Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case

Vermetten, D.L.; Wang, H.; Bäck, T.H.W.; Doerr, C.

Towards Dynamic Algorithm Selection for Numerical Black-Box Optimization: Investigating BBOB as a Use Case

Diederick Vermetten
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Hao Wang
Sorbonne Université, CNRS, LIP6
Paris, France

Thomas Bäck
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

One of the most challenging problems in evolutionary computation is to select from its family of diverse solvers one that performs well on a given problem. This algorithm selection problem is complicated by the fact that different phases of the optimization process require different search behavior. While this can partly be controlled by the algorithm itself, there exist large differences between algorithm performance. It can therefore be beneficial to swap the configuration or even the entire algorithm during the run. Long deemed impractical, recent advances in Machine Learning and in exploratory landscape analysis give hope that this dynamic algorithm configuration (dynAC) can eventually be solved by automatically trained configuration schedules. With this work we aim at promoting research on dynAC, by introducing a simpler variant that focuses only on switching between different algorithms, not configurations. Using the rich data from the Black Box Optimization Benchmark (BBOB) platform, we show that even single-switch dynamic Algorithm selection (dynAS) can potentially result in significant performance gains. We also discuss key challenges in dynAS, and argue that the BBOB-framework can become a useful tool in overcoming these.

CCS CONCEPTS

•**Theory of computation** → *Bio-inspired optimization; Online algorithms;*

1 INTRODUCTION

It is well known that, when solving an optimization problem, different stages of the process require different search behavior. For example, while exploration is needed in the initial phases, the algorithm needs to eventually converge to a solution (exploitation). State-of-the-art optimization algorithms therefore often incorporate mechanisms to adjust their search behavior *while optimizing*, by taking into account the information obtained during the run. These techniques are studied under many different umbrellas, such as *parameter control* [10], *meta-heuristics* [5], adaptive operator selection [24], or *hyper-heuristics* [6]. The probably best-known and most widely used techniques for achieving a dynamic search behavior are the *one-fifth success rule* [7, 32, 33] and the *covariance adaptation technique* that the family of CMA-ES algorithms [14, 15] is build upon. While each of these two control mechanisms tackles the problem of balancing performance in different phases of the search in its own way, they are mostly working with a specific algorithm, aiming to tune its performance by changing internal parameters or algorithm modules. This inherently limits the potential of these methods, since different algorithms can have widely varying performances during different phases of the optimization process. By switching between these algorithms during the search, these differences could potentially be exploited to get even better performance. We coin the problem of choosing which algorithms to switch between, and under which circumstances, the *Dynamic Algorithm Selection* (dynAS) problem.

Solving the dynAS problem would be an important milestone towards tackling the more general *dynamic Algorithm Configuration (dynAC)* problem, which also addresses the problem of selecting (and possibly adjusting) suitable algorithm configurations. Specifically, dynAS is limited to switching between algorithms from a discrete portfolio of pre-configured heuristics, whereas for dynAC, the algorithms come with (possibly several) parameters whose settings can have significant influence on the performance.

We do not solve dynAS here, but aim to show its potential for numerical optimization. We then aim to develop suitable environments to encourage and enable future research into achieving the identified potential of dynAS and, in the longer run, to extend this to the dynAC problem. As a first step, we need to identify a meaningful collection of algorithms and benchmark problems, which together cover the main characteristics and challenges of the dynAS problem, without imposing too many additional challenges. The Black-Box Optimization Benchmarking (BBOB) environment [12] with its rich data sets available at [1] suggests itself as a natural starting point for such considerations, since the community has already acquired a quite solid understanding of the problems and solvers in this test-bed over the last decade.

We perform a first assessment of the performance that one could expect to see when applying dynAS to the algorithms in the BBOB data sets, to understand whether the gains would justify further exploration of the dynAS paradigm on this test-bed. We find that – even when restricting the dynAS problem further to allowing only a single switch between algorithms in the portfolio – promising improvements over the best static solvers can be expected, in particular for the more complex problems (functions 19-24).

Our considerations are purely based on a theoretical investigation of the potential, which might be too optimistic for the single-switch dynAS case – most importantly, because of the problem

GECCO '20, Cancn, Mexico
2020. 978-1-4503-7128-5/20/07…$15.00
DOI: 10.1145/3377930.3390189

of *warm-staring* the algorithms: since the heuristics are adaptive themselves, their states need to be initialized appropriately at the switch. This may be a difficult problem when changing between algorithms of very different structure. We do not consider, on the other hand, the possibility to switch more than once, so that our bounds may be too too pessimistic for the full dynAS setting, in which an arbitrary number of switches is allowed.

Given the above limitations, we therefore also provide a critical assessment of our approach, and highlight ideas for addressing the main challenges in dynAS.

1.1 Related Work

The idea that a dynamic configurations and/or selection of algorithms can be beneficial in the context of iterative optimization heuristics is almost as old as evolutionary computation itself, in particular in the context of solving numerical optimization problems, see [20] for an entire book focusing mostly on dynamic algorithm configuration techniques. However, as mentioned above, existing works almost exclusively focus on changing parameters of selected components of an otherwise stable algorithmic framework. This includes most works on hyper-heuristics [6] and related concepts such as adaptive operator selection [24], and parameter control [10].

To the best of our knowledge, the full dynAC problem as described above was only recently formalized [4]. Biedenkamp *et al.* introduce dynAC as a Contextual Markov Decision Process (CMDP), where a policy can be learned to switch hyperparameters of a meta-algorithm, with some of these hyperparameters possibly encoding the choice between different algorithms.[1] They also show that artificial CMDPs can be solved effectively by using reinforcement learning techniques, providing a promising direction for future research on dynAC.

In the context of evolutionary computation, the concept of switching between different algorithms during the optimization process was recently investigated in [35], by a similar theoretical assessment as in this work. The approach was then tested in [37], where it was shown that the predicted gains can indeed materialize, with the caveat that one has to ensure a sufficiently accurate estimate for the median anytime performances of each algorithm. These two works, however, focus on a single family of numerical black-box optimization techniques, the modular CMA-ES framework suggested in [36]. Here in this work, in contrast, we explicitly want to go one step further, and study combinations of heuristics that are potentially of very different structure, such as, for example combining a Differential Evolution (DE) algorithm for the global exploration with a CMA-ES for the final convergence.

While the dynAC problem is solved by an unsupervised reinforcement learning approach in [4], we observe that dynAC in evolutionary computation is more frequently based on on supervised learning approaches, see [16, 23, 27] for examples. These techniques combine exploratory landscape analysis [25] and/or fitness landscape analysis [31] with supervised learning techniques, such as random forests, support vector machines, etc. While still in its infancy, even in the static algorithm configuration case [3, 17, 18, 28], these works may pave an interesting alternative to reinforcement learning, as they may more directly provide insight into (and make use of) the correlation between fitness landscapes and algorithms' performance.

[1] Note here that there is a long-standing debate about the classification of algorithm configuration vs. algorithm selection. That is, while some consider a parametrized algorithm framework an algorithm with different configurations, others argue that each such configuration is an algorithm by itself. We omit this discussion here, and use the convention that an algorithm can have possibly different configurations. Note, though, that – in the context of this work – this only makes a difference in the terminology. All concepts and ideas can be equivalently described using the other, possibly mathematically more stringent, convention.

2 PRELIMINARIES

2.1 Dynamic Algorithm Selection

Classically, algorithm selection attempts to find the best algorithm A from a portfolio $\mathcal{A}$ to solve a specific function f from a set of functions $\mathcal{F}$. Specifically, this static version of algorithm selection can be defined as follows:

Definition 2.1 (Static Algorithm Selection). Given an algorithm portfolio $\mathcal{A}$ and a function $f \in \mathcal{F}$, we aim to find:

$$\arg\min_{A \in \mathcal{A}} \text{PERF}(A, f) \,,$$

where PERF is a performance measure (which assigns lower values to better performing algorithms).

To extend algorithm selection to the dynamical case, we need to define a function which switches between algorithms. We use techniques from [4] to represent this as a policy function, and modify it as follows:

Definition 2.2 (Dynamic Algorithm Selection (dynAS)). Given an algorithm portfolio $\mathcal{A}$, a $f \in \mathcal{F}$ and a state description $s_t \in \mathbb{S}$ at time step t of an algorithm run. We want to find a policy $\pi : \mathbb{S} \to \mathcal{A}$ which minimizes $\text{PERF}(A_\pi, f)$

Note that this definition can be extended to dynamic algorithm configuration by changing the policy to be $\pi : \mathbb{S} \to (\mathcal{A} \times \Theta_A)$, where Θ_A is the configuration space of algorithm A.

2.2 The BBOB Benchmark

The Black Box Optimization Benchmark (BBOB) is widely accepted as the go-to benchmarking framework within the field of optimization. While BBOB has grown a lot over the years, the functions within their noiseless suite have remained stable. This suite contains 24 noiseless optimization functions, each of which being theoretically defined for any number of dimensions. In practice however, the commonly used dimension set is $\mathcal{D} = \{2, 3, 5, 10, 20, 40\}$. For each function, several transformation methods are defined, both for the variable as the objective spaces. These transformations are fixed, and different combinations lead to different versions of the function, called instances. Since these functions are defined mathematically, the optimal values are known in advance. Because of this, we can define target values we wish to reach in terms of closeness to this optimal value, instead of an abstract value. This gives the advantage of comparability between instances, which would not be possible when using raw target values.

The 24 noiseless functions have been studied in detail, not just from a performance perspective. Especially within the landscape analysis community, a lot of analysis of the BBOB-functions has

been performed, leading to a lot of useful insights about their properties. These properties are ideal to use when implementing dynAS in practice, as they are very influential on the local performance of algorithms. Generally, it is agreed that the 24 BBOB functions cover a broad range of potential challenges for different optimization algorithms [25], even though certain aspects, i.e., discontinuities or plateaus, are not very well represented [19].

The popularity of BBOB means that many researchers have benchmarked their algorithms on the BBOB-functions. Most of these have then submitted versions of their algorithms to competitions or workshops organized by the BBOB-team. Between the first competition in 2009 [13] and the latest workshop in 2019, a total of 226 algorithms have been submitted and their data made available to the public [1]. Because of this large amount of available data, there are plenty of baselines to compare algorithms against and gain inspiration from. These algorithms have often been well justified and rigorously tested. However, the implementations used are generally not freely available, and even if they are, they might be hard to combine into a single dynAS framework, since BBOB is available in many different languages. However, the majority of the algorithms is either directly available online or has been well-documented, making the challenge of implementing them doable.

Additionally, the large amount of algorithms which have been run on BBOB provide a good way to select sets of algorithms from which to build initial dynAS portfolios. However, since the BBOB-repository is largely the result from running competitions, many of the used algorithms are highly tuned, making them hard to beat and giving rise to the question of generalizability of dynAS results to other functions. Eventually, a move to true dynAC would resolve this issue, but these techniques will require a lot of further study to implement.

Since the BBOB-framework provides the functions, algorithms and performance baselines, it is an ideal candidate for initial experiments related to dynAS.

2.3 Performance Measures

To measure the performance of the algorithms on the BBOB-dataset, several approaches are possible. These usually fall into two categories: fixed-budget and fixed-target. The fixed-budget approach asks the question: "What target value is reached after x function evaluations?", while the fixed-target question can be phrased as: "How many function evaluations are needed to reach target y?".

In this paper, we will use the fixed-target approach. Since most algorithms in our data set are stochastic in nature, the question of how many function evaluations are needed to reach a certain target is dealing with random variables. For a certain function instance $f_i \in \mathcal{F}$ and dimension $d \in \mathcal{D}$, we let $t_j(A, f_i^{(d)}, \phi)$ denote the number of evaluations that algorithm $A \in \mathcal{A}$ needed in the j-th run to evaluate for the first time a point of target precision at least ϕ. Note that $t_j(A, f_i^{(d)}, \phi)$ is a random variable, which is commonly referred to as the *Hitting Time (HT)*. If run j did not manage to hit target ϕ within its allocated budget, we say that $t_j(A, f_i^{(d)}, \phi) = \infty$.

While just taking the average of the observed hitting time gives some estimate of the true mean, previous work [2] has shown that it is not a consistent, unbiased estimator of the mean of the distribution of hitting times. Instead, the Expected Running Time (ERT) is used. This is defined as follows:

Definition 2.3 (Expected Running Time (ERT)).

$$\mathrm{ERT}(A, f^{(d)}, \phi) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{K} \min\{t_i(A, f_j^{(d)}, \phi), B\}}{\sum_{i=1}^{n} \sum_{j=1}^{K} \mathbb{1}\{t_i(A, f_j^{(d)}, \phi) < \infty\}}.$$

Here, n is the number of runs of the algorithm, K the number of instances of function f and B the maximum budget for algorithm A on function $f_j^{(d)}$.

To allow for a fair comparison between instances, the BBOB-benchmark uses target 'precisions' for their analysis, instead of the raw target values seen by the algorithm. The precision is simply defined as the difference between the best-so-far-$f(x)$ and the global optimum. This is done to make runtime comparisons between different instances and even different functions possible.

3 METHODS

3.1 Analysis of Available data

Since the set of available algorithms from the BBOB-competitions is quite large, several issues in terms of data consistency arise. When processing the algorithms, we found that a small subset have issues such as incomplete files or missing data. We decided to ignore these algorithms, and work only with the ones which were made available within the IOHanalyzer tool [9]. This leaves us with a set of 182 out of 226 possible algorithms to do our analysis.

There are some caveats to this data, mostly related to the lack of a consistent policy for submission to the competitions over the years. For example, the 2009 competition required submission of 3 runs on 5 instances each, while the 2010 version changed this to 1 run on 15 instances. In theory, the instances should have very little impact on the performance of the algorithms, as they are selected in such a way to preserve the characteristics of the functions. However, in practice there has been some debate about the impact of instances on algorithm performance, claiming that the landscapes of different instances of the same function can look significantly different to an algorithm [18, 26, 29]. In the following, we ignore this discussion and assume that performance is not significantly impacted by the instances.

Another issue with the dataset are the widely inconsistent budgets for the different algorithms. These can be as low as $50D$ and as large as $10^7 D$. However, since we use a fixed-target perspective to study the performance of the algorithms, these differences are not very impactful.

Since the BBOB-competitions see an optimizer as having 'solved' an optimization problem when reaching a target precision of 10^{-8}, many of the algorithms will stop their runs after reaching this point to avoid unnecessary computation. Because of this, we will use the same target value in our computations. However, for some of the more difficult functions, this target can be challenging to reach within their budget. To avoid the problem of dealing with algorithms without any finished runs, we only consider an algorithm in our analysis when it has at least 15 runs on the function, of which at least one managed to reach the target 10^{-8}. Figure 1 plots the number of algorithms per each function/dimension pair

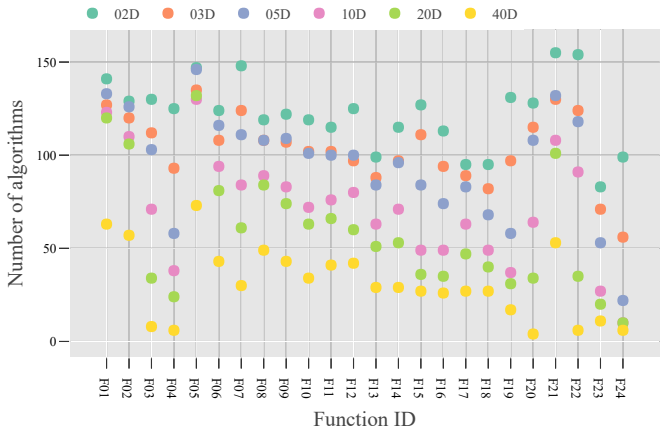

Figure 1: Number of algorithms with at least 15 independent runs and at least one them reaching the target $\phi = 10^{-8}$.

that satisfy all the requirements mentioned above. We observe large discrepancies between functions and dimensions, with the number of admissible algorithms ranging from 4 to 155, and note that there are no algorithms which are admissible on all functions in all dimensions.

3.2 DynAS for BBOB-Functions

In this work, we will restrict the dynAS problem on BBOB-functions to using policies which switch algorithms based on the target precisions hit. To get an indication for the amount of improvement which can be gained by dynAC over static algorithm configuration, we use the BBOB-data to theoretically simulate a simple policy which only implements a single switch of algorithm. We can define this as follows:

Definition 3.1 (Single-Switch dynAS). Let $f^{(d)}$ be a BBOB-function in dimension d and $\mathcal{A}$ the corresponding portfolio of admissible algorithms. A single-split policy is defined as the triple $(A_1, A_2, \tau) \in \mathcal{A} \times \mathcal{A} \times \Phi$, where $\Phi = \left\{10^{2-0.2i)} | i \in \{0, \dots, 50\}\right\}$ is the set of admissible splitpoints. This corresponds to the policy which starts the optimization procedure with algorithm A_1, and run this until target τ is reached, after which the algorithm is changed to A_2.

The performance of this single switch method can then be calculated as follows:

$$T(f^{(d)}, A_1, A_2, \tau, \phi) = \mathrm{ERT}(A_1, f^{(d)}, \tau) + \mathrm{ERT}(A_2, f^{(d)}, \phi) - \mathrm{ERT}(A_2, f^{(d)}, \tau)$$

Where ϕ is the final target precision we want to reach. For the BBOB-functions, we set $\phi = 10^{-8}$, as noted in Section 3.1.

Generally, to assess the performance of an *algorithm selection* method, its performance can be compared to the *Single Best Solver (SBS)*, which can be defined as follows:

Definition 3.2 (Single Best Solver). For each dimension $d \in \mathcal{D}$, we have:

$$\mathrm{SBS}_{\mathrm{static}}(\mathcal{F}^{(d)}) = \arg\min_{A \in \mathcal{A}} \sum_{f \in \mathcal{F}} \mathrm{PERF}(A, f^{(d)}, \phi)$$

Often, ERT is used as the performance function, but this value can differ widely between functions, leading to a biased weighting. To avoid this, we can instead use the ranking of ERT per function, to give equal importance to every function. Note that we have final target precision $\phi = 10^{-8}$.

While this SBS has a good average performance, it can easily be beaten by a decent *algorithm selection* technique. As such, a better baseline for performance is needed. This is the theoretically best algorithm selection method, which is called the Virtual Best Solver. This can defined as follows:

Definition 3.3 (Static Virtual Best Solver (VBS_{static})). For each function $f \in \mathcal{F}$ and dimension $d \in \mathcal{D}$, we have:

$$\mathrm{VBS}_{\mathrm{static}}(f^{(d)}) = \arg\min_{A \in \mathcal{A}} \mathrm{PERF}(A, f^{(d)})$$

For the BBOB functions, we use $\mathrm{PERF}(A, f^{(d)}) = \mathrm{ERT}(A, f^{(d)}, \phi)$ with $\phi = 10^{-8}$.

Note that the $\mathrm{VBS}_{\mathrm{static}}$ will always perform at least as good as the SBS, and theoretically gives an upper bound for the performance of any real implementation of algorithm selection techniques. Thus, the difference between SBS and $\mathrm{VBS}_{\mathrm{static}}$ gives an indication of the maximal possible performance gained by algorithm selection. For the BBOB-data, the relative ERT between these two methods is visualized in Figure 2. From this, we see that the differences can be extremely large, highlighting the importance of algorithm selection.

Similar to the way we defined $\mathrm{VBS}_{\mathrm{static}}$, we can define a Dynamic Virtual Best Solver, $\mathrm{VBS}_{\mathrm{dyn}}$, as follows:

Definition 3.4 (Dynamic Virtual Best Solver). For each BBOB-function $f \in \mathcal{F}$ and dimension $d \in \mathcal{D}$, we have:

$$\mathrm{VBS}_{\mathrm{dyn}}(f^{(d)}) = \arg\min_{(A_1, A_2, \tau) \in (\mathcal{A} \times \mathcal{A} \times \Phi)} T(f^{(d)}, A_1, A_2, \tau, \phi)$$

4 RESULTS

Since the number of algorithms considered in this paper is relatively large, many of the results are only shown for a subset of functions, dimensions or algorithms. The complete data is made available at [38]. An example of the available data is also shown in Table 1.

4.1 Overall Gain of Single-Switch DynAS

Before investigating the possible improvements to be gained by dynamic algorithm selection, we investigate the performance of the static algorithms from the BBOB-dataset. To achieve this, we look at the distribution of ERTs among the BBOB-functions. For dimension 5, this is visualized in Figure 3.[2] This figure shows the large differences in performance, both between the algorithms as well as between the different functions. We marked the performance of the $\mathrm{VBS}_{\mathrm{static}}$ and $\mathrm{VBS}_{\mathrm{dyn}}$, and see that their differences also vary largely between functions.

To zoom in on the differences between the $\mathrm{VBS}_{\mathrm{static}}$ and $\mathrm{VBS}_{\mathrm{dyn}}$ we see in Figure 3, we can compute for each function, dimension and corresponding algorithm portfolio the relative ERT of a the

[2]Note that for function F05, the linear slope, most algorithms simply move outside the search-space to find an optimal solution, which is accepted by the BBOB-competitions, but leads to a disadvantage to those algorithms which respect the bounds.

FID	VBS_{static}	ERT of VBS_{static}	A_1	A_2	$\log_{10}(\tau)$	ERT of VBS_{dyn}	speedup
1	fminunc	13.0	HMLSL	HCMA	1.2	6.6	1.97
2	LSfminbnd	94.7	BrentSTEPrr	LSfminbnd	2.0	52.4	1.81
3	BrentSTEPrr	315.5	STEPrr	BrentSTEPif	-0.2	246.8	1.28
4	BrentSTEPif	763.9	STEPrr	BrentSTEPif	-0.2	578.1	1.32
5	MCS	10.8	ALPS	MCS	1.8	6.0	1.80
6	MLSL	1050.9	fmincon	GLOBAL	-7.0	928.2	1.13
7	PSA-CMA-ES	1129.8	GP5-CMAES	PSA-CMA-ES	0.0	792.3	1.43
8	fminunc	399.1	OQNLP	DE-BFGS	0.6	304.7	1.31
9	fminunc	188.3	fminunc	DE-AUTO	0.0	152.3	1.24
10	DTS-CMA-ES	262.4	fmincon	DTS-CMA-ES	-2.0	199.8	1.31
11	DTS-CMA-ES	268.3	HMLSL	DTS-CMA-ES	-2.2	153.6	1.75
12	NELDERDOERR	1909.7	HMLSL	BFGS-P-StPt	-3.2	1041.5	1.83
13	IPOPsaACM	835.1	DE-AUTO	IPOPsaACM	-3.6	661.7	1.26
14	DTS-CMA-ES	546.6	DE-BFGS	DE-SIMPLEX	-6.0	348.6	1.57
15	PSA-CMA-ES	10029.7	LHD-10xDefault-MATSuMoTo	PSA-CMA-ES	0.4	6982.4	1.44
16	IPOPsaACM	6767.1	GLOBAL	CMA-ES-TPA	-0.4	5115.0	1.32
17	PSA-CMA-ES	4862.3	PSA-CMA-ES	IPOP400D	-5.8	4201.8	1.16
18	PSA-CMA-ES	6717.4	PSA-CMA-ES	CMA-ES multistart	-5.2	5687.3	1.18
19	DTS-CMA-ES	18768.0	OQNLP	DTS-CMA-ES	-1.6	463.0	40.54
20	DEctpb	10670.3	DEctpb	OQNLP	-0.4	3360.7	3.18
21	GLOBAL	2095.5	MLSL	NELDERDOERR	0.0	1209.8	1.73
22	GLOBAL	1079.9	RAND-2xDefault-MATSuMoTo	GLOBAL	0.4	844.1	1.28
23	CMA-ES-MSR	18971.4	DTS-CMA-ES	SSEABC	-2.6	10295.0	1.84
24	OQNLP	285173.0	GP5-CMAES	CMAES-APOP-Var2	0.0	52387.0	5.44

Table 1: Relative gain of optimal single-switch dynamic algorithm combination VBS_{dyn} over the best static algorithm VBS_{static} for all 24 BBOB functions in dimension 5. ERT values are computed from data available at https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob. We only consider algorithms with at least 15 runs, one of which reaching target precision $\phi = 10^{-8}$, which is also the target used for the ERT calculations. The full version of this table, also for other dimensions, is available at [38]. Abbreviations: FID = function ID (as in [12], τ = splitpoint target, speedup = ERT_stat/ERT_dyn. We also shortened DTS-CMA-ES_005-2pop_v26_1model to DTS-CMA-ES for readability

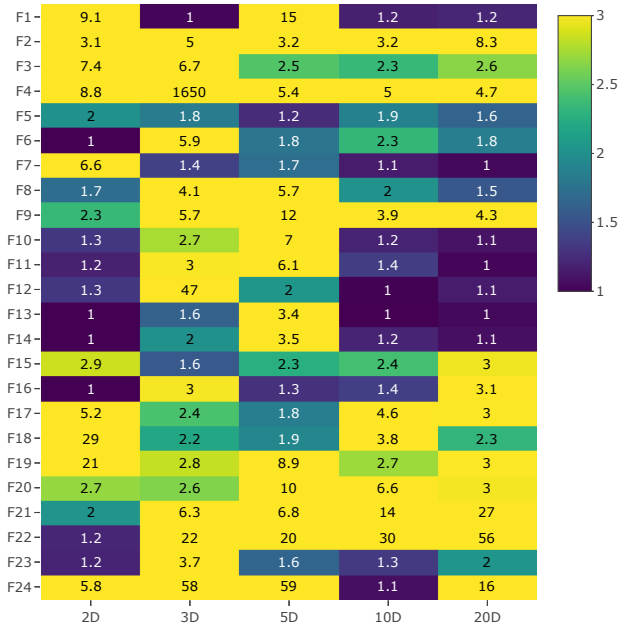

Figure 2: Relative ERT of the SBS over the VBS_{static}. The selected SBS are: Nelder-Doerr (2D), HCMA(3, 10 and 20D) and BIPOP-aCMA-STEP (5D). Dimension 40 was removed because no algorithm hit the final target on all functions in this dimension.

Single-Switch VBS_{dyn} over VBS_{static}. Specifically, this is calculated as $\frac{\text{ERT}(\text{VBS}_{\text{dynamic}}(f^{(d)}))}{\text{ERT}(\text{VBS}_{\text{static}}(f^{(d)}))}$. This value is shown for each (function, dimension)-pair in Figure 4. From this figure, we can see that for most functions, the improvements when using a single configuration change are quite large. Especially for the functions which are traditionally considered more difficult for a black-box optimization

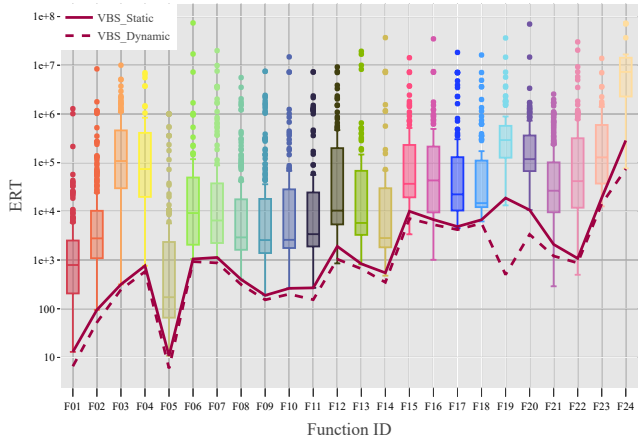

Figure 3: Distribution of ERTs among all algorithms for all 24 BBOB-functions in dimension 5. Please recall from Fig. 1 that the number of data points varies between functions. Also shown are the ERTs of the VBS_{static} and VBS_{dyn}.

algorithm to solve, the possible improvement is massive. In terms of the median over all (function, dimension)-pairs, the VBS_{dyn} is 1.49 faster than the VBS_{static}.

4.2 Selected Algorithm Combinations

Since the VBS_{dyn} shows a lot of potential improvement over the classical VBS_{static}, it makes sense to study its behaviour in more detail. To achieve this, we can zoom in on a single (function, dimension)-pair and study the behaviour of the VBS_{dyn} and split algorithm configurations in general. In Figure 5, we show the ERT of the best possible switch between any combination of algorithms in our portfolio $\mathcal{A}$, on function 21 in dimension 10. This figure shows some clear patterns in the horizontal and vertical lines. A horizontal line, such as the one for the MLSL-algorithm [21], indicates that an algorithm adds to the performance of most algorithms by being the

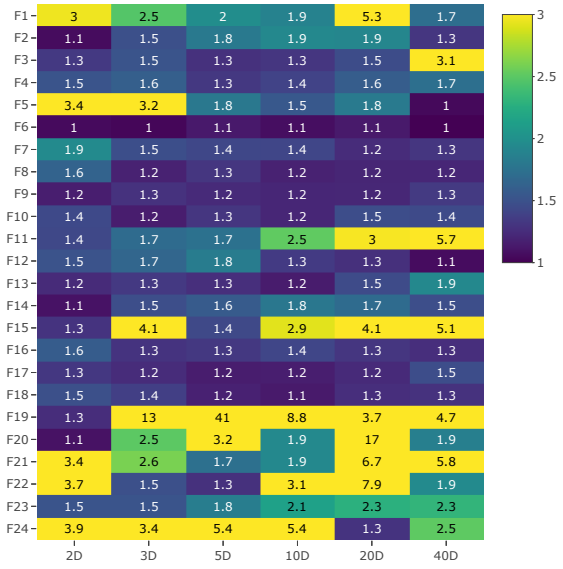

Figure 4: Heatmap of the ratio of ERTs between the Virtual Best Static Solver and the Virtual Best Dynamic Solver, for each (function, dimension)-pair.

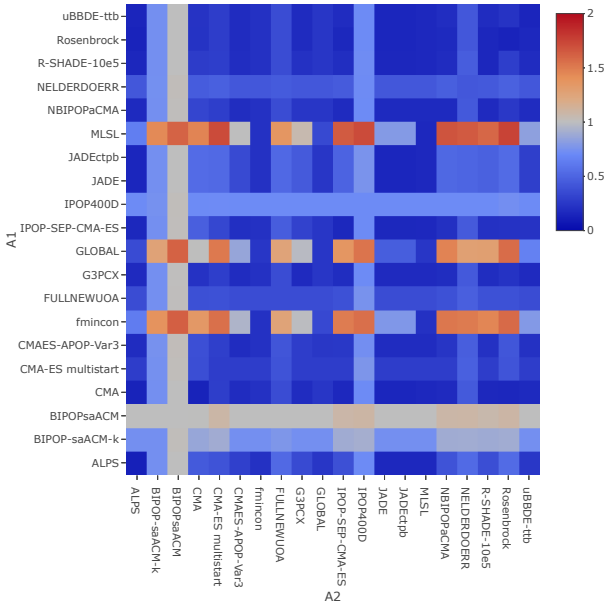

Figure 5: Relative ERT of configuration switches relative to $\text{VBS}_{\text{static}}$, for function 21 in 10 dimensions. The X- and Y-axes indicate algorithms selected as A_2 and A_1 respectively. Larger values (red) indicate better algorithm combinations.

A_1-algorithm. This can be interpreted as having a good exploratory search behaviour, but poor exploitation. There are also vertical lines present, which indicate the algorithms which perform well as A_2-algorithms. These are less pronounced than the horizontal lines, which might indicate that the choice of A_2 algorithms has less impact on the performance than the choice of A_1.

We see that there are different algorithms which perform well as either the first or second part of the search. This gives rise to the question of how to quantify these differences, and more generally, how to quantify the benefit which can be gained by selecting an algorithm as A_1 or A_2. This can be done by executing the following steps to compute a quantitative value for the benefit gained by selecting an algorithm for a part of the search:

Definition 4.1 (Improvement-values). The initial performance value I_1 and finishing performance value I_2 of algorithm A on function $f^{(d)}$ can be defined as:

$$I_1(A) = \frac{\min_{A_2 \in \mathcal{A}, \tau \in \Phi} T(A, A_2, \tau, \phi)}{\min_{A_1, A_2 \in \mathcal{A}, \tau \in \Phi} T(A_1, A_2, \tau, \phi)}$$

$$I_2(A) = \frac{\min_{A_1 \in \mathcal{A}, \tau \in \Phi} T(A_1, A, \tau, \phi)}{\min_{A_1, A_2 \in \mathcal{A}, \tau \in \Phi} T(A_1, A_2, \tau, \phi)}$$

Note that for the $\text{VBS}_{\text{dyn}} = (A_1, A_2, \tau)$, we always have $I_1(A_1) = 1 = I_2(A_2)$, and values can not be below 1. Intuitively, the larger the value of I_1, the worse the algorithm can perform as the first part of the search, and similarly for I_2.

The values of I_1 and I_2 for dimension 5 are shown in Figures 6 and 7 respectively. To ensure the readability of the figures, only a subset of algorithms is chosen. This is done by selecting the algorithm with the best value for each function, and then adding to it the set of algorithms which have the best average value over all functions[3]. From these figures, we see clear differences, both between functions and between algorithms. While some algorithms occur in both Figures 6 and 7, many are included only once, indicating that they are relatively good choices for one part of the search, but not the remainder. The clearest example of this is HMLSL [30], which performs very well as A_1, but has relatively high I_2-values. This is caused by the fact that this algorithm typically converges quickly to a value close to the optimum, but has issues in the final exploitation phase, thus only being beneficial to use at the start of the search. We also notice that in general, the I_2-values are much lower across all algorithms, indicating that the choice of starting algorithm is the most important for dynAS, while most good algorithms can provide similar benefits to the final part of the search.

4.3 Small Portfolio: Case Study

Since the algorithm space we consider is quite large, it can be challenging to gain insights into the individual algorithms. To show that dynamic algorithm selection is also applicable to smaller portfolio's, we limit ourselves to 5 algorithms. These are representative of some widely used algorithm families: Nelder-Doerr [8], DE-Auto [40], Bipop-aCMA-Step [22], HMLSL [30] and PSO-BFGS [41].With this reduced algorithm portfolio, we can study the improvements over their respective $\text{VBS}_{\text{static}}$ in more detail, and find interesting algorithms combinations to explore further.

In Figure 8, we show the relative improvement in ERT over $\text{VBS}_{\text{static}}$ of the best combination of two algorithms. In each subplot, all 24 functions are represented. Note that the diagonal represents the static algorithms, which can never lead to an improvement over the $\text{VBS}_{\text{static}}$. We notice some clear trends in this figure. Specifically,

[3]Missing values and values larger than 3 are set to 3 to reduce the large impact of outliers on the average.

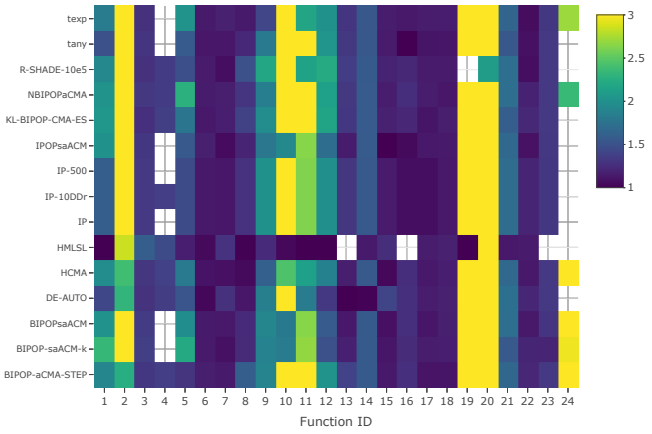

Figure 6: I_1-values for a group of 15 selected algorithms in dimension 5. Darker colors correspond to better values.

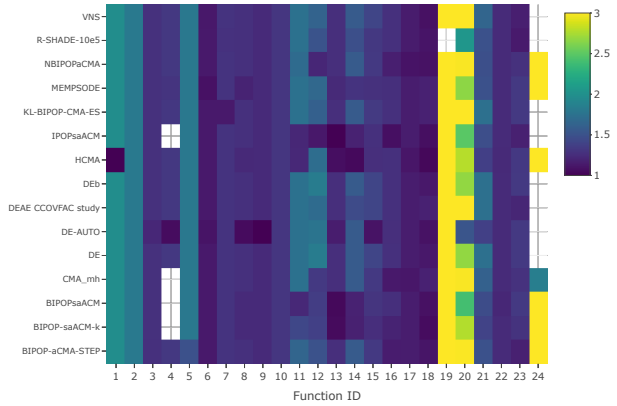

Figure 7: I_2-values for a group of 15 selected algorithms in dimension 5. Darker colors correspond to better values.

we notice that using HMSLS as A_2 is rarely effective, while it provides large benefits when used in the initial part of the search. We also note that Nelder-Doerr has the reverse behaviour, seemingly performing much better in the final exploitation phase.

To illustrate the configuration switches which can be considered in this algorithm portfolio, we can zoom in on function 12 in dimension 3 and look at the fixed-target curve showing ERT. This is done in Figure 9, where we also indicate the best switching points between algorithms. This figure highlights the different behaviors of the algorithms in the portfolio, and thus indicates where switching algorithms would be beneficial. The best possible switch in this function would occur from PSO-BFGS to Nelder-Doerr, at target $10^{-6.4}$, leading to a relative speedup of 1.76 over $\text{VBS}_{\text{static}}$.

To decide which algorithms to use in an algorithm portfolio such as the one used here, two main ways of selecting the algorithms are possible. The first is to use some knowledge about the algorithms to determine which are important. This is useful for initial exploration, but might lead to useful algorithms being ignored. Instead, one can use performance information, such as the I_1 and I_2-values, to provide some initial representation of the usefulness of algorithms to the portfolio. This approach is much more generic, however the

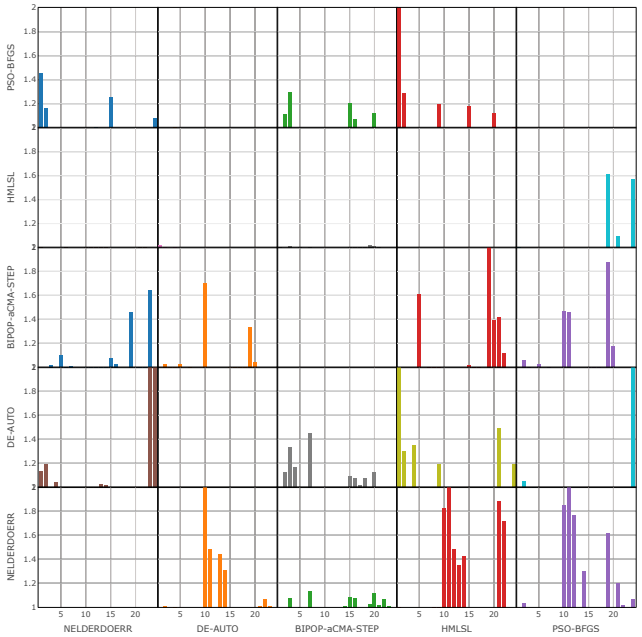

Figure 8: Overview of the best possible ERTs of the combination of algorithms A_1 and A_2 over $\text{VBS}_{\text{static}}$. Each plot represents a single A_1 (X-axis), A_2 (Y-axis) combination, where each bar represents a single function, in dimension 3. Values are capped at 2.

choice of measures can be challenging. For example, the I_1 and I_2 measures are hard to extend to more general k-switch dynAS methods. Instead, an extension of marginal contributions [42] and related concepts such as measures building on Shapley values (like those suggested in [11]) would capture algorithm contribution to a portfolio in a much more robust sense, and thus be useful additions to the dynAS setting.

5 DISCUSSION AND FUTURE WORK

Summary. The previous results have shown that there is still a large amount of improvement possible over the $\text{VBS}_{\text{static}}$ by using dynamic algorithm selection. We have shown several methods to gain insights into the differences between different algorithms and functions. However, the results shown in the previous sections rely on an underlying assumption of feasibility of algorithm switching. For many algorithms, this switching mechanism can be implemented in a relatively straightforward manner, i.e. between different population-based algorithms, such as different CMA-ES variants, for which the algorithm switching methods have already been implemented [37].

Warm-start. For other algorithms combinations, a dynamic switch during the optimization procedure might be more challenging. For example, a switch from a single-solution algorithm to a population-based one gives rise to an information deficit, which needs to be dealt with to properly initialize the new population. Because of this, the gains indicated by simply combining the ERT values might be tough to achieve in practice.

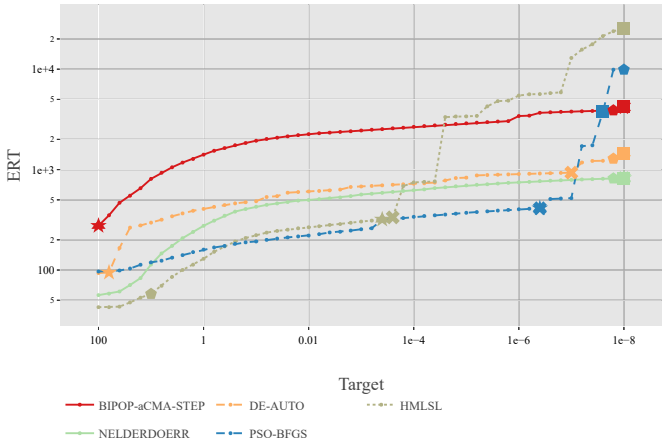

Figure 9: ERT-curves for a selected algorithm portfolio of size 5 on F12 in 3D. Markers indicate optimal switch points between algorithms. Their color and symbol indicate the starting and finishing algorithms respectively.(star = Nelder-Doerr, triangle = DE-AUTO, cross = BIPOP-aCMA-STEP, square = HMLSL and pentagon = PSO-BFGS).

More generally, internal parameters are different between algorithms. So the first challenge to overcome is that one needs to decide how to "warm-start" the algorithms, to assure an optimal internal state for the required phase of the optimization process. To be able to achieve the performance of the VBS_{dyn}, such warm-start techniques will need to be implemented without the need of additional function evaluations, which could be a big challenge. We would considering to use reinforcement learning approaches to be a promising first step for this task, but since those are quite expensive in terms of computational cost, we hope to see other approaches evolve in the near future.

Stochasticity. Assuming such warm-start mechanisms are implemented, as was previously done for example within CMA-ES, it has been shown that the theoretical improvements can still be tough to achieve in practice [37]. This is largely caused by the fact that hitting times are stochastic with relatively large variances, which can cause ERT to be unstable. When selecting the (A_1, A_2, τ)-triple, differences in ERT might be obscured by the variance of the hitting times, leading to a worse performance than expected. These effects might become even more important when dealing with larger algorithm spaces, or when incorporating hyperparameters in the search (see paragraph *Hyperparameter tuning*). Analyzing the robustness of common solvers therefore seems to be an essential building block for the development of reliable dynAC approaches.

Switch point. Another challenge which needs to be overcome to achieve effective dynamic algorithm selection is the question how to identify suitable switching points. In this work we used target precision, which is usually not applicable in practice, since the algorithm has no knowledge about the precise value of the optimum. Because of this, we would need to find some other way to use the knowledge of the algorithm to determine when to switch, i.e., the state of internal parameters, landscape features computed from additionally or previously evaluated points, the evolution of fitness values, population diversity, etc.

True dynamic switching. While improving the way a switching point is detected is a big challenge to overcome, it also provides new opportunities to improve performance. The estimates shown in this paper consider only a single algorithm switch, whereas a truly dynamic approach could benefit from switching more often, to fully exploit the differences in search behaviour of the different algorithms.

Hyperparameter tuning. A second factor of improvement can come from adding hyperparameter tuning into the dynamic process; i.e., when moving from the algorithm selection setting to a dynamic variant of *Combined Algorithm Selection and Hyperparameter optimization* (CASH [34, 39]). A dynamic CASH approach would allow the algorithms to specialize even more, so they can focus even more on performing as good as possible on their specific part of the optimization process.

Extensions. As any benchmark study, our results are – for the time being – limited to the 24 noiseless BBOB functions. Extending them to other classes of numerical black-box optimization problems forms another important avenue for future research. In this context, we consider supervised learning approaches building on exploratory landscape analysis [25] as particularly promising. It has previously been shown to yield promising results for the task of configuring the hyper-parameters of CMA-ES [3]. Note, though, that all existing studies concentrate on static algorithm configuration and/or selection. We would therefore need to extend exploratory landscape analysis to the dynamic setting. First steps into this direction have been made in [16], where it is shown that the fitness landscapes, as seen by the algorithm, can change quite drastically during the run.

Short-term. All the objectives listed above are quite ambitious. We therefore also formulate a few short-term goals for our research. Building on the techniques used to select interesting algorithms in Section 4.3, we aim to create smaller algorithm portfolio's of algorithms for intial implementations of dynAS. This could be done based on techniques studied in this paper, or using measures like the Shapley value [11], allowing for much smaller portfolios which nonetheless capture the different performances of the algorithms. With such a portfolio we can then more efficiently carry out research on the problems mentioned above, i.e., how to warm-start the algorithms and how to decide when to switch from one algorithm to another.

ACKNOWLEDGMENTS

This work has been supported by the Paris Ile-de-France region.

REFERENCES

[1] Anne Auger, Dimo Brockhoff, Nikolaus Hansen, Tea Tuar, and Konstantinos Varelas. 2020. Data from BBOB-workshops and competitions on 24 noiseless functions. https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob.
[2] Anne Auger and Nikolaus Hansen. 2005. A restart CMA evolution strategy with increasing population size. In *Proc. of Congress on Evolutionary Computation (CEC'05)*. 1769–1776. https://doi.org/10.1109/CEC.2005.1554902
[3] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per instance algorithm configuration of CMA-ES with limited budget. In *Proc. of*

Genetic and Evolutionary Computation (GECCO'17). ACM, 681–688. https://doi.org/10.1145/3071178.3071343

[4] André Biedenkapp, H. Furkan Bozkurt, Frank Hutter, and Marius Lindauer. 2019. Towards White-box Benchmarks for Algorithm Control. *CoRR* abs/1906.07644 (2019). arXiv:1906.07644 http://arxiv.org/abs/1906.07644

[5] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. 2013. A survey on optimization metaheuristics. *Information Sciences* 237 (2013), 82–117.

[6] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: a survey of the state of the art. *JORS* 64, 12 (2013), 1695–1724. https://doi.org/10.1057/jors.2013.71

[7] Luc Devroye. 1972. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN.

[8] Benjamin Doerr, Mahmoud Fouz, Martin Schmidt, and Magnus Wahlström. 2009. BBOB: Nelder-Mead with resize and halfruns. In *Proc. of Genetic and Evolutionary Computation (GECCO'09)*, Franz Rothlauf (Ed.). ACM, 2239–2246. https://doi.org/10.1145/1570256.1570312

[9] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2018. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281* (Oct. 2018). arXiv:1810.05281 https://arxiv.org/abs/1810.05281 The BBOB datasets from [1] are available in the web-based interface of IOHanalyzer at http://iohprofiler.liacs.nl/.

[10] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3 (1999), 124–141.

[11] Alexandre Fréchette, Lars Kotthoff, Tomasz P. Michalak, Talal Rahwan, Holger H. Hoos, and Kevin Leyton-Brown. 2016. Using the Shapley Value to Analyze Algorithm Portfolios. In *Proc. of the AAAI Conference on Artificial Intelligence*. AAAI Press, 3397–3403. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12495

[12] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tuar, and Tea Tuar. 2016. COCO: Performance Assessment. *arXiv:1605.03560 [cs]* (May 2016). http://arxiv.org/abs/1605.03560 arXiv: 1605.03560.

[13] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proc. of Genetic and Evolutionary Computation (GECCO'10)*. ACM, 1689–1696.

[14] Nikolaus Hansen and Andreas Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *CEC*. 312–317. https://doi.org/10.1109/ICEC.1996.542381

[15] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[16] Anja Janković and Carola Doerr. 2019. Adaptive landscape analysis. In *Proc. of Genetic and Evolutionary Computation (GECCO'19)*. ACM, 2032–2035. https://doi.org/10.1145/3319619.3326905

[17] Pascal Kerschke and Heike Trautmann. 2017. Automated Algorithm Selection on Continuous Black-Box Problems By Combining Exploratory Landscape Analysis and Machine Learning. *arXiv preprint arXiv:1711.08921* (2017).

[18] P. Kerschke and H. 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

[19] Benjamin Lacroix and John McCall. 2019. Limitations of Benchmark Sets and Landscape Features for Algorithm Selection and Performance Prediction. In *Proc. of Genetic and Evolutionary Computation (GECCO'19)*. Association for Computing Machinery, New York, NY, USA, 261fi?!262. https://doi.org/10.1145/3319619.3322051

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

[21] Marco Locatelli and Fabio Schoen. 1999. Random Linkage: a family of acceptance/rejection algorithms for global optimisation. *Mathematical Programming* 85, 2 (1999).

[22] Ilya Loshchilov, Marc Schoenauer, and Michele Sèbag. 2013. Bi-Population CMA-ES Agorithms with Surrogate Models and Line Searches. In *Proc. of Genetic and Evolutionary Computation (GECCO'13)*. Association for Computing Machinery, New York, NY, USA, 1177fi?!1184. https://doi.org/10.1145/2464576.2482696

[23] Katherine Mary Malan and Irene Moser. 2019. Constraint Handling Guided by Landscape Analysis in Combinatorial and Continuous Search Spaces. *Evolutionary Computation* 27, 2 (2019), 267–289. https://doi.org/10.1162/evco_a_00222

[24] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, and Michèle Sebag. 2012. Adaptive Operator Selection and Management in Evolutionary Algorithms. In *Autonomous Search*, Youssef Hamadi, Eric Monfroy, and Frédéric Saubion (Eds.). Springer, 161–189. https://doi.org/10.1007/978-3-642-21434-9_7

[25] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory landscape analysis. In *Proc. of Genetic and Evolutionary Computation (GECCO'11)*. ACM, 829–836.

[26] Mario A. Muñoz, Michael Kirley, and Saman K. Halgamuge. 2015. Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content. *IEEE Trans. Evolutionary Computation* 19, 1 (2015), 74–87. https://doi.org/10.1109/TEVC.2014.2302006

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

[28] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. 2015. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences* 317 (2015), 224–245.

[29] Mario Andrs Muoz Acosta, Michael Kirley, and Kate Smith-Miles. 2018. Reliability of Exploratory Landscape Analysis. (06 2018). https://doi.org/10.13140/RG.2.2.23838.64327

[30] László Pál. 2013. Benchmarking a hybrid multi level single linkageal gorithm on the bbob noiseless testbed. In *Proc. of Genetic and Evolutionary Computation (GECCO'15)*. 1145–1152.

[31] Erik Pitzer and Michael Affenzeller. 2012. *A Comprehensive Survey on Fitness Landscape Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 161–191. https://doi.org/10.1007/978-3-642-23229-9_8

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

[33] Michael A. Schumer and Kenneth Steiglitz. 1968. Adaptive step size random search. *IEEE Trans. Automat. Control* 13 (1968), 270–276.

[34] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD*. ACM, 847–855.

[35] Sander van Rijn, Carola Doerr, and Thomas Bäck. 2018. Towards an Adaptive CMA-ES Configurator. In *PPSN (Lecture Notes in Computer Science)*, Vol. 11101. Springer, 54–65. https://doi.org/10.1007/978-3-319-99253-2_5

[36] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. 2016. Evolving the structure of Evolution Strategies. In *SSCI*. 1–8. https://doi.org/10.1109/SSCI.2016.7850138

[37] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Online selection of CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation (GECCO'19)*. ACM, 951–959. https://doi.org/10.1145/3321707.3321803

[38] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. 2020. Github repository with Project Data. (2020). https://github.com/Dvermetten/BBOB_DynAS.

[39] Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2020. Integrated vs. Sequential Approaches for Selecting and Tuning CMA-ES Variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM. https://doi.org/10.1145/3377930.3389831

[40] Costas Voglis, Grigoris S. Piperagkas, Konstantinos E. Parsopoulos, Dimitris G. Papageorgiou, and Isaac E. Lagaris. 2012. MEMPSODE: an empirical assessment of local search algorithm impact on a memetic algorithm using noiseless testbed. In *Proc. of Genetic and Evolutionary Computation (GECCO'12)*, Terence Soule and Jason H. Moore (Eds.). ACM, 245–252. https://doi.org/10.1145/2330784.2330820

[41] Costas Voglis, Grigoris S. Piperagkas, Konstantinos E. Parsopoulos, Dimitris G. Papageorgiou, and Isaac E. Lagaris. 2012. MEMPSODE: comparing particle swarm optimization and differential evolution within a hybrid memetic global optimization framework. In *Proc. of Genetic and Evolutionary Computation (GECCO'12)*, Terence Soule and Jason H. Moore (Eds.). ACM, 253–260. https://doi.org/10.1145/2330784.2330821

[42] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2012. Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors. In *Proc. of Theory and Applications of Satisfiability Testing (SAT'12) (Lecture Notes in Computer Science)*, Vol. 7317. Springer, 228–241. https://doi.org/10.1007/978-3-642-31612-8_18