Making a case for (Hyper-)parameter tuning as benchmark problems

Carola Doerr, Johann Dréo, Pascal Kerschke

Making a Case for (Hyper-)Parameter Tuning as Benchmark Problems

- A Discussion Paper -

Carola Doerr
Sorbonne Université, CNRS
Paris, France

Johann Dreo
Thales Research and Technology
Palaiseau, France

Pascal Kerschke
University of Münster
Münster, Germany

ABSTRACT

One of the biggest challenges in evolutionary computation concerns the selection and configuration of a best-suitable heuristic for a given problem. While in the past both of these problems have primarily been addressed by building on experts' experience, the last decade has witnessed a significant shift towards automated decision making, which capitalizes on techniques proposed in the machine learning literature.

A key success factor in automated algorithm selection and configuration are good training sets, whose performance data can be leveraged to build accurate performance prediction models. With the long-term goal to build landscape-aware parameter control mechanisms for iterative optimization heuristics, we consider in this discussion paper the question how well the 24 functions from the BBOB test bed cover the characteristics of (hyper-)parameter tuning problems. To this end, we perform a preliminary landscape analysis of two hyper-parameter selection problems, and compare their feature values with those of the BBOB functions. While we do see a good fit for one of the tuning problems, our findings also indicate that some parameter tuning problems might not be very well represented by the BBOB functions. This raises the question if one can nevertheless deduce reliable performance-prediction models for hyper-parameter tuning problems from the BBOB test bed, or whether for this specific target the BBOB benchmark should be adjusted, by adding or replacing some of its functions.

Independently of the aspect of training automated algorithm selection and configuration techniques, hyper-parameter tuning problems offer a plethora of problems which might be worthwhile to study in the context of benchmarking iterative optimization heuristics.

CCS CONCEPTS

• **Computing methodologies** → **Randomized search**; • **Software and its engineering** → *Search-based software engineering*;

1 INTRODUCTION

Evolutionary algorithms, as well as all other commonly used search heuristics, are, strictly speaking, parametrized frameworks that the user needs to instantiate before execution. The instantiation typically requires to set a number of parameters, such as the population size, search radius, selective pressure, etc. The parametrization offers a great flexibility, which allows the user to adjust the algorithm's behavior to the needs of her problem. However, this flexibility comes at the cost of creating another optimization problem on top: *parameter tuning*.

Parameter tuning was very early identified as a very important problem [20], since the performance of search heuristics is typically highly sensitive to their parameters [11]. To date, the parameter tuning problem is considered to be one of the most important challenges in evolutionary computation (and beyond) [2, 8, 19, 35].

Non-surprisingly, significant research efforts have been undertaken to assist the user in the parameter tuning problem. These works have stimulated the design of automated tools which solve the tuning problem for a given algorithm (on a given sub-problem), see [1, 3, 25, 34, 36] for only a few examples. In a number of complementary workstreams the question of how to *control* good parameter values has been addressed. The underlying observation of parameter control is that – for a very significant fraction of algorithms and many problems – the range of good parameter values can change quite drastically during the optimization process. Parameter control is therefore not only aiming to *identify* optimal parameter values, but also to *track* their evolution throughout the whole run. Paradoxically, parameter control mechanisms are again parametrized, so that one might tend to conclude that not much can be gained by such approaches. Note, however, that they gain us the flexibility to adjust the first-order parameters during the run. Numerous examples are known where parameter control outperforms static choices, see [13, 26] for recent surveys. The parameters of the control mechanism are commonly referred to as *hyper-parameters*, and when considering their optimization we speak of *hyper-parameter tuning*.

Giving the omnipresent use of search heuristics throughout all industrial branches and all data-driven scientific disciplines, we therefore see that the parameter tuning and hyper-parameter tuning problems constitute tasks for which a solid understanding is at very high demand.

Unfortunately, (hyper-)parameter tuning problems can be very complex. Examples in which a small change in the parameter value results in an exponential performance gap can be found in [16, 32]. [32] shows an example for which an exponential performance gap is

caused by a threshold behavior, i.e., there is an abrupt change in performance at a certain parameter value. On the other hand, it has also been demonstrated that a number of parameter tuning problems are actually much more benign (e.g., smooth, convex, and/or unimodal) than one might have expected in the first place [39]. This diversity of behaviours calls for more studies on the understanding of parameter tuning problems, apart from purely performance-oriented studies.

With this discussion paper, we are aiming at starting a focused discussion on how well (hyper-)parameter problems are represented in common benchmarks, and whether there is need and room to extend these benchmarks by (hyper-)parameter tuning problems. With this goal in mind, we provide a preliminary study in which we evaluate the fitness landscape of two hyper-parameter tuning problems taken from the evolutionary computation literature, and compare their structure to those of the 24 noiseless functions of the BBOB test bed [21].

Methodology: Exploratory Landscape Analysis. Exploratory landscape analysis (ELA) [31, 37] is a way to measure the structure of a problem f by sampling from it some (typically random or quasi-random) solutions, evaluating these, and mapping the resulting set of $(x, f(x))$-pairs to a vector of real numbers that are meant to capture the important characteristics of the problem. The objective is to know what makes the problem difficult, so as to adapt the algorithm's behaviour. The landscape-aware parameter tuning approach is thus to observe the problem through the estimation of its landscape *features* and use them to tune the algorithm's parameters to optimize its performance.

This approach led to promising results when trying to find the best suitable hyper-parameter tuning to solve a given problem instance. In this "per instance algorithm configuration" [33] approach, a set of features is first measured on a random sample and the parameter tuning is derived from a previously learned mapping [24]. This approach necessitates to learn the landscape-parameter mapping, called an empirical performance model [4], on an existing benchmark.

In such a setting, the efficiency of the landscape-aware parameter tuning strongly depends on the quality of the benchmark. More precisely, it depends on the so-called "footprint" [27, 38] of the benchmark instances across the features space. If the benchmark does not exhibit (some of) the landscapes of the real-world problems, then the performances of the adapted algorithm may actually be worse than a more generic counterpart.

Thus, the most urgent question addressed by this paper reduces to: *Are landscapes of algorithm configuration problems similar to those observed in benchmarks that are commonly used for learning?*

In this preliminary work, we use the Black-Box Optimization Benchmark (BBOB) [22] as a reference. The noiseless functions of BBOB have been used as the learning benchmark in previous work on landscape-aware per-instance algorithm configuration [4], which led to good performance improvements [6]. However, it should be noted that the landscape-aware parameter tuning problem is in itself a noisy problem involving a machine learning process. As such, the recently proposed "Nevergrad" benchmark [41, 42] may be also considered.

Disclaimer: While we present two case studies within this discussion paper, we do not claim that they are representative. Our main goal is to trigger a discussion on (1) whether feature-based approaches are suitable to discriminate between different types of benchmark problems, (2) if (hyper-)parameter tuning is well represented in the BBOB test bed, in that we can extrapolate reasonable performance predictions from it, and (3) whether hyper-parameter tuning problems are suitable as benchmark problems beyond their own interest, i.e., do these problems provide additional insights into the working principles of common optimization heuristics?

Algorithm 1: The self-adjusting $(1 + (\lambda, \lambda))$ GA variant with five hyper-parameters $\alpha, \beta, \gamma, A, b$

Initialization: Sample $x \in \{0,1\}^n$ u.a.r.;
Initialize $\lambda \leftarrow 1$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Mutation phase:
    Sample ℓ from $\mathrm{Bin}_{>0}(n, p = \alpha\lambda/n)$;
    for $i = 1, \ldots, \lambda_1 = \mathrm{nint}(\lambda)$ **do** $x^{(i)} \leftarrow \mathrm{flip}_\ell(x)$;
    Choose $x' \in \{x^{(1)}, \ldots, x^{(\lambda_1)}\}$ with $f(x') = \max\{f(x^{(1)}), \ldots, f(x^{(\lambda_1)})\}$ u.a.r.;
  Crossover phase:
    for $i = 1, \ldots, \lambda_2 = \mathrm{nint}(\beta\lambda)$ **do** $y^{(i)} \leftarrow \mathrm{cross}_{c=\gamma/\lambda}(x, x')$;
    Choose $y \in \{x', y^{(1)}, \ldots, y^{(\lambda_2)}\}$ with $f(y) = \max\{f(x'), f(y^{(1)}), \ldots, f(y^{(\lambda_2)})\}$ u.a.r.;
  Selection and update step:
    if $f(y) > f(x)$ **then** $x \leftarrow y$; $\lambda \leftarrow \max\{b\lambda, 1\}$;
    if $f(y) = f(x)$ **then** $x \leftarrow y$; $\lambda \leftarrow \min\{A\lambda, n-1\}$;
    if $f(y) < f(x)$ **then** $\lambda \leftarrow \min\{A\lambda, n-1\}$;

2 FIRST CASE STUDY: TUNING THE FIVE-DIMENSIONAL $(1 + (\lambda, \lambda))$ GA

To investigate how well hyper-parameter tuning fits into the BBOB test bed, we describe in this section a first set of experiments, in which we compare the feature values of a hyper-parameter tuning problem with those of the 24 noiseless BBOB functions. More precisely, we consider the fitness landscape of the five-dimensional hyper-parameter tuning problem of the generalized $(1 + (\lambda, \lambda))$ GA presented in [9]. We tune its performance on the 5,000-dimensional OneMax problem. The generalized $(1 + (\lambda, \lambda))$ GA is summarized in Algorithm 1, we do not discuss it in great detail here, and point the interested reader to [9, 12, 14] for an in-depth discussion of this algorithm. For the tuning problem we consider the following ranges: $\alpha \in [0.33, 10]$, $\beta \in [1, 10]$, $\gamma \in [0.333, 10]$, $A \in [1.01, 2.5]$, and $b \in [0.4, 0.99]$. For the BBOB test bed, we used the R-package smoof [7] and considered the first five instances per function. A detailed description of the BBOB functions can be found in [22], and a survey on the COCO benchmarking platform, which BBOB is a part of, is available at [21].

For each of the 120 BBOB instances (24 functions with five instances each), as well as for the hyper-parameter tuning problem, we used a low-discrepancy point set generator to sample $3^5 = 243$ points x in total at which we evaluated the fitness. More precisely,

we have used a Halton point set for the generation of the configurations at which we evaluate the $(1 + (\lambda, \lambda))$ GA tuning problem [40], and we use a Latin Hypercube sample for the BBOB evaluations (for convenience, as it is already implemented as standard sampler in FLACCO [23, 31], the tool that we use to compute the feature values, see below). For the hyper-parameter tuning problem we scale the $[0, 1]^d$-distributed points from the generator to the above-specified range of the parameters $\alpha, \beta, \gamma, A, b$. For each of the 243 configurations we run the self-adjusting $(1 + (\lambda, \lambda))$ GA eleven times, average the optimization times of the resulting eleven values, and assign this average running time as fitness to the respective configuration.

The heatmaps in Figure 1 give a first indication how the sampled configurations are distributed, and how the fitness landscape of the five-dimensional tuning problem looks like, in a global perspective (upper figure) and for a zoom into the configurations achieving an average optimization time of at most 10^7 function evaluations (bottom figure). We note here that none of the evaluated configurations performs extremely well: the best configuration has an average running time of around 43,395 evaluations, whereas in [9] several configurations are reported to have a much smaller average optimization time, which can go as low as 29,000 function evaluations. We also note that the plots indicate some regularity, despite the fact that the Halton point set was meant to provide a good distribution. It might be worth investigating how much our results would change depending on the sample distribution. The point sets from [40] are known to achieve low L2-discrepancy, but other measures such as the star discrepancy or energy level measures might be more suitable. This, however, forms a separate research thread which we ignore in this present work.

For each of the 121 sets of 243 $(x, f(x))$-pairs we then compute the feature values using FLACCO [23, 31]. Since some feature computations are stochastic, we perform ten independent runs per data set, leaving us with 1,200 feature vectors for the BBOB functions and ten feature vectors for the hyper-parameter tuning problem.

For a first indication how similar the tuning problem is to those in the BBOB testbed, we next determine minimum and maximum for each of the 66 features (we here disregard all *basic* features, as well as cost-related subfeatures as they are not related to the fitness landscape itself). We then evaluate if the feature values of the tuning problem are within the range of the BBOB functions. As a result, we obtain that all but four feature values fall in the BBOB range. Figure 2 depicts the distribution of the conspicuous feature values for the 120 BBOB instances against the value of the tuning problem. All four features are based on deterministic computations, and the distribution is therefore only with respect to the 120 deterministic feature values.

The idea behind the four conspicuous features mentioned above can be summarized as follows: the cell mapping angle (*cm_angle*) features [28] discretize the search space into three equidistant intervals per dimension, resulting in $3^5 = 243$ cells in total. For all non-empty cells, i.e., cells which contain at least one of the sampled observations, the angle (in degree) between the worst, the center and the best observation of the respective cell is computed. Afterwards, the angles of all feasible (i.e., non-empty) cells are aggregated by means of the arithmetic mean (*angle.mean*) and their standard deviation (*angle.sd*). For the nearest better clustering (*nbc*) features [29], two distance sets are computed. The first distance set comprises the distances of all points to their respective nearest neighbors, whereas the second one contains all nearest better neighbor distances, i.e., the distances to the nearest neighbor among all observations with a better fitness value. *nn_nb.mean_ratio* computes the ratio between the mean distances of the two distance sets. In contrast, *dist_ratio.coeff_var* first computes the distance ratio per observation and afterwards aggregates these ratios by means of their coefficient of variation (arithmetic mean divided by standard deviation).

2.1 Averaged Feature Analysis

We perform a second analysis in which we average the feature values across the ten independent runs of FLACCO and, in case of the BBOB functions, the five instances. This approach reveals seven (feature) outliers, whose distributions are summarized in Figure 3. The features listed therein extend the previous four outliers from Figure 2 by one gradient homogeneity (*cm_grad*, [28]) feature and two y-distribution (*ela_distr*, [37]) features. For the former, we first compute the (normalized) gradients between each point in a cell and its nearest neighbor, then point each gradient towards its better end, and afterwards sum up all gradient vectors (per cell). At last, the lengths of all cumulated gradient vectors are aggregated by means of the standard deviation (*sd*). In contrast, the y-distribution features simply aggregate the frequency of the fitness values. More precisely, *kurtosis* and *skewness* are the kurtosis and skewness of all fitness values $f(x)$ from the sample.

2.2 Tuning vs BBOB Functions

While we have merely looked at aggregated data in the analyses above, we next compare the feature values of the tuning problem to each of the individual BBOB functions. To this end, we compute for each of the 24 BBOB functions the vector of average feature values, and compare them to the average feature values of the tuning problem. We recall here that for the tuning problem, the averaging only concerns the randomness stemming from the feature computation, whereas for the BBOB functions we also aggregate over the five instances. Figure 4 displays the results of this comparison. While we see a seemingly good fit of the ELA feature values of the tuning problem with those of BBOB's F1, F3, F8, F13, F14 and F15 (see Figure 4b), no BBOB function could be identified for a good fit with respect to the other feature sets displayed in Figure 4a and 4c.

The plots in Figure 4 also raise the interesting question if one can design an aggregated measure that captures the similarity between two or more fitness landscapes. To this end, a more solid understanding of the relevance of each feature needs to be developed.

3 SECOND CASE STUDY: THE (1+1) EA WITH SELF-ADJUSTING MUTATION RATES

We add a second case study to our findings from Section 2, in which we consider the algorithm proposed in [17], a (1+1) evolutionary algorithm (EA) with self-adjusting mutation rates. Its pseudocode can be found in Algorithm 2. The $(1 + 1)$ $\text{EA}_{>0}$ works as follows: after a random initialization, the algorithm always maintains a best-so-far solution x, breaking ties towards the last evaluated search point of current-best fitness value. In each iteration one

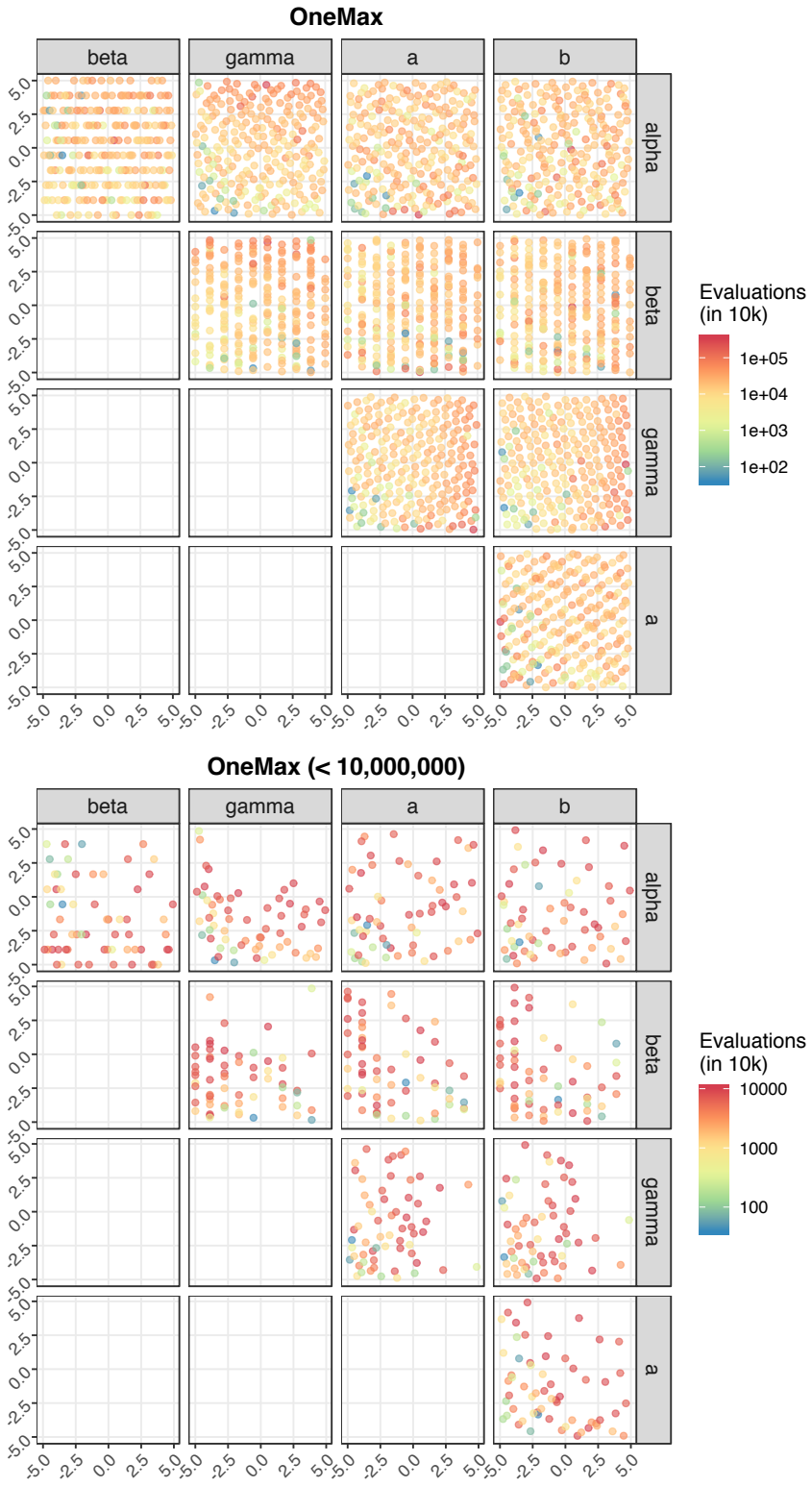


Figure 1: Heatmaps depicting the pairwise hyper-parameter landscapes of the "GA on OneMax" tuning problem for two precision values: all configurations (top) and configurations with less than 10^7 function evaluations (bottom).

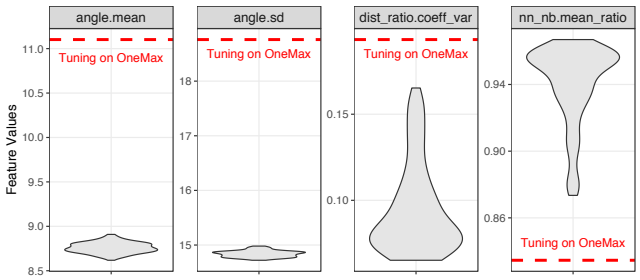


Figure 2: Features based on 120 BBOB instances (shown as violin plots) in comparison to the respective feature values based on the hyper-parameter landscape of the "$(1 + (\lambda, \lambda))$ GA on OneMax" tuning problem (red horizontal line), for the four features for which the tuning problem does not fall into the range of the BBOB instances.

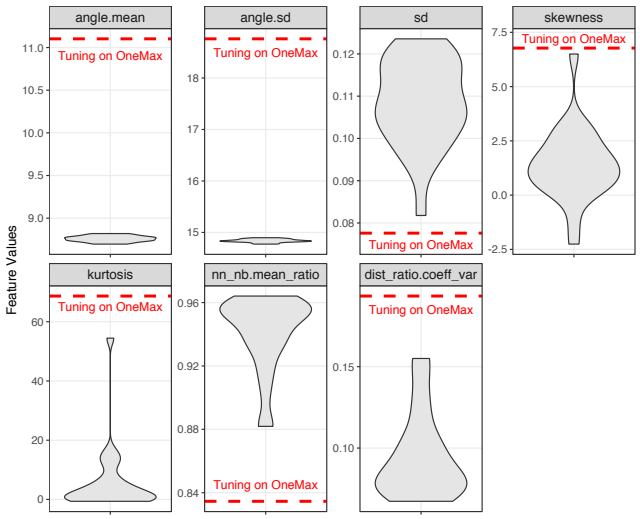


Figure 3: Features based on the 24 BBOB functions (shown as violin plots) in comparison to the respective feature values based on the landscape of the "$(1 + (\lambda, \lambda))$ GA on OneMax" hyper-parameter tuning problem (red horizontal line), for the seven features for which the tuning problem does not fall into the range of the BBOB functions.

new offspring y is created by flipping each bit of the *parent* x independently with mutation rate p (standard bit mutation). When $y = x$ a new bit string is sampled until we obtain an offspring $y \neq x$. Equivalently, we can sample the *mutation strength*, i.e., the number of (randomly distributed) bits to be flipped, from the conditional binomial distribution $\mathrm{Bin}_{>0}(n, p)(k)$. This distribution evaluates to 0 for $k = 0$ and to $\mathrm{Bin}(n, p)(k)/(1-(1-p)^n) = \binom{n}{k} p^k (1-p)^{n-k}/(1-(1-p)^n)$ for $1 \leq k \leq n$. This is the description used in Algorithm 2 and explains the subscript "$_{>0}$" in the name of the algorithm. Once an offspring $y \neq x$ is sampled, the algorithm evaluates the fitness $f(y)$ and replaces the parent x by y if and only if $f(y) \geq f(x)$. In this case the mutation rate p is increased to Ap, where $A > 1$ is one

Algorithm 2: The $(1 + 1)$ EA$_{>0}$ with update strengths A and b and initial mutation rate $p_0 \in [1/n^2, 1/2]$ for the maximization of a pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and compute $f(x)$;
Set $p = p_0$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Sample ℓ from $\mathrm{Bin}_{>0}(n, p)$;
  $y \leftarrow \mathrm{flip}_\ell(x)$;
  evaluate $f(y)$;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$ and $p \leftarrow \min\{Ap, 1/2\}$
  else
    $p \leftarrow \max\{bp, 1/n^2\}$

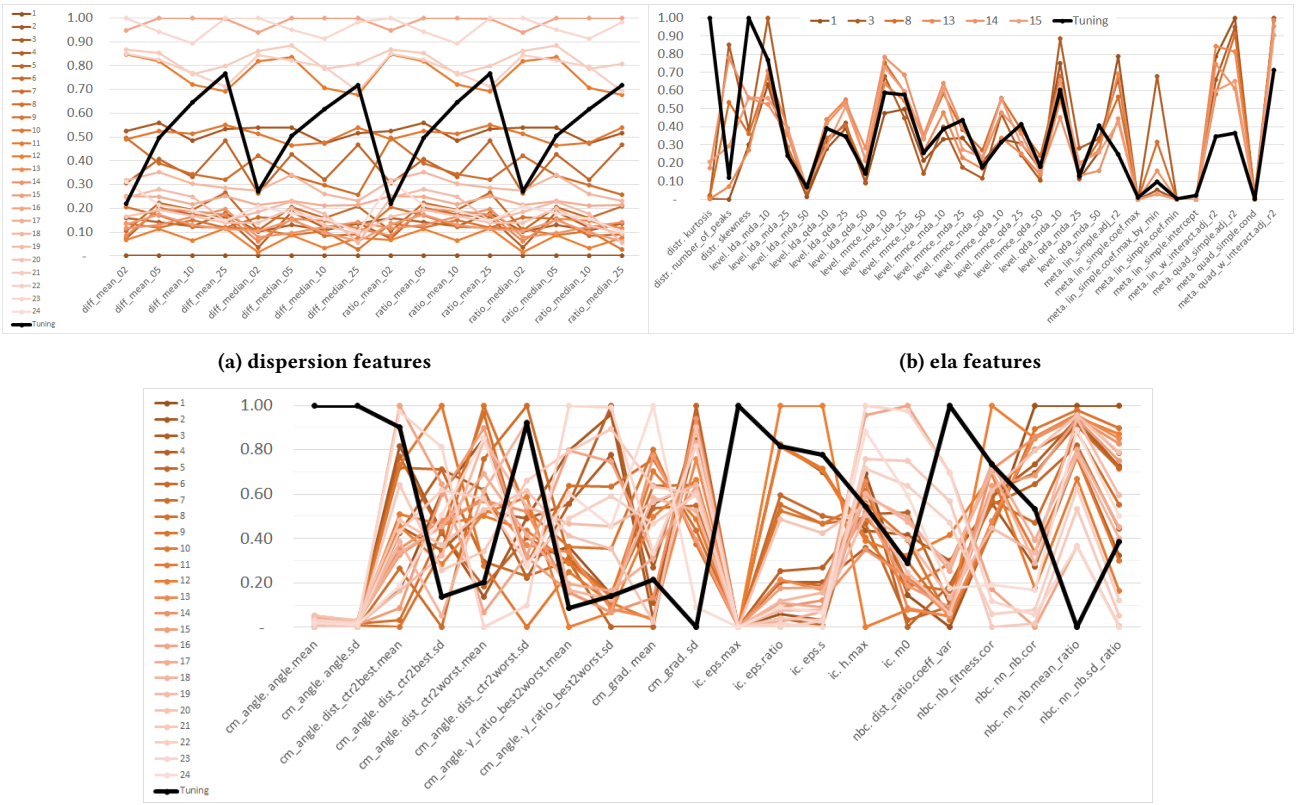
of the three hyper-parameters of this algorithm. When $f(y) < f(x)$ the offspring y is discarded and the mutation rate lowered to bp, where $b < 1$ is the second hyper-parameter. The search process continues until a user-defined stopping criterion is met.

The self-adjusting $(1+1)$ EA$_{>0}$ has thus three hyper-parameters: the *update strengths* A and b and the initial mutation rate $p_0 \in (0, 1)$. It has been demonstrated in [18] that the influence of the initial mutation rate is negligibly small when considering "easy" optimization problems such as OneMax and LeadingOnes. We therefore fix in the following $p_0 = 1/n$ and concentrate on the two-dimensional tuning problem $A \in (1, 6]$ and $b \in (0, 1)$. This is the tuning problem that was studied in [17], with a purely performance-oriented mindset. Here, instead, we are rather interested in the characteristics of its fitness landscape.

We consider as test case the $(1+1)$ EA$_{>0}$ on the 250-dimensional LeadingOnes problem $\mathrm{Lo} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \in [i] : x_j = 1\}$. In this section, all numerical evaluations for the tuning problem are built upon the data from [17]. This data is as follows: For each configuration $x = (A, b)$ with $A \in \{1 + 0.1k \mid 1 \leq k \leq 50\}$ and $b \in \{0.02k \mid 1 \leq k < 50\}$ we assign as $f(x)$-value the average running time of 101 independent runs of the self-adjusting $(1+1)$ EA$_{>0}$ with update strength A and b. We note that this results in a total number of 2,450 $(x, f(x))$-pairs for the tuning problem. For the BBOB data, however we build our computations on $50D = 100$ points only (for the simple reason that we had this data readily available - an update with a similar-sized sample will be done for the camera-ready version, should this paper be accepted).

3.1 Alternative Parameter Representations

We briefly note that a recent work [15] shows that a better parametrization of A and b would require that $b = (1/A)^{1/(s-1)}$, so that A is the update strength and s the *success rule* of the algorithm. This setting generalizes the well-known *one-fifth success rule* from evolution strategies [10, 43, 44]. It was proven in [15] that the optimal update strength F for the $(1+1)$ EA$_{>0}$ on the LeadingOnes problem satisfies $F = 1 + o(1)$, while the optimal success rule is around 1.285. With this setting, the expected optimization time on the n-dimensional LeadingOnes function is around $0.404n^2$ (to be very precise, the latter is based on a numerical evaluation



(a) dispersion features **(b) ela features**

(c) cell mapping, information content, and nearest better clustering features

Figure 4: Comparison of selected feature values of the tuning problem (bold black line) with those of the 24 BBOB functions (slim red lines). All values have been normalized to the interval $[0, 1]$

of an otherwise rigorous mathematically proven bound). Since such a parametrization requires a very deep understanding of the underlying optimization process, and since we want to build our analysis upon the data provided by [17], we ignore this alternative parametrization in the following, and consider the tuning of A and b as used in Algorithm 2. We mention this alternative representation because we consider it an interesting question to investigate the influence of such representations on the fitness landscapes and on the difficulty of the underlying tuning problem. We plan to address this question in future work.

3.2 Heatmaps

To get a first impression of the global fitness landscape, we first plot again some heatmaps of the tuning problem, cf. Figure 5. The left figure displays the full data set, whereas the ones in the middle and on the right zoom into those configurations which obtain an average running time of at most 50,000 and 30,000, respectively. For comparison, the best configurations achieve a running time slightly above $25{,}000 \approx 0.403n^2$.

We observe that the landscape appears to be very flat when looking at the largest resolution. However, some structure can be observed when zooming into the more performant runs. In principle, it might be possible to extend the results from [15] to formulate the dependence of the average running time of the self-adjusting $(1+1)$ $\text{EA}_{>0}$ on the LeadingOnes problem. Based on those results, a rather smooth fitness landscape can be expected. The heatmap structure may indicate that this tuning problem is not too different from the BBOB functions, a question that we will evaluate in more detail in the next subsection.

3.3 Feature Value Comparison

Similar as in Section 2 we again first look for feature values of the tuning problem that do not fall into the range of the 120 BBOB instances. This is the case for all but three features, whose distributions are illustrated in Figure 6.

The first feature (*dist_ctr2worst.mean*) again exploits information based on the $3^5 = 243$ cells of the discretized search space. For all (non-empty) cells, it computes the distance between the cell's worst point and center. Afterwards the computed distances are averaged across the non-empty cells. The second feature (*nb_fitness.cor*) measures the correlation between an observation's fitness value and its "indegree" (i.e., the number of points from the sample for which the current observation constitutes the nearest better neighbor). At last, *nn_nb.sd_ratio* is the ratio between the standard deviations of the nearest neighbor and nearest better neighbor distance sets.

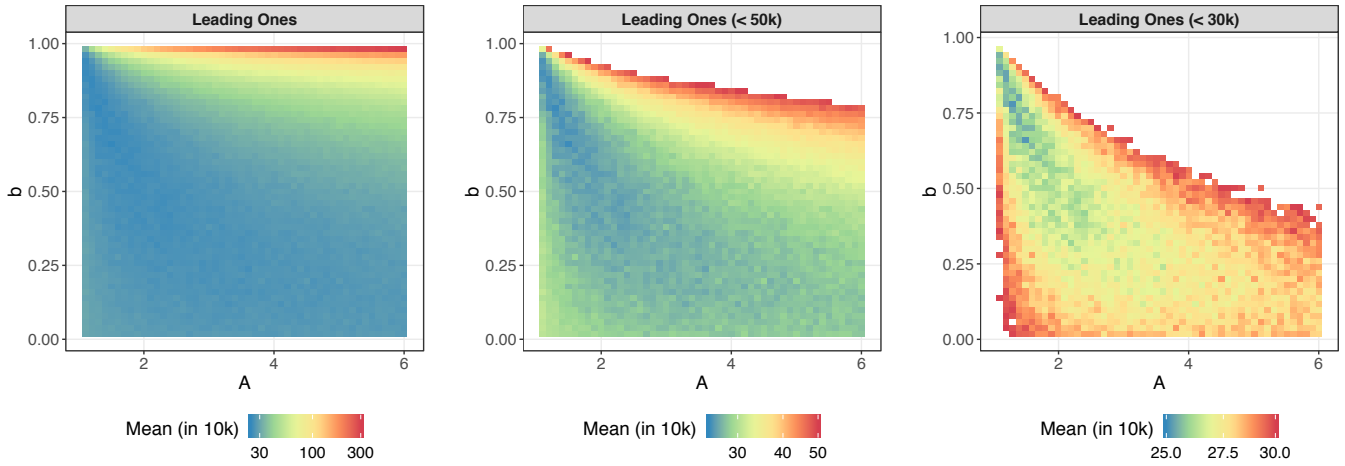


Figure 5: Heatmaps depicting the hyper-parameter landscapes of LeadingOnes for different precision values: all configurations (left), as well as configurations with $< 50k$ (middle) and $< 30k$ evaluations (right).

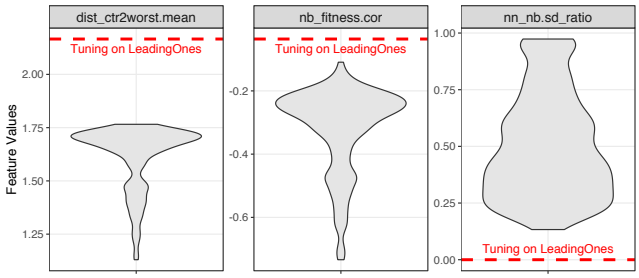


Figure 6: Features based on the 24 BBOB functions (shown as violin plots) in comparison to the respective feature values based on the landscape of the LeadingOnes hyper-parameter tuning problem (red horizontal line), for the three features for which the tuning problem does not fall into the range of the BBOB functions.

3.4 Detailed Comparison

Finally, we investigate how the feature values of the parameter tuning problem compare to those of the BBOB functions. Based on the data given in Figure 7, it seems that this tuning problem is indeed, as conjectured in the beginning of this section, better represented by the BBOB functions than the problem considered in Section 2. In particular for functions F3 and F15, visual comparison leads us to observe a slightly better fit across various (but not all) feature values.

4 CONCLUSIONS AND OUTLOOK

With this work we aim to trigger a discussion on the suitability of hyper-parameter tuning as benchmarking problems for optimization algorithms. Using two examples stemming from the theory of EA literature we have given some indication that the fitness landscapes described by hyper-parameter tuning may currently not be very well reflected in the 24 BBOB functions. However, we cannot rule out the possibility that performance data from the BBOB test bed can nevertheless provide discriminating information about the algorithms' performances, in particular by combining data across several benchmark functions. For a proper scientific evaluation, we therefore suggest a more detailed, double-sided study on:

(1) whether BBOB performance data can nevertheless be used as training sets to build reasonable performance models for (hyper-)parameter tuning problems, and/or
(2) which parameter tuning problems are particularly appropriate for benchmarking (general-purpose and/or problem-specific) solvers

We recall that an answer to the second question must not ignore the fact that for a reasonable experimental setting the function evaluations of the tuning problem should not be too time-consuming (unless one is interested in the very expensive optimization case, in which one deals with very small evaluation budgets). Finding suitable tuning problems that are *fast to evaluate*, but at the same time also *representative* for "real-world" tuning problems might be a challenging task.

Going forward, we plan on addressing both problems listed above by investigating parameter tuning problems of different types, e.g., exhibiting different dimensionalities, different underlying problems, etc. In concrete terms, a candidate tuning problem that we are particularly keen on understanding is the (ideally automated) configuration of the hyper-parameters of the CMA-ES. We furthermore plan on extending the tuning of its hyper-parameters to setting different CMA-ES modules that have been suggested in the research literature. A suitable framework for such a study is provided by the modular CMA-ES framework available from [45].

One of our long-term objectives is the development of *landscape-aware techniques to control the selection and the configuration of algorithms*, which leverage existing experimental data and appropriate tools from the machine learning literature to select algorithms and parameters *on the fly*. Besides building a suitable training set, such an approach also requires to build efficient predictors for the fitness

(a) dispersion features **(b) ela features**

(c) cell mapping, information content, and nearest better clustering features

Figure 7: Comparison of selected feature values of the "$(1+1)$ EA$_{>0}$ on 250d-LeadingOnes" tuning problem (bold black line) with those of selected or all 24 BBOB functions (slim red lines). All values have been normalized to the interval $[0, 1]$

landscapes. That is, we need to design mechanisms that require only a very limited amount of additional function evaluations for approximating the feature values in the optimization problem's current state. This question comprises three main difficulties: (1) predicting the feature values based on a small number of samples (see [5, 30] for a discussion and first approaches), (2) selecting a subset of features that is large enough for landscape discrimination, while being small enough to reduce the learning complexity [30, 31], and (3) the refinement of feature value computations that are designed to capture the structure of the *global optimization problem* to a *local perspective*, which measures how the relevant part of the search space currently "seen by the algorithm" looks like.

5 PERSPECTIVES

Tuning problems might help to bridge the gap between continuous and discrete evolutionary computation. Independently from the type of sub-problem the search heuristic is solving, the most generic hyper-parameter setting problems involve several parameter types: continuous (e.g., mutation rate), integer (e.g., population size), or qualitative parameters (e.g., choice of modules). Understanding discrete *and* continuous optimization is therefore required for most real-world applications.

We also want to outline the *interplay between machine learning and black-box optimization*: Recent works on landscape-aware heuristics have shown that machine learning can be used to learn a feature-parameter mapping [4, 6, 31]. That alone forms an interesting use case for machine learning practitioners, as the corresponding data set is cleanly generated on-demand and at almost any scale, by sampling a well-defined decision space.

In both domains, the hyper-parameter setting problems are very similar and search heuristics are already applied to solve them both. However, while the landscape-aware approach has already been proven to be useful in the black-box optimization context, we are not aware if similar concepts are in use within the machine learning communities.

ACKNOWLEDGMENTS

Our work is supported by the *European Cooperation in Science and Technology* through COST Action CA15140. C. Doerr acknowledges support from the Paris Ile-de-France Region. P. Kerschke is supported by the *European Research Center for Information Systems (ERCIS)*.

We thankfully acknowledge the research network *COnfiguration and SElection of ALgorithms (COSEAL)* and the participants of the Dagstuhl seminar 16412 on *Automated Algorithm Selection and Configuration* for numerous discussions on the subject of this work.

REFERENCES

[1] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *Proc. of International Conf. on Artificial Intelligence (IJCAI'15)*. AAAI, 733–739.

[2] Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss. 2010. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer.

[3] Thomas Bartz-Beielstein, Oliver Flasch, Patrick Koch, and Wolfgang Konen. 2010. SPOT: A Toolbox for Interactive and Automatic Tuning in the R Environment. In *Proc. of the 20th Workshop Computational Intelligence*. Universitätsverlag Karlsruhe, 264–273.

[4] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2016. Feature Based Algorithm Configuration: A Case Study with Differential Evolution. In *Proc. of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV) (Lecture Notes in Computer Science (LNCS))*, Vol. 9921. Springer, 156 – 166. https://doi.org/10.1007/978-3-319-45823-6_15

[5] Nacim Belkhir, Johann Dreo, Pierre Savéant, and Marc Schoenauer. 2016. Surrogate Assisted Feature Computation for Continuous Problems. In *Proc. of Learning and Intelligent Optimization (LION'16) (Lecture Notes in Computer Science)*, Vol. 10079. Springer, 17–31. https://doi.org/10.1007/978-3-319-50349-3_2

[6] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per Instance Algorithm Configuration of CMA-ES with Limited Budget. In *Proc. of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 681 – 688. https://doi.org/10.1145/3071178.3071343

[7] Jakob Bossek. 2017. smoof: Single- and Multi-Objective Optimization Test Functions. *The R Journal* (2017). https://journal.r-project.org/archive/2017/RJ-2017-004/index.html

[8] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-Heuristics: A Survey of the State of the Art. *Journal of the Operational Research Society* 64, 12 (01 Dec 2013), 1695–1724. https://doi.org/10.1057/jors.2013.71

[9] Nguyen Dang and Carola Doerr. 2019. Hyper-Parameter Tuning for the $(1 + (\lambda, \lambda))$ GA. In *Proc. of the 21st Annual Conference on Genetic and Evolutionary Computation (GECCO'19)*. ACM. https://doi.org/10.1145/3321707.3321725 To appear.

[10] Luc Devroye. 1972. *The Compound Random Search*. Ph.D. Dissertation. Purdue University, West Lafayette, IN, USA.

[11] Laurence Charles Ward Dixon. 1972. The Choice of Step Length, a Crucial Factor in the Performance of Variable Metric Algorithms. *Numerical Methods for Non-Linear Optimization* (1972), 149–170.

[12] Benjamin Doerr and Carola Doerr. 2018. Optimal Static and Self-Adjusting Parameter Choices for the $(1+(\lambda, \lambda))$ Genetic Algorithm. *Algorithmica* 80 (2018), 1658–1709.

[13] Benjamin Doerr and Carola Doerr. 2018. Theory of Parameter Control Mechanisms for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices. In *Theory of Randomized Search Heuristics in Discrete Search Spaces*, Benjamin Doerr and Frank Neumann (Eds.). Springer. To appear. Available online at https://arxiv.org/abs/1804.05650.

[14] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From Black-Box Complexity to Designing New Genetic Algorithms. *Theoretical Computer Science* 567 (2015), 87–104.

[15] Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2019. Self-Adjusting Mutation Rates with Provably Optimal Success Rules. In *Proc. of the 21st Annual Conference on Genetic and Evolutionary Computation (GECCO'19)*. ACM. To appear. Full version is available online at http://arxiv.org/abs/1902.02588.

[16] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. 2013. Mutation Rate Matters Even When Optimizing Monotonic Functions. *Evolutionary Computation* 21 (2013), 1–27.

[17] Carola Doerr and Markus Wagner. 2018. On the Effectiveness of Simple Success-Based Parameter Selection Mechanisms for Two Classical Discrete Black-Box Optimization Benchmark Problems. In *Proc. of the 20th Annual Conference on Genetic and Evolutionary Computation (GECCO'18)*. ACM, 943–950. https://doi.org/10.1145/3205455.3205560

[18] Carola Doerr and Markus Wagner. 2018. Sensitivity of Parameter Control Mechanisms with Respect to Their Initialization. In *International Conference on Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11102. Springer, 360–372. https://doi.org/10.1007/978-3-319-99259-4_29

[19] Matthias Feurer and Frank Hutter. 2019. Hyperparameter Optimization. In *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 3–38.

[20] John J. Grefenstette. 1986. Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. on Systems, Man, and Cybernetics* 16, 1 (1986), 122 – 128.

[21] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[22] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA. https://hal.inria.fr/inria-00362633

[23] Christian Hanster and Pascal Kerschke. 2017. flaccogui: Exploratory Landscape Analysis for Everyone. In *Proc. of the Genetic and Evolutionary Computation Conference Companion (GECCO'17)*. ACM, 1215–1222.

[24] Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. 2006. Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 213 – 228.

[25] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proc. of Learning and Intelligent Optimization (LION'11)*. Springer, 507–523.

[26] Giorgos Karafotias, Mark Hoogendoorn, and Ágoston Endre Eiben. 2015. Parameter Control in Evolutionary Algorithms: Trends and Challenges. *IEEE Transactions on Evolutionary Computation* 19 (2015), 167–187.

[27] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (2019), 3 – 45. https://doi.org/10.1162/evco_a_00242

[28] Pascal Kerschke, Mike Preuss, Carlos Hernández, Oliver Schütze, Jian-Qiao Sun, Christian Grimme, Günter Rudolph, Bernd Bischl, and Heike Trautmann. 2014. Cell Mapping Techniques for Exploratory Landscape Analysis. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Springer, 115–131. https://doi.org/10.1007/978-3-319-07494-8_9

[29] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2015. Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In *Proc. of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO'15)*. ACM, 265–272. https://doi.org/10.1145/2739480.2754642

[30] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. 2016. Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In *Proc. of the 18th Annual Conference on Genetic and Evolutionary Computation (GECCO'16)*. ACM, 229–236.

[31] Pascal Kerschke and Heike Trautmann. 2019. Automated Algorithm Selection on Continuous Black-Box Problems By Combining Exploratory Landscape Analysis and Machine Learning. *Evolutionary Computation* 27, 1 (2019), 99–127. https://doi.org/10.1162/evco_a_00236

[32] Johannes Lengler. 2018. A General Dichotomy of Evolutionary Algorithms on Monotone Functions. In *International Conference on Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11102. Springer, 3–15. https://doi.org/10.1007/978-3-319-99259-4_1

[33] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. 2002. Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions. In *Principles and Practice of Constraint Programming-CP 2002*. Springer, 556–572.

[34] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18 (2017), 185:1–185:52. http://jmlr.org/papers/v18/16-558.html

[35] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz (Eds.). 2007. *Parameter Setting in Evolutionary Algorithms*. Studies in Computational Intelligence, Vol. 54. Springer.

[36] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[37] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proc. of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)*. ACM, 829–836. https://doi.org/10.1145/2001576.2001690

[38] Mario Andrés Muñoz Acosta and Kate Amanda Smith-Miles. 2017. Performance Analysis of Continuous Black-Box Optimization Algorithms via Footprints in Instance Space. *Evolutionary Computation (ECJ)* 25, 4 (2017), 529 – 554. https://doi.org/10.1162/evco_a_00194

[39] Yasha Pushak and Holger H. Hoos. 2018. Algorithm Configuration Landscapes: - More Benign Than Expected?. In *International Conference on Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11102. Springer, 271–283. https://doi.org/10.1007/978-3-319-99259-4_22

[40] François-Michel De Rainville, Christian Gagné, Olivier Teytaud, and Denis Laurendeau. 2012. Evolutionary Optimization of Low-Discrepancy Sequences. *ACM Transactions on Modeling and Computer Simulation* 22 (2012), 9:1–9:25. https://doi.org/10.1145/2133390.2133393

[41] Jérémy Rapin, Marcus Gallagher, Pascal Kerschke, Mike Preuss, and Olivier Teytaud. 2019. Exploring the MLDA Benchmark on the Nevergrad Platform. In *Proc. of the 21st Annual Conference on Genetic and Evolutionary Computation (GECCO'19) Companion*. ACM. to appear.

[42] Jérémy Rapin and Olivier Teytaud. 2018. Nevergrad - A Gradient-Free Optimization Platform. https://GitHub.com/FacebookResearch/Nevergrad. (2018).

[43] Ingo Rechenberg. 1973. *Evolutionsstrategie*. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart.

[44] Michael A. Schumer and Kenneth Steiglitz. 1968. Adaptive Step Size Random Search. *IEEE Transactions on Automatic Control* 13 (1968), 270–276.

[45] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. 2016. Evolving the Structure of Evolution Strategies. In *Proc. of IEEE Symposium Series on Computational Intelligence (SSCI'16)*. IEEE, 1–8. https://doi.org/10.1109/SSCI.2016.7850138