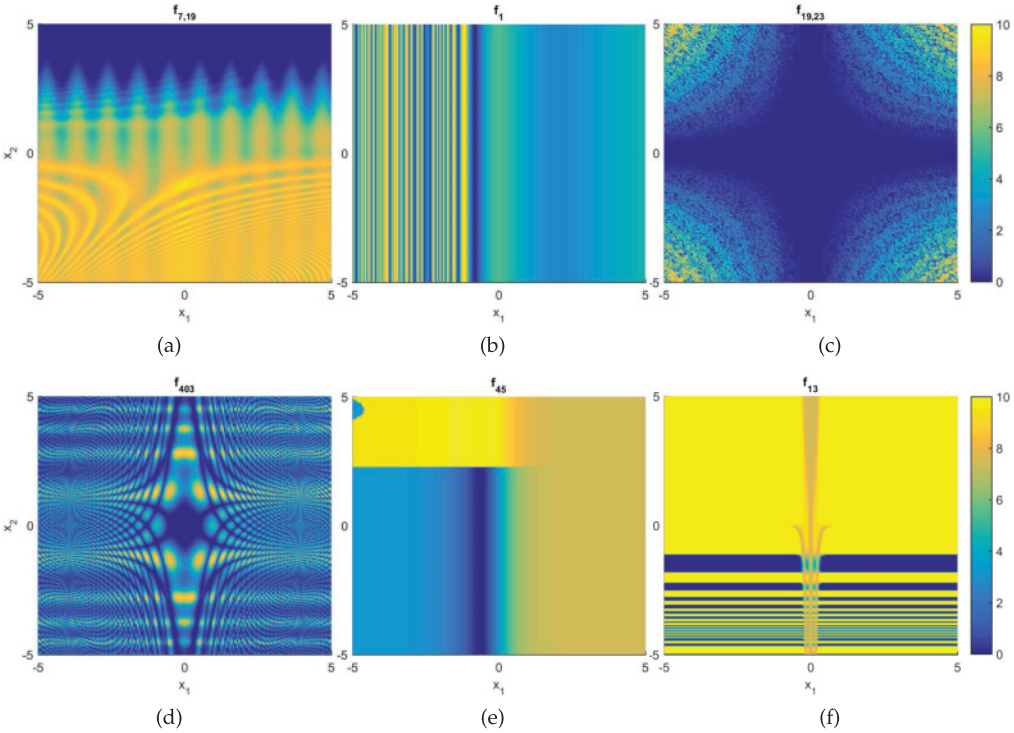


Figure 13: A selection of instances with unique algorithm performance characteristics. The top row are uniquely easy instances, whereas the bottom row are uniquely hard. The leftmost column correspond to BIPOP-CMA-ES, the center column to BFGS, and the rightmost to Nelder-Mead.

However, half of the space leads to the global optimum located in a channel in the central area of the instance. Figure 13c shows a uniquely easy instance for Nelder-Mead, located at $\iota = [-0.9221 \ -1.1338]$ on the central area of the space. This instance, albeit multimodal, has a clear structure that leads to the global optimum. Figure 13d is a uniquely hard instance for BIPOP-CMA-ES, located at $\iota = [-0.7360 \ -2.6003]$ at the lower-left edge of the COCO set. This instance is multimodal with a global structure that creates several funnels that drive away from the global optimum. Figure 13e is a uniquely hard instance for BFGS, located at $\iota = [-6.3863 \ -1.0538]$ at the central-left area of the space. This instance has neutral and deceptive areas, along with abrupt changes that will make the calculation of a gradient not possible. Finally, Figure 13f is a uniquely hard instance for Nelder-Mead, located at $\iota = [8.7052 \ -4.7961]$ at the upper right corner of the space. This instance has a large neutral area crossed by a channel. As a collective, these instances indicate that we need to discard those that are not uniquely easy and return to the feature selection stage of the method.

5 Conclusion

In this article, we have proposed a method for generating diverse instances with controllable characteristics for continuous BBO, as defined by a target vector of Exploratory Landscape Analysis (ELA) features. To illustrate the effectiveness of our method, we constructed sets of two- and ten-dimensional test instances. We employed three strategies to generate our instance sets. In the first strategy, the target was an existing test instance from the COCO benchmark used to validate our ability to recreate a known benchmark function. In the second and third strategies, the targets are selected through a Latin hypercube design (LHD) to encourage an evolved set of instances that span the feature and instance space. The results presented in Section 3 showed that we can fill areas of both spaces previously unexplored. We validated the new instances by testing the performance of BIPOP-CMA-ES, BFGS, and Nelder-Mead on them, and comparing the results to those achieved on the COCO benchmark set in Section 4. We showed that the new test instances have properties that can make them more challenging. We now discuss the implications of this work.

New and more diverse instances are necessary ingredients to provide tools for creating better experiments, drawing solid conclusions, formulating new hypotheses to test, and proposing better theories about algorithm performance. The method presented not only allows us to achieve such outcomes, but also we can catalog the instances through their features, and make predictive models of algorithm performance. Derivations of these instances can be easily obtained. For example, rotations and translations could be generated using the same methodology employed by COCO (Hansen et al., 2014).

While two-dimensional functions can be visually inspected and their characteristics potentially labeled (Mersmann et al., 2015), along with ten dimensions, they are not considered as challenging (Hansen et al., 2011). Therefore, exploring other dimensions is a necessary future step. A more flexible generation routine that allows vector operations, such as Cartesian GP (Miller, 2011), could facilitate the construction of higher-dimensional problems, while losing some control on the ways variables could interact. However, this is limited by the computational cost of generating the instances. The compiled MATLAB code required approximately two hours of computing time in a 3.2-GHz, 16-core, and 32-GB machine, to generate a two-dimensional instance, and approximately 12 hours to generate a ten-dimensional one. The bulk of the computation was spent on calculating the features, whose code was heavily optimized using C/C++ libraries such as OpenCV and MIXMOD, and vectorized MATLAB code.

Fine-tuning the routine parameters may yield functions with less bloat, as observed in Equations (2) to (4), which could potentially reduce computational time. Alternatively, a two-dimensional function can be scaled arbitrarily using a linear combination (Langdon and Poli, 2007), or a chained expansion as commonly employed on the N-Dimensional Rosenbrock function.

The results also suggest that the eight features selected may limit the diversity of the instances generated. For example, at one extreme of R^2_Q we tend to find rather simple functions, as this feature describes a perfectly quadratic function. Testing the generation routine using the complete feature set from Table 1 did not improve substantially the results, while greatly increased the computational cost. This is because the full set contains highly correlated features that seem to give little information to the GP routine. Given the computational cost, we suggest using only unstructured ELA methods (Muñoz, Sun et al., 2015), that is, those that use the same experimental design to extract a sample; hence, they minimize the cost overhead due to the analysis. Some methods worth to explore are those based on Cell-Mapping and Nearest-Better Clustering (Kerschke et al., 2014, 2015). However, we also recognize the limited nature of ELA methods, as they are in essence biased to our understanding of what makes a problem hard. Therefore, a completely new way to describe a function may be necessary. For example, a potentially useful way to characterize the difference between the instances may be the Kullback-Leibler (KL) divergence of the cost distributions. This is an area of great potential and new features can be proposed and validated using the framework presented here.

Our approach to generate targets also influences our ability to generate new instances. By targeting the eight-dimensional feature space we were able to create more complex structures, but it was substantially more difficult to reach the targets. On the other hand, using the two-dimensional instance space, it was much easier to reach the target. However, the two-dimensional projection using principal component analysis (PCA) retains less information from the feature space given that we selected the features using correlation. Therefore, we lost significant control over the properties of the instance. Including the algorithm performance in the mapping as in Muñoz et al. (2018) could improve the results. However, this method would also be affected by the low correlation of the features due to its linear nature. Moreover, collecting the complete performance data could add significant computational work.

Finally, it should be acknowledged that using (a) the root mean-squared error as cost function, and (b) Latin hypercube sampling to target the space, assumes that the geometry of the space is Euclidean and its density is uniform. This results in notions of scale and correlation of the features entering the notion of diversity. Moreover, empty spaces in the space may be unreachable, and targeting them is unlikely to bear good results. While care has been taken to minimize these effects, using a more robust measure of distance such as KL divergence may solve these issues.

Broadening the scope of this work, we are adapting these methods to other problems besides black-box continuous optimization, for example, machine learning (Muñoz et al., 2018), time series modeling (Kang et al., 2017), and combinatorial optimization (Smith-Miles et al., 2014; Smith-Miles and Bowly, 2015). The main differences between BBO and other problem domains is the set of features required to describe the problem instance, and the method used to construct them. We have recently created an on-line tool (MATILDA—Melbourne Algorithm Test Instance Library with Data Analytics, matilda.unimelb.edu.au) to enable automated instance space analysis of a large collection of well-studied problems from optimization and machine learning. All the

instances and results from this article can be downloaded from MATILDA. Researchers can also upload other problems and generate instance spaces, assess the adequacy of benchmark test instances, analyze algorithm performance, and perform automated algorithm selection. We look forward to growing the collection of library problems available on MATILDA in the coming years as the instance space analysis methodology becomes more widely adopted.

Acknowledgments

Both authors were with School of Mathematical Sciences, Monash University, Clayton, VIC 3800, Australia, while conducting the experimental part of this work. Funding was provided by the Australian Research Council through the Australian Laureate Fellowship FL140100012. We also thank Dr. Toan Nguyen, who implemented optimized versions of the metafeatures routines, and Philip Chan, who set up access to additional computational resources for the ten-dimensional function generations.

References

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)*, pp. 313–320.

Broyden, C. (1970). The convergence of a class of double-rank minimization algorithms, 1. General considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90.

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

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

Gallagher, M. (2016). Towards improved benchmarking of black-box optimization algorithms using clustering problems. *Soft Computing*, 20(10):3835–3849.

Hansen, N. (2009). Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 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.

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 and Evolutionary Computation Conference (GECCO)*, pp. 1689–1696.

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

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

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.

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

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.

Kang, Y., Hyndman, R., and Smith-Miles, K. (2017). Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, 33(2):345–358.

Kerschke, P., Preuß, M., Hernández, C., Schütze, O., Sun, J., Grimme, C., Rudolph, G., Bischl, B., and Trautmann, H. (2014). Cell mapping techniques for exploratory landscape analysis. In *EVOLVE V*, Vol. 288 of *Advances in intelligent systems and computing*, pp. 115–131. Berlin: Springer.

Kerschke, P., Preuß, M., Wessing, S., and Trautmann, H. (2015). Detecting funnel structures by means of exploratory landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 265–272.

Kerschke, P., Preuß, M., Wessing, S., and Trautmann, H. (2016). Low-budget exploratory landscape analysis on multiple peaks models. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 229–236.

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.

Liang, J., Qu, B., Suganthan, P., and Chen, Q. (2014). *Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization*. Technical report. Zhengzhou University and Nanyang Technological University.

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.

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

Marin, J. (2012). How landscape ruggedness influences the performance of real-coded algorithms: A comparative study. *Soft Computing*, 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 and Evolutionary Computation Conference (GECCO)*, 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.

Miller, J. (2011). Cartesian genetic programming. In J. Miller (Ed.), *Cartesian genetic programming*, pp. 17–34. Berlin: Springer Natural Computing Series.

Muñoz, M., Villanova, L., Baatar, D., and Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147.

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 Congress on Evolutionary Computation*, pp. 1336–1342.

Muñoz, M., and Smith-Miles, K. (2017). Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary Computation*, 25(4):529–554.

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.

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

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

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

Searson, D., Leahy, D., and Willis, M. (2010). GPTIPS: An open source genetic programming toolbox for multigene symbolic regression. In *International MultiConference of Engineers and Computer Scientists*, pp. 77–80.

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 and Operations Research*, 45:12–24.

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

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

van Hemert, J. (2006). Evolving combinatorial problem instances that are difficult to solve. *Evolutionary Computation*, 14(4):433–462.

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 Applied Metaheuristic Computing*, 1(2):50–60.

Whitley, D., Rana, S., Dzubera, J., and Mathias, K. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85:245–276.