Universiteit Leiden

Master Computer Science

Dynamic Algorithm Selection for Continuous Black-box Optimization

Name:	Dominik Jürgen Willi Schröder
Student ID:	s2402793
Date:	30/04/2021
Specialisation:	Business Studies
1st supervisor:	Prof. Dr. Thomas Bäck
2nd supervisor:	Dr. Hao Wang
PhD supervisor:	Diederick Vermetten
Ext. supervisor:	Dr.-ing. Carola Doerr (Sorbonne Université, France)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgement

First and foremost, I would like to express my highest appreciation and gratitude to my supervisors in this research project, Thomas Bäck, Hao Wang, Carola Doerr, and Diederick Vermetten. Your encouragement, continuous advice, and thoughtful guidance made this project possible and motivated me every day.

I also feel deeply thankful for my family and closest friends. Even in difficult times, you helped me to keep energy levels high and supported me in any other possible way. It means the world to me.

I would like to thank the staff at Leiden University and especially LIACS for ensuring that all administrative tasks were taken care of without any obstacles. In particular, I would like to thank Bas van Stein for offering useful advice during the master classes, as well as José Visser for her friendly support along my educational journey and graduation.

I would also like to thank the 'Studienstiftung des deutschen Volkes' for their broad support throughout my studies, not least for connecting me with inspiring fellow scholars.

Table of Contents

1 Introduction

Optimization problems arise in many disciplines, ranging from engineering and natural sciences to economics, logistics and production planning. The common challenge is to determine which input variables lead to the best possible output for a given problem, such as finding the best hyperparameter configuration for a machine learning model or designing a fuel-saving car body. In many applications, the function that maps solution candidates to a quantitative rating of their quality is not explicitly given, so that the optimum cannot be derived analytically. In this black-box scenario, evaluating the objective function is only possible through simulations or real-world experiments, both of which can be time-consuming and expensive. Thus, randomly searching for the optimal configuration of problem variables typically leads to high costs, especially for problems with large search spaces.

To find the optimum with less resources, substantial research effort is spent on engineering efficient optimization algorithms. Through stochastic models and self-adapting search parameters, these algorithms require fewer function evaluations to approach the optimum compared to random searching. Even if the global optimum cannot ultimately be found, optimization algorithms are designed to find sufficient solutions within only a few evaluations.

Over the last decades, various different algorithms from different algorithm classes have been developed. These algorithms differ considerably in their internal search procedures, such as maintaining different adaptive parameters during the optimization. Moreover, they show different search behaviours. For example, some algorithms perform better in the initial part of the optimization, called *exploration*, while others converge quicker in the final part, called *exploitation*. Therefore, algorithms typically show complementary performances on different problem instances. An algorithm that performs well on low-dimensional, unimodal problems may at the same time perform poorly on high-dimensional, multimodal problems.

Consequently, for each individual problem instance, the single best algorithm needs to be selected to achieve superior performance. This is commonly referred to as algorithm selection problem (Rice 1976). Even though the selection can be performed manually, it is a challenging task that requires profound expert knowledge. Therefore, various algorithm selection models have been developed to assist users with the decision on the most suitable solver. Widely known approaches are parallel algorithm portfolios and automated selection models that compute solution landscape features with a small proportion of the available evaluation budget.

While these *static* approaches have been proven to be effective, combining multiple algorithms on a single optimization run may lead to even greater performance gains. The concept of *dynamic algorithm selection* (dynAS) proposes to switch between different algorithms throughout the optimization to benefit from their distinct strengths during different search phases. Recent work by Vermetten, Wang, et al. (2020) showed that, in theory, even a single-switch dynAS approach may lead to significant performance improvements. Contrary to typical *hyperheuristics* that chain algorithms after a specified amount of function evaluations, their study assumes running the first algorithm until it reaches a certain function value, to then continue the optimization with the second algorithm until it reaches the final target value.

The work at hand aims to extend this research by implementing dynAS for a small portfolio of continuous black-box optimizers. The main objective is to examine if switching between algorithms is actionable, that is, if the predicted performance gains can be realized when implementing an algorithm switch. In particular, this research is focused on the switch method, or in other words, how information collected during the initial optimization phase can be passed on from one algorithm to the other, given the distinctions between different algorithm classes. This is based on the notion that the second algorithm needs to be *warmstarted*, meaning its adaptive parameters need to be reasonably set to realize performance improvements. Following a data-driven approach similar to the one presented in Vermetten, Wang, et al. (2020), this study zeros in on particular use cases that consist of selected problems, algorithm combinations and switch points.

The remainder of this work is structured as follows: In Section 2, previous work related to dynamic algorithm selection and continuous black-box optimization will be reviewed. The research question will be outlined in more detail in Section 3. Then, the algorithm portfolio assembled for this research will be described in Section 4, including implementation details and challenges. Afterwards, the approach to switch between these different algorithms will be explained and tested in a series of experiments on selected use cases, summarized in Sections 5 and 6. Finally, this report will discuss the results in Section 7 and provide an outlook for future work on dynamic algorithm selection in Section 8.

2 Related work

2.1 Benchmarking continuous optimizers

The algorithms studied here follow the logic of *iterative search heuristics* (IOH). Once initialized, an IOH typically performs an iterative search routine until it reaches a pre-defined termination criterion. In a generalized approach, this routine consists of query-based sampling, evaluation, and selection operators. The search history is utilized to update internal parameters and distributions that determine how new solution candidates are sampled. A generic IOH scheme is shown in Algorithm 1.

A common challenge when developing new optimization algorithms is how to assess their performance. On a single data set or function, a particular algorithm may find the optimum within just a few function evaluations. However, iterative heuristics often employ randomization when generating or selecting solution candidates. Thus, their performance can only be assessed empirically by aggregating multiple algorithm runs. Another question is how this performance generalizes to other types of problems. After all, the algorithm may be highly tuned on the problem at hand. To evaluate its search behaviour reliably, the algorithm needs to be tested on multiple different problems and problem classes.

Therefore, a procedure called *benchmarking* is applied. A benchmark consists of a fixed set of problems. Any new algorithm is tested on exactly the same functions to allow performance comparisons between the different solvers. In the domain of numerical optimization, a platform called *COCO* (COmparing Continuous Optimizers) (Hansen, Auger, Mersmann, et al. 2016) has been established for this purpose over the last years. The COCO platform offers interfaces to run solvers on various test suites that consist of multiple benchmark functions. Moreover, the COCO platform provides methods to trace and process performance data. The work at hand will focus on the COCO test suite *Black-Box Optimization Benchmark* (BBOB), which comprises 24 noiseless, single-objective test functions of the form $f\colon [-5,5]^d \to \mathbb{R}$ that need to be minimized (Hansen, Finck, et al. 2009). The functions are commonly grouped into five different problem classes, as shown in Table 1. The range of functions contains difficulties that are expected to occur frequently in the continuous domain, such as the presence of many local optima. The dimensionality, i.e. the number of problem variables, is commonly set to $d \in \{2, 3, 5, 10, 20, 40\}$ in the BBOB context. Furthermore, different problem instances are available based on randomized optima locations and transformations in the problem's function value and variable space (Hansen, Finck, et al. 2009). Three exemplary solution landscapes of BBOB functions in dimension 2 are shown in Figure 1.

Table 1: BBOB functions grouped into different problem classes

Function IDs	**Problem class**
f1 – f5	separable functions
f6 – f9	functions with low conditioning
f10 – f14	unimodal functions with high conditioning
f15 – f19	multimodal functions with adequate global structure
f20 – f24	multimodal functions with weak global structure

Algorithm 1 Generic algorithm design for IOHs on continuous functions, taken from Wang et al.(2020)

procedure IOH
  $t \leftarrow 0$ ▷ iteration counter
  $\mathcal{H}(0) \leftarrow \emptyset$ ▷ search history information
  choose a distribution $\Lambda(0)$ on $\mathbb{N}$ ▷ distribution of the number of samples
  while termination criterion not met **do**
    $t \leftarrow t + 1$
    sample $\lambda(t) \sim \Lambda(t-1)$
    based on $\mathcal{H}(t-1)$ choose a distribution $D(t)$ on $\mathcal{S}^{\lambda(t)}$ ▷ search space $\mathcal{S}$
    sample $(x^{(t,1)}, ..., x^{(t,\lambda(t))}) \sim D(t)$ ▷ solution candidates
    evaluate $f(x^{(t,1)}), ..., f(x^{(t,\lambda(t))})$ ▷ function evaluation
    choose $\mathcal{H}(t)$ and $\Lambda(t)$
  end while
end procedure

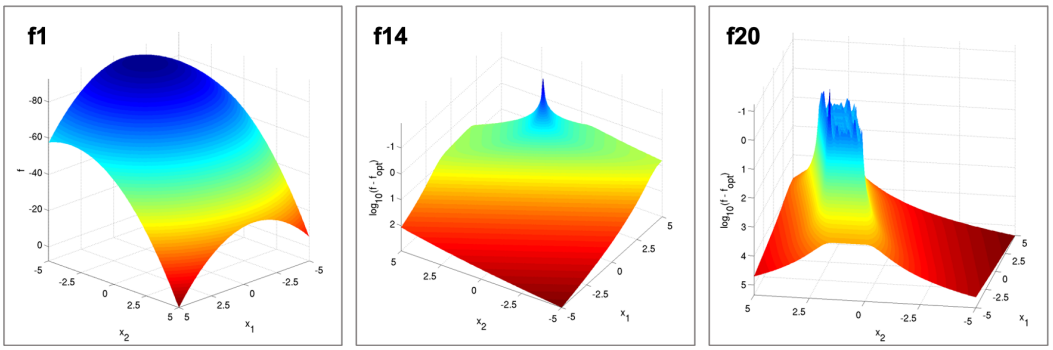


Figure 1: Exemplary functions from the BBOB test suite. Left: f1, the simple Sphere function from the class of separable functions. Middle: f14 is a unimodal function with high conditioning. Right: f20, a multimodal function with weak global structure. Illustrations taken from Hansen, Finck, et al. 2009.

The BBOB test suite has been used for workshops and competitions at academic conferences since 2009. So far, 232 different optimization algorithms[1] have been submitted to the BBOB platform. The corresponding performance data has been analyzed to calculate the potential of dynamic algorithm selection in Vermetten, Wang, et al. (2020). In the report at hand, the data will be studied further to identify use cases for algorithm switches as well as to assess the quality of algorithm implementations.

The *IOHprofiler* tool (Doerr et al. 2018) will be used for analysing algorithm performance in more detail. It consists of multiple components: The *IOHexperimenter* component can be used to benchmark algorithms on the BBOB test suite, similar to the COCO platform. The *IOHanalyzer*[2] component provides a convenient way to inspect empirical performance data. It takes in data files such as the ones created with COCO or IOHexperimenter and offers detailed insights into various metrics (Wang et al. 2020).

[1] Algorithm overview available at https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob (as of 15th April 2021)
[2] available at https://iohprofiler.liacs.nl/

2.2 Empirical performance measures

In the domain of numerical black-box optimization, performance is usually associated with the number of applied objective function evaluations. Yet, several empirical metrics are available that can take either a fixed-budget or fixed-target perspective. Fixed-budget metrics assume a situation where computational resources are scarce. The lower the function value drops within a limited budget of function evaluations[3], the better is the algorithm's performance.

In fixed-target approaches, the question is how many function evaluations an algorithm requires to reach a certain target. The *hitting time* $T(A, f, d, \phi)$ of algorithm A on function f in dimension d measures how many function evaluations are performed until the target precision $\phi = f_{\text{opt}} - f_{\text{best-so-far}}$ is reached. If the algorithm did not find the target within the allocated budget, the hitting time is set to $T(A, f, d, \phi) = \infty$.

Since algorithms may not always reach the defined target precision, previous work on continuous black-box optimization commonly refers to the *Expected Running Time* (ERT) (e.g. Hansen, Auger, Ros, et al. 2010, Kerschke and Trautmann 2019, Boks, Wang, and Bäck 2020). Assuming algorithm restarts for unsuccessful runs, the ERT estimates the expected hitting time of this restarting strategy by summing up the number of function evaluations to reach target precision for all runs and dividing it by the number of successful runs:

$$\text{ERT}(A, f, d, B, \phi) = \frac{\sum_{i=1} \min\{(T_i(A, f, d, \phi)), B\}}{\sum_{i=1} \mathbb{1}(T_i(A, f, d, \phi) < \infty)} \tag{1}$$

where i denotes the algorithm's i-th run on the problem and $\mathbb{1}$ stands for the characteristic function. The budget B is still required in the ERT equation to define after how many evaluations without reaching target precision the algorithm restarts. Even though the ERT has been established as standard metric in numerical optimization, it has been criticized by researchers, among other things, for unrealistically relaxed budget allocations (Bartz-Beielstein and Preuss 2011), using an absolute precision for varying objective function values (Kerschke and Trautmann 2019), and the lack of distinction between different problem instances (Kerschke and Trautmann 2019). Another weakness is that even though ERT is a fixed-target measure, changing the evaluation budget may have an influence on performance as it works as penalty term whenever an algorithm run does not reach target precision. Nonetheless, to ensure comparability with previous work, this report will focus on ERT as main performance measure.

Comparing algorithm performances based on the absolute number of function calls may be in some cases misleading, since the amount of evaluations is disproportionally elevated in higher dimensions and on more difficult-to-solve functions. To improve comparability between algorithms within our portfolio, we will also refer to the *relative ERT* (relERT). This metric is obtained by normalizing an algorithm's absolute ERT with the ERT of the best performing solver from the portfolio on the respective function and dimension. If an algorithm never reaches target precision on a certain function-dimension pair, its ERT is not defined. In such cases, we ap-

[3] in minimization

ply a PAR10 penalty score, meaning that we set the relERT to the tenfold of the portfolio's highest relERT on the respective function and dimension.

Moreover, this report will refer to the *Empirical Cumulative Distribution Function* (ECDF) to compare algorithms with each other. For a number of independent runs r, the ECDF is defined as proportion of runs where an algorithm A reaches target precision ϕ within a budget of function evaluations B over all runs.

$$\hat{F}_{\mathrm{T}}(B; A, f, d, \phi) = \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, \phi, i) \leq B)/r \tag{2}$$

The ECDF curve can be aggregated over multiple functions and dimensions. In IOHanalyzer, multiple target values can be included in the ECDF calculation to provide more insights on an algorithm's performance even before the target precision is reached. In this case, equation (2) is aggregated over a set of target values Φ:

$$\hat{F}_{\mathrm{T}}(B; A, f, d, \Phi) = \frac{1}{r|\Phi|} \sum_{\phi \in \Phi} \sum_{i=1}^{r} \mathbb{1}(T(A, f, d, \phi, i) \leq B) \tag{3}$$

The ECDF curve depicts the proportion of solved problems $p(f, d, \phi)$, consisting of function-dimension-target pairs, based on the available budget. Thus, it is a useful way to compare the overall performance of different algorithms.

2.3 Algorithm selection and configuration

The algorithm selection problem was first formalized by John R. Rice in 1976. Given a portfolio of algorithms $\mathcal{A}$ and a performance measure m for a set of problems $\mathcal{P}$, the task is to find a mapping $\mathcal{S}(x) : x \in \mathcal{P} \mapsto A \in \mathcal{A}$ that assigns algorithms to members of the problem class so that algorithm performance m is maximized (Rice 1976). A metaheuristic that would always select the best performing algorithm for a new problem instance is called *virtual best solver* (VBS). However, in the context of black-box optimization, prior knowledge about the problem characteristics is not available without additional function evaluations. Thus, the VBS only represents the theoretically ideal performance level for algorithm selection models. Moreover, the *no free lunch* theorem predicates that any solver performs exactly the same if averaged over all possible problems (Wolpert and Macready 1997). This also applies to algorithm selection metaheuristics. Nonetheless, these models still hold potential to improve performance if assumptions about the problem class can be made, which is often the case in real-world applications.

Dynamic algorithm selection extends the approach outlined above by selecting an algorithm not only for each problem instance, but for each time step of the optimization. Based on the assumption that different algorithms are better suited for certain search phases, e.g. exploration or exploitation, dynAS routines switch between different solvers online, that is, during the optimization process for a single problem instance. A recent study by Vermetten, Wang, et al. (2020) demonstrated that significant performance improvements are theoretically feasible with a single-split dynAS model, based on an extensive analysis of BBOB performance data. In their work, dynAS is defined as finding a policy $\pi : \mathbb{S} \to \mathcal{A}$ that selects an algorithm at each time step t of the optimization process, given the current internal state description of the algorithm $s_{\mathrm{t}} \in \mathbb{S}$. The work at hand extends their research

by studying if and how the theoretic performance gains can be materialized when applying dynAS to a small algorithm portfolio.

DynAS is closely linked to the problem of dynamic algorithm configuration (dynAC) that aims to find an optimal algorithm configuration at each time step of the optimization. Speck et al. (2020) define dynAC as finding an optimal control policy $\tilde{\pi}^* : \mathbb{N}_0 \times \tilde{\mathcal{S}} \times \mathcal{I} \to \tilde{\Theta}$ that realizes an algorithm configuration $\theta \in \tilde{\Theta}$ based on its current internal state $\tilde{s_t} \in \tilde{\mathcal{S}}$, given a problem instance $i \in \mathcal{I}$ and a time step $t \in \mathbb{N}_0$. According to Biedenkapp et al. (2019), algorithm selection can be seen as special case of dynAC where the algorithm selected per problem instance is defined as categorical hyperparameter. Recent studies have demonstrated that approaches based on reinforcement learning are suitable to learn efficient algorithm control policies (Biedenkapp et al. 2019, Speck et al. 2020).

Unlike online switching approaches in dynAS and dynAC, Kerschke and Trautmann (2019) propose to use a machine learning model to automatically select algorithms for new problem instances. The model is trained with problem features extracted by *Exploratory Landscape Analysis* (ELA), a technique that computes solution landscape properties, such as convexity or curvature, with a small proportion of the available function evaluations. On the BBOB test suite, their best selection model continuously outperforms the algorithm portfolio's single best solver. However, this approach is only viable if performance data on similar problem classes is available for each algorithm in the portfolio. Furthermore, automated algorithm selection leaves the theoretic potential of a dynamic VBS over a static one untapped.

Previous work on realizing the potential of dynAS has been focused on algorithms from the same algorithm class. For example, Vermetten, Rijn, et al. (2019) implemented a switch between different configurations of the modular CMA-ES framework (Rijn et al. 2016). Switching between algorithms that belong to different classes has so far only been realized in hybrid models. They either combine algorithm methods *within* their main iteration loop (e.g. Boks, Wang, and Bäck 2020, Pál 2013) or perform a local search algorithm at the end of an optimization run with a fixed number of function evaluations (e.g. Voglis et al. 2012). On the contrary, the approach presented here aims to switch between algorithm from different algorithm classes *outside* of their main iteration loop, after they have reached a certain function value. That is, the first algorithm (A_1) runs until a switch point τ, at which the second algorithm (A_2) continues with the optimization. The main challenge associated with this approach is how to hand-over information collected during A_1's run to A_2 to initialize its self-adapting parameters in a reasonable way, called *warmstarting*.

3 Research questions

In this work, we aim to implement a single-switch dynamic algorithm selection routine for a small portfolio of algorithms on selected function-dimension pairs. To begin with, we focus our research on assembling a suitable set of optimization algorithms. With that portfolio, we will investigate the effects of actually switching between the different solvers. Eventually, we will zoom in on different warmstarting approaches and their ability to realize the aspired performance gains of dynAS.

Figure 2 summarizes our main research focus in one graphic. It illustrates mock-up ERT curves of two algorithms in a dynAS process. The first algorithm, A_1, shows superior performance in the beginning. After reaching a precision of approximately 10^{-5}, the ERT curve leaps. If at that point, we switched to the second algorithm, A_2, we could improve performance significantly. The two main research questions are:

(1) How does the actual ERT curve of such a switch look like, i.e. does it follow the same course as A_2, shifted downwards to continue at the A_1 ERT curve, or will we see a mismatch due to not yet learned internal parameters?

(2) How do we resume the optimization run when switching to A_2, without restarting the search from scratch, and how do we initialize internal parameters in a reasonable way?

In particular, we aim to investigate the following more detailed sub-questions:

- How do we assemble a diverse algorithm portfolio that is suitable for research on dynAS?
- How do we identify use cases, that is, algorithm combinations on certain function-dimension pairs, and the respective switch points?
- What is the effect of switching between two algorithms on the overall performance?
- How do we warmstart the second algorithm in a reasonable way? How can we use information obtained during the first algorithm's run for warmstarting, given the distinct ways to handle information in diverse algorithmic designs?
- What are the use cases where dynAS shows superior performance compared to the portfolio's single best algorithm, and why?
- How do we ensure to switch at the desired switch point? What is the effect of switching at different switch points on performance?

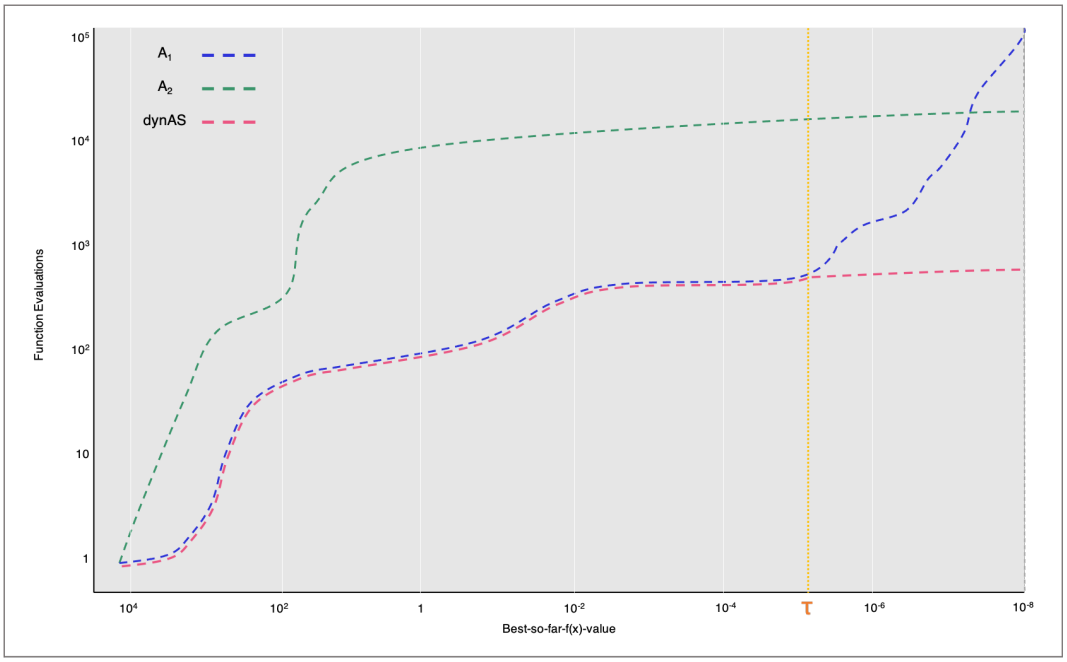


Figure 2: Mock-up ERT chart illustrating this work's research target. The blue line shows a typical A_1 ERT curve, the green line A_2, respectively. The pink line shows the aspired ERT curve of an algorithm combination where the second algorithm continues after the first algorithm reached a pre-defined switch point τ. The pink and blue lines are only slightly dislocated to make them both visible.

4 Algorithm portfolio

In this work, we focus on a small, yet diverse algorithm portfolio, rather than including the entirety of available solvers. This allows a more detailed research on dynamic algorithm selection and the associated warmstarting procedures. The following sections will outline which algorithms are included in the portfolio, why they have been selected, and how they work. Finally, particulars on the algorithm implementations and their respective empirical performance will be presented.

4.1 Portfolio composition

A key interest of this research is to study the warmstarting procedure when switching between algorithms that differ considerably with regards to their internal processes and adaptive parameters. Therefore, a portfolio should consist of algorithms from different algorithm classes. This report adopts the taxonomy proposed by Stork, Eiben, and Bartz-Beielstein (2018). Algorithms are grouped into different classes based on similarities in their initialization, generation, and selection methods as well as their control parameters. Table 2 shows how the different algorithm classes are characterized.

Table 2: Algorithm classes based on new taxonomy for optimization algorithms proposed by Stork, Eiben, and Bartz-Beielstein (2018). For each class, initialization, generation, and selection strategies are outlined, as well as its typical control parameters. Surrogate and hybrid algorithms have been omitted from the overview, since they are not considered in this work.

Algorithm class	**Initialization**	**Generation**	**Selection**	**Control Parameters**
1. Hill-Climbing	Single solution at random in the valid search space or based on prior knowledge	Variation of the last observed candidate, e.g. with gradient-based or stochastic methods	Elitist selection, greedy - does not accept inferior solutions	Step size that controls speed of convergence
2. Trajectory (exploring)	Typically single solution at random in the valid search space	Variation of the last observed candidate	Allow inferior solutions by parameter-driven acceptance function	Control acceptance function, e.g. temperature
3. Trajectory (systematic)	Typically single solution at random in the valid search space	Variation of the last observed candidate in attractive sub-spaces	Define attractive or avoidable sub-spaces	Definition of sub-spaces
4. Population (classic)	Multiple individuals (population) at random	Cross-over and mutation	Different selection methods, e.g. tournament selection	Several, e.g. population size, crossover-probability
5. Population (model-based)	Multiple individuals (population) at random	Cross-over and mutation with respect to distribution	Different selection methods	Several, often self-adaptive, e.g. evolution paths

As a starting point, this overview reveals interesting algorithm combinations for detailed research. For example, when switching from a hill climbing algorithm that only maintains a single solution, how can we initialize multiple individuals for a population-based algorithm? How can we hand-over information about interesting sub-spaces that has been generated by running a systematic trajectory algorithm to another algorithm? How can we reasonably set control parameters, such as step size, if these parameters are not maintained in a different algorithm class? Consequently, the portfolio contains representatives from different algorithm classes to allow research on a diverse set of algorithm combinations.

Another source of information to compose the algorithm portfolio is the data that has been generated by Vermetten, Wang, et al. (2020). In their work, the authors analysed algorithms that have been previously submitted to the BBOB competition. From the set of 226 possible algorithms, only the ones that had complete data files, performed at least 15 runs on a function, and reached at least a target precision of $\phi = 10^{-8}$ were considered. Then, a split policy was applied to all possible algorithm combinations and a set of split points, given by $(A_1, A_2, \tau) \in \mathcal{A} \times \mathcal{A} \times \Phi$, where $\Phi = \{10^{2-0.2i} | i \in \{0, ..., 50\}\}$ denotes the set of split points, A_1 denotes the first algorithm that runs until a split point τ, and A_2 denotes the second algorithm that runs from τ until it reaches the target $\phi = 10^{-8}$. The performance of such a single-split dynamic solver was then calculated by:

$$T(f, d, A_1, A_2, \tau, \phi) = ERT(A_1, f, d, \tau) + ERT(A_2, f, d, \phi) - ERT(A_2, f, d, \tau) \quad (4)$$

For each function-dimension pair, the best performing combination (A_1, A_2, τ) has been identified, called *Virtual Best Dynamic Solver* :

$$\mathrm{VBS}_{\mathrm{dyn}}(f, d) = \underset{(A_1, A_2, \tau) \in (\mathcal{A} \times \mathcal{A} \times \Phi)}{\arg\min} T(f, d, A_1, A_2, \tau, \phi) \quad (5)$$

This analysis highlights which individual algorithms appear frequently in one of the best performing combinations, such as the hybrid algorithm HMLSL that is part of 15 combinations as A_1, or the DE-AUTO algorithm that appears 8 times as A_2.

However, most algorithms on the BBOB platform belong to a higher level *algorithm family*. For example, the algorithms PSA-CMA-ES and IPOP-CMA-ES-2019 are both variants of the covariance matrix adaption evolution strategy (CMA-ES). Findings on the warmstarting procedure for the overall algorithm family will presumably be transferable to its variants as well, as long as they maintain similar internal parameters, such as the covariance matrix in CMA-ES. Therefore, the frequency of different algorithm families to be included in $\mathrm{VBS}_{\mathrm{dyn}}$ as either A_1 or A_2 is of higher interest for this research than just the frequency of individual algorithms. Figure 3 builds on the previously mentioned data while enriching it with information on the corresponding algorithm family for each solver that is part of a $\mathrm{VBS}_{\mathrm{dyn}}$. It shows that CMA-ES appears most frequently, both as A_1 and A_2. The Multi-Level Single Linkage (MLSL) algorithm and Quasi-Newton methods occur frequently as A_1, while Differential Evolution (DE) seems to be a good choice for A_2.

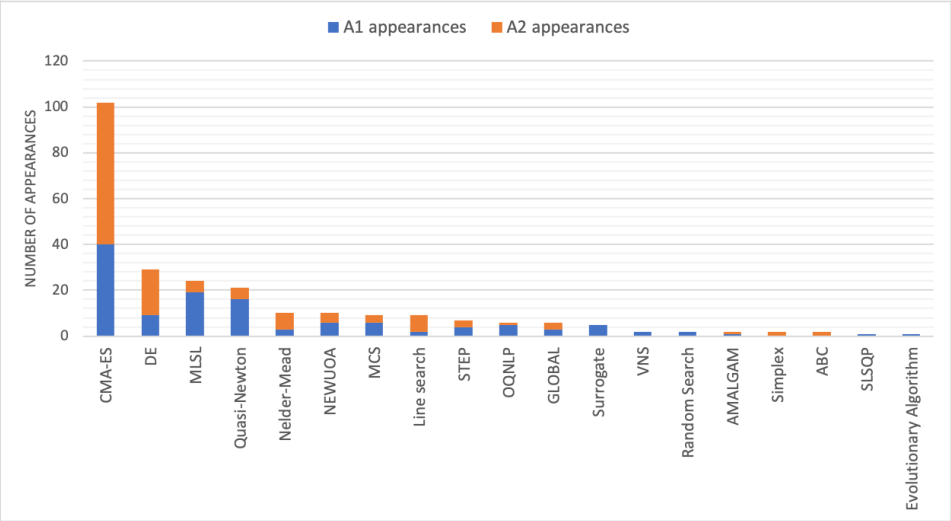


Figure 3: Number of appearances as A_1 and A_2 in best performing single-switch algorithm combinations VBS_{dyn} for different algorithm families, based on the data provided in Vermetten, Wang, et al. (2020). Hybrid algorithms are omitted from this overview as they consist of multiple algorithm families.

This work's portfolio consists of the five algorithms highlighted below. Each algorithm will be briefly introduced in the following section.

1. Broyden-Fletcher-Goldfarb-Shanno **(BFGS)**
2. Multi-Level Single Linkage **(MLSL)**
3. Particle Swarm Optimization **(PSO)**
4. Covariance Matrix Adaption - Evolution Strategy **(CMA-ES)**
5. Differential Evolution **(DE)**

4.2 Algorithm details

4.2.1 BFGS

BFGS is an optimization algorithm named after Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970) who all derived the BFGS update formula independently from each other at around the same time. BFGS belongs to the family of Quasi-Newton methods that approximate the Jacobian or Hessian instead of actually computing it. The optimum is located by finding the roots of the first-order derivative of the objective function following the iterative process of Newton's method. This is based on the assumption that the region around the optimum can be approximated as quadratic function. The algorithm is included in the portfolio due to the high number of appearances of Quasi-Newton methods as A_1 in the previously outlined analysis.

Algorithm 2 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

procedure BFGS
  Set initial x_0
  $B_0 \leftarrow I$ ▷ approximate inverse Hessian matrix
  $k \leftarrow 0$ ▷ iteration counter
  while termination criterion not met **do**
    $p_k = -B_k \cdot \nabla f(x_k)$ ▷ search direction
    $\alpha_k \leftarrow \text{argmin } f(x_k + \alpha_k \cdot p_k)$ ▷ line search to find step size
    $s_k \leftarrow \alpha_k \cdot p_k$
    $x_{k+1} \leftarrow x_k + s_k$
    evaluate $\nabla f(x_{k+1})$ ▷ gradient at x_{k+1}
    $y_k \leftarrow \nabla f(x_{k+1}) - \nabla f(x_k)$ ▷ gradient difference
    $B_{k+1} \leftarrow (I - \frac{y_k s_k^T}{y_k^T s_k})^T \cdot B_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}$ ▷ B_k update
    $x_k \leftarrow x_{k+1}$
    $k \leftarrow k + 1$
  end while
  return best x_k
end procedure

The optimization procedure of BFGS is shown in Algorithm 2. The algorithm starts by initializing a single solution candidate x_0 and setting the initial approximate inverse Hessian matrix B_0 as identity matrix. Then, the main iteration loop is initiated by determining the search direction $p_k = -B_k \cdot \nabla f(x_k)$. The gradient at the current point $\nabla f(x_k)$ is approximated by the finite difference method as it cannot be directly computed for black-box functions. Afterwards, the step size α_k is determined by line search. This can be done exact by solving argmin $f(x_k + \alpha_k \cdot p_k)$ or inexact with respect to the Wolfe conditions. The new solution candidate x_{k+1} is obtained by taking a step $s_k = \alpha_k \cdot p_k$ in the search direction. After evaluating the gradient at the new point and calculating the gradient difference $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$, the approximate inverse Hessian matrix is updated by the BFGS formula:

$$B_{k+1} = (I - \frac{y_k s_k^T}{y_k^T s_k})^T \cdot B_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k} \tag{6}$$

Finally, the iteration procedure starts again until a termination criterion is met. The algorithm returns the best solution x_k that has been observed during the optimization run.

4.2.2 MLSL

The Multi-Level Single Linkage (MLSL) algorithm combines global search phases based on clustering with local search routines. It belongs to the class of systematic trajectory algorithms. The key idea is to only start a local search in previously unexploited areas of attraction. The algorithm has been proposed by Kan and Timmer in 1987. As shown in the algorithm family appearance analysis depicted in Figure 3, MLSL and its variants seem to perform exceptionally well in the early part of optimization, which is why this algorithm is included in the portfolio.

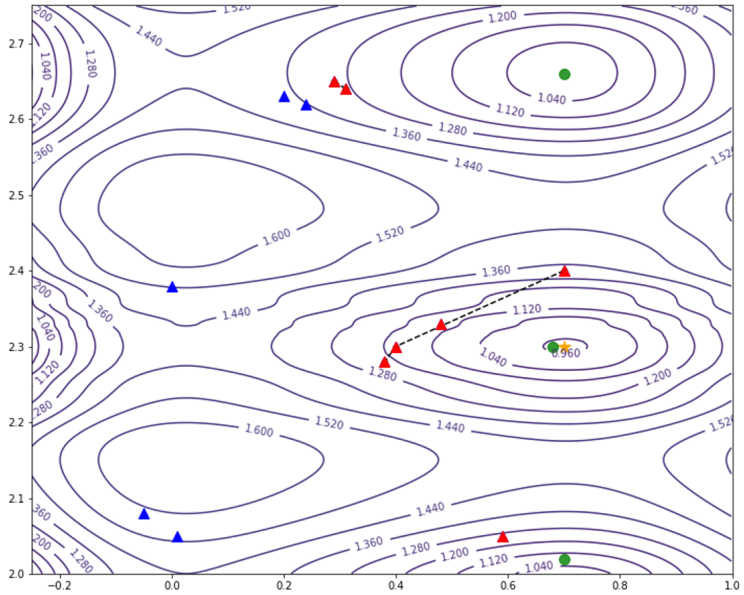


Figure 4: MLSL search routine on BBOB function 3 in dimension 2. Triangles (blue and red) show initial sample in the search space. Red triangles highlight points included in the reduced sample. Points that fulfill distance condition are clustered (dotted black lines). A local search routine is started at the best cluster points, with results depicted by green disks. The star icon shows where the global optimum is located.

Algorithm 3 Cluster method Multi-Level Single Linkage (MLSL)

procedure MLSL
  $X \leftarrow \emptyset$ ▷ Population
  $X^* \leftarrow \emptyset$ ▷ Points returned by local search
  $k \leftarrow 0$ ▷ iteration counter
  while termination criterion not met **do**
    $k \leftarrow k + 1$
    Add N random points to the population X
    $X_r \leftarrow$ best γkN best points from X ▷ reduced sample
    for $i \leftarrow 1$ **to** length(X_r) **do**
      if NOT (there is such a j that $f(x_j) < f(x_i)$ and $\|x_j - x_i\| < r_k$) **then**
        Start a local search method (LS) from x_i
        $x^* \leftarrow LS(x_i)$
        $X^* \leftarrow X^* \cup \{x^*\}$
      end if
    end for
  end while
  return best observed x^*
end procedure

Algorithm 3 and Figure 4 show the detailed search routine for MLSL. To start with, N points are sampled in the search space and then evaluated. The best γkN points are stored in the reduced sample X_r, where γ is a constant that determines the sample size. Afterwards, a local search routine is initialized for each $x_i \in X_r$, as long as there is no point x_j within a critical distance r_k that has a lower function value. This procedure is essentially implementing clusters. The critical distance r_k is given by:

$$r_k(x) = \frac{1}{\sqrt{\pi}}(\Gamma(1 + \frac{d}{2}) \cdot \lambda(X) \cdot \frac{\zeta ln(kN)}{kN})^{1/d} \tag{7}$$

where $\lambda(X)$ denotes the Lebesgue measure of the search space X, Γ denotes the gamma function, and ζ is a constant. In the BBOB context, the Lebesgue measure is given by $\lambda(X) = \sqrt{100d}$. If $\zeta > 2$, the probability of starting a local search is decreasing to 0 with an increasing number of iterations (Kan and Timmer 1987). The solutions x^* that are returned by the local search method are stored in X^*. In the next iteration, another N points are added to the population and the outlined routine repeats. Once the stopping criterion is met, the best solution candidate in X^* is returned.

4.2.3 PSO

Particle Swarm Optimization (PSO) is a nature-inspired algorithm that simulates the behaviour of animals aggregating in swarms, such as birds or fish. The particles move around the search space based on their individual velocity, determining both speed and direction. The change in velocity is influenced by the particle's best found position so far, called *cognitive component*, and the best position found by its neighbours, called *social component*. Thereby, the swarm iteratively converges to the optimum. The algorithm has been first introduced by Kennedy and Eberhart in 1995. It is included in the portfolio to experiment with a classic population-based algorithm which internal parameters are presumably simpler to warmstart compared to a model-based population algorithm. Moreover, in our initial experiments PSO showed advantageous exploitation behaviour on some difficult-to-solve BBOB functions, such as function 19 or function 24.

The algorithmic procedure of PSO with global best neighbourhood topology is shown in Algorithm 4. During initialization, a swarm of particles with random positions and velocities is created. Then, the iteration loop starts by updating the inertia weight, a parameter that controls the influence of the previous particle velocity in the velocity update and decreases over time. For each particle in the swarm, the velocity v_k is updated based on its current velocity, its best so far position $x_{best,k}$, and the best so far position found by the entire swarm x_{gbest}:

$$v_k = \omega \cdot v_k + \mathcal{U}1 \otimes (x_{best,k} - x_k) + \mathcal{U}2 \otimes (x_{gbest} - x_k) \tag{8}$$

where $\mathcal{U}_i$ is a random vector in $[0, \phi_i]^d$ and $\otimes$ denotes element-wise vector multiplication. Afterwards, the particle's position is updated by $x_{k+1} = x_k + v_k$. This update procedure is also shown in Figure 5. Finally, the solution candidate at the new position is evaluated to check if $x_{best,k}$ or x_{gbest} have to be updated. The algorithm repeats this procedure until a stopping criterion has been met and returns the best observed position.

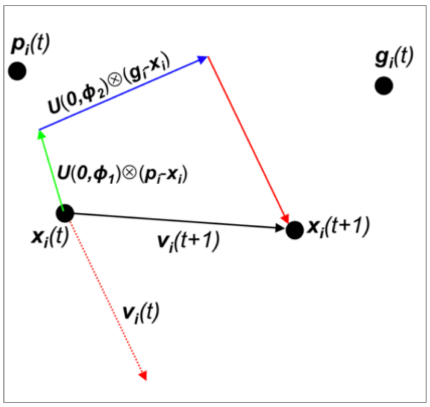


Figure 5: Velocity update procedure in PSO. The initial particle position is $x_i(t)$. It moves with velocity $v_i(t)$. The green arrow indicates the cognitive force, pulling the particle in the direction of its best so far position at $p_i(t)$. The blue arrow shows the social force, pulling the particle in the direction of the global best position so far $g_i(t)$. By adding all three vectors, the particle updates its velocity to $v_i(t+1)$ and arrives at its new position $x_i(t+1)$.

Algorithm 4 Population algorithm Particle Swarm Optimization (PSO)

procedure PSO
  $x_{gbest} \leftarrow \emptyset$ ▷ global best position
  $f_{gbest} \leftarrow \infty$ ▷ global best fitness
  for $i \leftarrow 1$ **to** swarm size **do** ▷ Initialize swarm
    Add particle i to the swarm
    $x_i \leftarrow$ random vector in $[x_{min}, x_{max}]^d$ ▷ particle position
    $v_i \leftarrow$ random vector in $[v_{min}, v_{max}]^d$ ▷ particle velocity
    if $f_i < f_{gbest}$ **then**
      $f_{gbest} \leftarrow f_i$
      $x_{gbest} \leftarrow x_i$
    end if
  end for
  while termination criterion not met **do**
    Update inertia weight ω
    for $k \leftarrow 1$ **to** swarm size **do**
      $\mathcal{U}_i \leftarrow$ random vector in $[0, \phi_i]^d$
      $v_k \leftarrow \omega \cdot v_k + \mathcal{U}1 \otimes (x_{best,k} - x_k) + \mathcal{U}2 \otimes (x_{gbest} - x_k)$ ▷ Update velocity
      $x_k \leftarrow x_k + v_k$ ▷ Update position
      if $f_k < f_{best,k}$ **then** ▷ Update personal best
        $f_{best,k} \leftarrow f_k$
        $x_{best,k} \leftarrow x_k$
      end if
      if $f_k < f_{gbest}$ **then** ▷ Update global best
        $f_{gbest} \leftarrow f_k$
        $x_{gbest} \leftarrow x_k$
      end if
    end for
  end while
  return best observed x
end procedure

4.2.4 CMA-ES

The Covariance Matrix Adaption Evolution Strategy (CMA-ES) is a widely known optimizer from the class of population-based algorithms. It follows methods inspired by biological evolution, such as mutation, recombination, and selection. Solution candidates are sampled from a multivariate normal distribution $\mathcal{N}(m, C)$, where m denotes the distribution mean and C denotes the covariance matrix that adapts and rotates the distribution. The pair-wise dependencies between variables stored in the covariance matrix are continuously updated by incorporating previous evolution paths. The idea behind this approach is that both parallel and anti-parallel correlation of consecutive mutation steps would be inefficient, while no correlation would be ideal (Hansen and Ostermeier 1996). The algorithm works derivative-free and performs especially well on ill-conditioned and non-separable problems (Hansen and Ostermeier 2001). It is included in this work's algorithm portfolio due to the high number of appearances as both A_1 and A_2, its state-of-the-art performance, and the variety of internal strategy parameters available for research on the warmstarting procedure.

Algorithm 5 Model-based population algorithm Covariance Matrix Adaption Evolution Strategy (CMA-ES)

procedure CMA-ES
  Set λ ▷ population size
  Set σ ▷ step size
  Initialize m ▷ distribution mean
  $C \leftarrow I$ ▷ Initialize covariance matrix
  $p_\sigma \leftarrow 0$, $p_c \leftarrow 0$ ▷ Initialize evolution paths
  while termination criterion not met **do**

  Sample new points
    for $k \leftarrow 1$ **to** λ **do**
      $z_k \sim \mathcal{N}(0, I)$ ▷ Sample from normal distribution
      $y_k \leftarrow BDz_k \sim \mathcal{N}(0, C)$ ▷ Apply eigenvectors B and eigenvalues D of C
      $x_k \leftarrow m + \sigma y_k \sim \mathcal{N}(m, \sigma^2 C)$
    end for
    $\langle y \rangle_w \leftarrow \sum_{i=1}^{\mu} w_i y_{i:\lambda}$ where $\sum_{i=1}^{\mu} w_i = 1$, $w_i > 0$ for $i = 1...\mu$
    $m \leftarrow m + c_m \sigma \langle y \rangle_w$ ▷ Shift distribution mean

  Step size update
    $p_\sigma \leftarrow (1 - c_\sigma) p_\sigma + \sqrt{c_\sigma (2 - c_\sigma) \mu_{eff}} C^{-\frac{1}{2}} \langle y \rangle_w$ ▷ Update evolution path
    $\sigma \leftarrow \sigma \times exp(\frac{c_\sigma}{d_\sigma}(\frac{\|p_\sigma\|}{E\|\mathcal{N}(0,I)\|} - 1))$

  Covariance matrix update
    $p_c \leftarrow (1 - c_c) p_c + h_\sigma \sqrt{c_c (2 - c_c) \mu_{eff}} \langle y \rangle_w$ ▷ Update evolution path
    $w_i^\circ \leftarrow w_i \times (1 \text{ if } w_i \geq 0 \text{ else } n / \|C^{-\frac{1}{2}} y_{i:\lambda}\|^2)$
    $C \leftarrow (1 + c_1 \delta(h_\sigma) - c_1 - c_\mu \sum w_j) C + c_1 p_c p_c^T + c_\mu \sum_{i=1}^{\lambda} w_i^\circ y_{i:\lambda} y_{i:\lambda}^T$
  end while
  return best observed x
end procedure

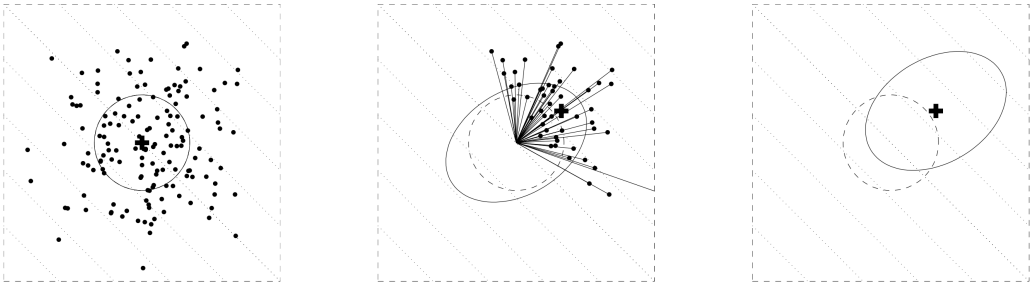


Figure 6: Procedure of the CMA-ES algorithm. Left: New individuals are sampled around the distribution mean (plus symbol). The distribution is isotropic. Center: The best individuals are selected for shifting the distribution mean. The distribution is adapted to an ellipsoid shape with information about evolution paths. Right: Distribution is centered around the new distribution mean for next generation. Taken from Hansen (2016).

The CMA-ES algorithm, as shown in Algorithm 5, starts by initializing step size σ, population size λ, and distribution mean m as well as setting the constants c_1, c_c, c_σ, c_μ, c_m and weights w_i. The covariance matrix C is initialized as identity matrix. The iteration loop starts with sampling mutation vectors y_k from $\mathcal{N}(0, C)$. New individuals are generated by $x_k = m + \sigma y_k$. The distribution mean is updated by $m \leftarrow m + c_m \sigma \langle y \rangle_w$, where $\langle y \rangle_w$ is the weighted average of the best μ new individuals. Next, the step size σ is updated with the evolution path p_σ. Both evolution paths p_σ and p_c are updated by cumulation. Finally, the covariance matrix is adapted by combining a rank-one update with a rank-μ update. In the next iteration, the updated distribution mean, step size, and covariance matrix determine the distribution for sampling new individuals from $\mathcal{N}(m, \sigma^2 C)$. The algorithm runs until a termination criterion is met and returns the best individual observed so far.

4.2.5 DE

Differential evolution is a population-based algorithm first proposed by Storn and Price in 1997. Inspired by biological evolution, it applies methods like mutation, recombination, and selection. Unlike gradient-based algorithms, the key idea in DE is to sample new solution candidates purely based on numerical differences between existing population members. Thus, the algorithm is applicable even for non-differentiable functions. The difference method also enables DE to automatically adapt from global to local search. However, if the algorithm is trapped in a local optimum, it does not automatically scale back. Due to its high number of appearances as A_2 in the analysis depicted in Figure 3, differential evolution is included in this work's algorithm portfolio.

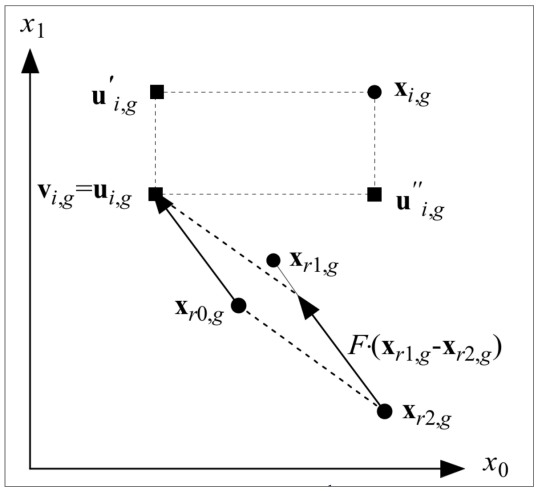


Figure 7: Mutation and cross-over procedure in differential evolution. A mutation vector $v_{i,g}$ is generated by applying $x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g})$ to three randomly selected members from the population. The distinct trial vector $u_{i,g}$ is created by cross-over of $v_{i,g}$ and the parent vector $x_{i,g}$.

Algorithm 6 Classic population-based algorithm Differential Evolution (DE)

procedure DE
  Initialize population of $N \geq 4$ points
  while termination criterion not met **do**
    for $i \leftarrow 1$ **to** N **do**
      $v_{i,g} = x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g})$ ▷ Generate mutation vector
      for $j \leftarrow 1$ **to** d **do** ▷ Create trial vector $u_{i,g}$
        if $\text{rand}_j(0,1) \leq C_r$ or $j = j_{\text{rand}}$ **then**
          $u_{j,i,g} \leftarrow v_{j,i,g}$
        else
          $u_{j,i,g} \leftarrow x_{j,i,g}$
        end if
      end for
      if $f(u_{i,g}) \leq f(x_{i,g})$ **then** ▷ Selection
        $x_{i,g+1} \leftarrow u_{i,g}$
      else
        $x_{i,g+1} \leftarrow x_{i,g}$
      end if
    end for
  end while
  return best observed x
end procedure

Algorithm 6 shows the detailed procedure of differential evolution. A population of $N \geq 4$ random solution candidates is initialized. The cross-over rate $C_r \in [0, 1]$ and the scaling factor $F \in [0, 1]$ are set. The iteration loops starts with generating a mutation vector $v_{i,g}$ for each population member $x_{i,g}$ by applying $x_{r0,g} + F \cdot (x_{r1,g} - x_{r2,g})$, where $x_{ri,g}$ are random members from the population distinct from $x_{i,g}$. Then, the cross-over operator creates a trial vector $u_{i,g}$ where each value j is taken from the mutation vector with a probability of C_r, and from its parent $x_{i,g}$ otherwise. Finally, the trial vector's fitness is compared to the parent's fitness, and only accepted for the next generation if it is superior. This scheme of mutation and cross-over is also shown in Figure 7. The algorithm stops once the termination criterion has been met and returns the best found solution.

4.3 Re-implementation study

4.3.1 Implementation details

For each algorithm from the just defined portfolio, we require an implementation to start experimentation on dynamic algorithm selection. We orientate ourselves towards existing algorithm submissions on the BBOB platform. In cases where the authors do not provide the code - or at least not in Python - we fall back on a different implementation while employing algorithm settings and parameters similar to the ones outlined in the respective submissions. This pragmatic approach is motivated by two factors: First, this work is mainly focused on the effects of chaining within a diverse set of algorithms, rather than achieving the highest possible performance with highly tuned solvers. Second, referring to previous BBOB submissions allows us to compare their performances and search behaviours to the data obtained while running our own implementations, effectively conducting a re-implementation study.

For **BFGS**, we use the implementation within Scipy's *optimize* module[4] (Virtanen et al. 2020). Similar to Baudis (2014), we keep the algorithm's default settings. The major difference to their implementation is that we refrain from basin hopping as a restart strategy. The gradient tolerance parameter g_{tol}, a threshold to terminate the algorithm run based on the norm of the current gradient vector, is changed from 10^{-5} to 10^{-10} to reach even lower target precision values. The initial guess x_0 is sampled randomly in $[-5, 5]^d$. Line search is performed inexact with respect to Wolfe conditions. Finally, it is worth noting that in the Scipy version, thus in our implementation as well, BFGS does not handle boundary constraints.

MLSL has been implemented anew.[5] The algorithmic parameters are set based on the BBOB submission from Pál (2013), with $\gamma = 0.1$, $N = 50d$, $\zeta = 2$, and a budget allocation for local search of 10% of the overall function evaluation budget. The major difference in our Python implementation is that we use the *Powell* method from Scipy's *optimize* module for the local search routine rather than MATLAB's *fmincon* interior-point method. We use the default settings to run the Powell method, except from a reduced fitness tolerance parameter $f_{tol} = 10^{-8}$ to reach even lower target precision values, as well as fixed bounds according to the BBOB definition.

For **PSO**, we use our own implementation[6] that is largely based on an existing Python implementation on Github.[7] Design choices are made in line with the BBOB submission from El-Abd and Kamel (2009). The neighbourhood topology is set to *global best*, that is, a particle's social component during the velocity update is influenced by the best position found by any other particle. The swarm size accounts for 40 particles, while the velocity update constants are set to $\phi_1 = \phi_2 = 1.4944$. In terms of boundary handling, particles that violate a constraint are positioned on the respective boundary, and their velocity is reset to a zero vector. Particle velocities are bounded to $[v_{min}, v_{max}] = [-5, 5]$.

[4] Scipy version 1.5.2., available at https://github.com/scipy/scipy/blob/master/scipy/optimize/optimize.py, as of 3rd February 2021

[5] available at https://github.com/Schroedo1994/Realizing_dynAS, as of 20th February 2021

[6] see footnote 5

[7] available at https://gist.github.com/tstreamDOTh/4af1d6b5a641deda16641181aa1e9ee8, as of 23rd February 2021

The inertia weight is iteratively updated based on the number of already performed function calls, $\omega = 0.9 - 0.8 \cdot \frac{\text{function evaluations}}{\text{evaluation budget}}$. While El-Abd and Kamel (2009) initialize particle velocities as uniform random vectors in $[-5, 5]^d$, in our implementation, we initialize $v_{i,0}$ in $[-1, 1]^d$ to favour exploiting search behaviour.

The modular CMA-ES framework[8] as proposed by Rijn et al. (2016) et al. represents the **CMA-ES** algorithm family in our portfolio. An updated version has recently been published by Nobel et al. (2021). The modular structure simplifies access to internal parameters and allows switching between different algorithm options, such as turning on or off an increasing population size behaviour. Consequently, by employing the modular CMA-ES implementation, we expect advantages for handing over information during the warmstarting procedure. The population size is set depending on the dimensionality of the problem at hand, with $\lambda = 4 + \lfloor 3 \cdot \log d \rfloor$. The modules for additional features, such as an increasing population size, are all turned off. The learning rates are kept in their default settings, that is:

$$
\begin{aligned}
c_s &= (\mu_{\text{eff}} + 2)/(d + \mu_{\text{eff}} + 5) \\
c_c &= (4 + (\mu_{\text{eff}}/d))/(d + 4 + (2\mu_{\text{eff}}/d)) \\
c_1 &= 2/(d + 1.3)^2 + \mu_{\text{eff}} \\
c_\mu &= \min((1 - c_1), (2 \cdot ((\mu_{\text{eff}} - 2 + (1/\mu_{\text{eff}})))/((d + 2)^2 + \mu_{\text{eff}})))
\end{aligned}
$$

where $\mu_{\text{eff}} = (\sum_{i=1}^{\mu} w_i)^2 / \sum_{i=1}^{\mu} w_i^2$. The step size is initially set to $\sigma = 0.5$. Lastly, the initial centre of mass m is sampled randomly with $m_i \in [0, 1)$.

As **DE** implementation, we employ the version that is available via Scipy's *optimize* module[9] (Virtanen et al. 2020). Most parameters are left in their default settings, such as the cross-over rate $C_r = 0.7$. We set the population size to $N = 5d$ and change the convergence tolerance parameter *tol* from 0.01 to $tol = 10^{-12}$. The mutation scaling factor F is sampled from the interval $[0.5, 1]$ at each generation. In the Scipy implementation of differential evolution, the mutation update follows a *best-1* strategy, that is, the best point found so far is used for mutation instead of a random point $x_{r0,g}$ from the population. The mutation update is thus given by $v_{i,g} = x_{best} + F \cdot (x_{r1,g} - x_{r2,g})$. Finally, we set the option *polish* to *False*. Otherwise, we would essentially operate a hybrid algorithm that runs a few iterations of Scipy's *L-BFGS-B* algorithm after DE has converged. Posik and Klema (2012) also run the Scipy version of differential evolution on the BBOB suite. Except from an adapted cross-over probability of $C_r = 0.5$, they do not explicitly state how certain parameters are set but refer to the respective default settings. We thus assume a population size of $N = 15d$ and an active *polish* feature in their implementation.

[8] We used the pre-release version 1.0.8. of the configurable CMA-ES framework, now available as modCMAES with new versioning at https://github.com/IOHprofiler/ModularCMAES, as of 23rd February 2021

[9] Scipy version 1.5.2., available at https://github.com/scipy/scipy/blob/master/scipy/optimize/_differentialevolution.py, as of 23rd February 2021

4.3.2 Implementation challenges

As part of our research, we also learned about the challenges and difficulties attached to re-implementing existing algorithmic approaches. A widely known credo in almost all sciences is that research should be reproducible. Only then, scientific results can be validated and extended by other researchers. Therefore, we devote this section to a brief discussion of challenges that we observed over the course of this project.

First and foremost, re-implementing an existing algorithm would be easiest if the actual code was made available. Even though the experimental data is provided alongside algorithm submissions on the BBOB platform, the overview table[10] does not link to the code that generated it. This complicates understanding details like design choices, parameter settings and experimental setup.

Researchers often provide the required implementation details within the corresponding publication. However, there are two major problems with that. Firstly, publications from different authors and different publication channels follow distinct designs. As a consequence, details like parameter settings can be found in various sections, for example *algorithm presentation* or *experimental setup*, so that they are potentially missed by readers. Secondly, in many cases, not all parameters and settings are stated in the publications. For example, Posik and Klema (2012) simply state that "[f]or most parameters of DE [...], default values from the literature were used". This makes it difficult for us to replicate their results, since we cannot be certain whether modules like the *polish* have been turned on or not, or what the exact population size accounted for.

Another problem observed during implementation is the adaptability of existing approaches that have been coded in different programming languages. For our project, we required Python code to operate both the switching between algorithms and the experiments on *IOHexperimenter*. Often, existing implementations were only available in C++ or MATLAB, hindering an immediate application within our project.

In conclusion, reproducability could be significantly improved if the algorithmic and experimental code was made available alongside with publications more frequently. A standardized and agreed-upon publication framework to mention design choices and parameter settings would help to easily spot the relevant sections as a reader. Eventually, providing interfaces or versions in multiple programming languages makes existing algorithms more accessible.

4.3.3 Running algorithms on BBOB suite

We test our algorithm portfolio on the BBOB suite. All five algorithms run on the 24 noiseless BBOB functions. We set the dimensionality to $d \in \{2, 3, 5, 10, 20\}$. Similar to a large part of existing research, the algorithms run on the first five problem instances, $i \in \{1, 2, 3, 4, 5\}$. We perform 5 runs on each instance, resulting in a total of 25 runs per function-dimension pair. The experiment budget accounts for $10{,}000d$. Lastly, we set the target precision to $\phi = 10^{-8}$. The generated data

[10] available at https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob, as of 2nd March 2021

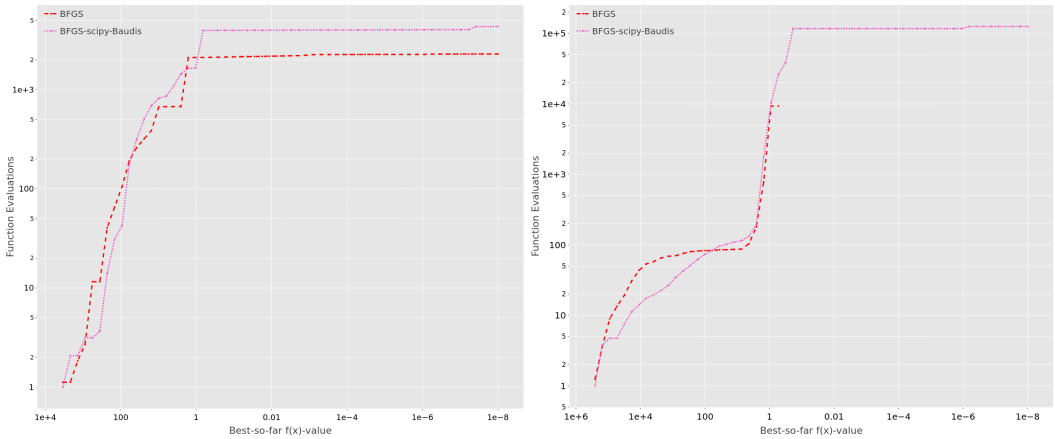


Figure 8: ERT charts for our implementation of BFGS (red) in comparison with the one from Baudis (2014) (pink). Left image shows ERT chart for function 15 in dimension 2, right image shows ERT chart for function 20, dimension 5.

forms the basis for various analyses throughout this report. It can be accessed and reviewed at Schröder (2021).

4.3.4 Comparison with previous BBOB submissions

To begin with, we use the empirical performance data to assess the quality of our implementations by comparing it with data from previous submissions on the BBOB platform. It must be noted that this comparison is only partly valid, given deviating parameter settings and experimental designs. For example, Posik and Klema (2012) run DE only once on 15 different instances with a budget of $50,000d$, while we employ 5 runs on 5 instances with an evaluation budget of just $10{,}000d$. Nonetheless, comparing the resulting data provides us with an indication about the quality of our implementations and the effect of differing design choices.

Reviewing ERT charts helps us to understand how our implementations compare with existing BBOB submission. Appendix 1 shows the ERT charts of our algorithms side by side with the algorithms mentioned in Section 4.3.1 for all 24 functions in two different dimensions. Just a comparison for CMA-ES is missing in this overview, since performance data for the modular CMAES framework is not yet available on the BBOB platform.

With our implementation of BFGS, we achieve similar performance on many BBOB functions, for instance on functions 1, 2, or 8 in dimensions 2 and 5. Figure 8 (left) illustrates this finding, with resemblant ERT curves for function 15 in dimension 2. On the other hand, Figure 8 (right) shows one example of a function-dimension pair where our implementation is inferior. More such examples originate from multimodal functions like 17, 18 or 23. This is probably caused by the lack of basin hopping in our version, leading BFGS to be trapped in local optima more frequently.

As depicted in Figure 9 (left) for the example of function 3 in dimension 2, both versions of PSO show a highly similar search behaviour on most functions. Only in higher dimensions, the differences become more evident. This is likely caused by the alternative way of initializing the particle velocities, leading to advantageous

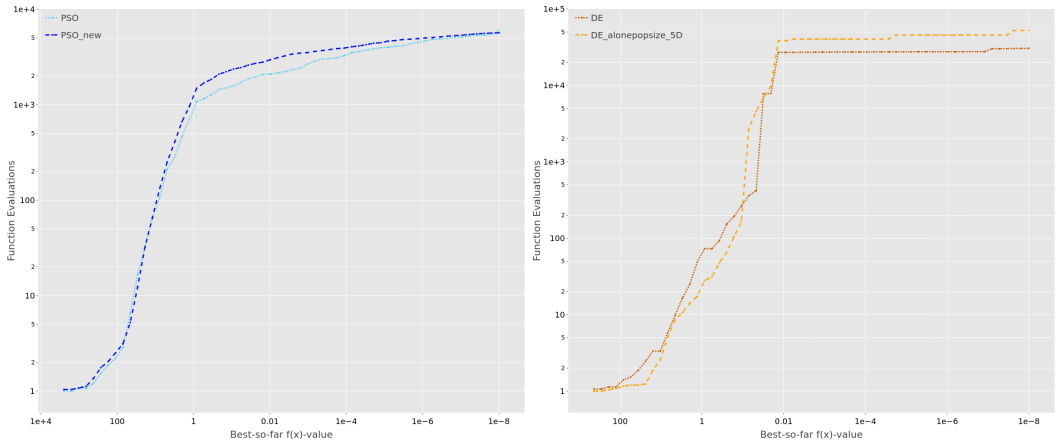


Figure 9: ERT charts to compare this work's algorithm implementation with the related BBOB submission. Left image shows our implementation ('PSO_new', dark blue) and the one from El-Abd and Kamel (2009) ('PSO', light blue) on function 3, dimension 2. Right image shows our implementation of DE ('DE_alonepopsize_5D', orange) in comparison with Posik and Klema (2012) ('DE', brown) on function 19, dimension 2.

performance of our implementation on some functions, like function 22 in dimension 10, yet unfavourable behaviour on other functions, e.g. function 16 in dimension 10.

The data shows that our version of DE performs similar to the related BBOB submission for most functions. For example, Figure 9 (right) shows the matching ERT curves of both algorithms on function 19 in dimension 2. While our implementation performs better in higher dimensions, such as on f11–f14 in dimension 10, the BBOB submission is superior on some functions in lower dimensions, for example functions 15 or 24 in dimension 2. Possible explanations for this observation are the differing population size and cross-over probabilities in both versions.

For MLSL, we observe significant differences between our implementation and the one from Pál (2013). The change of the local search method from *fmincon* to the Powell algorithm is likely causing these distinct performances. Nevertheless, our version of MLSL offers multiple opportunities to improve performance as part of an algorithm switch, as will be shown in the following sections.

To conclude, comparing the empirical performance data of our algorithm implementations with data from existing BBOB submissions demonstrated that our portfolio is suitable for further experiments on dynAS. While search behaviour and performance are similar for most function-dimensions pairs, both versions exhibit advantages on particular functions or dimensions. The differences originate largely from design choices and parameter settings, although differing experimental settings must be taken into account as well.

4.4 Performance and search behaviour analysis

With the performance data described in Section 4.3.3, we can not only evaluate the quality of our portfolio, but also develop a better understanding of each algorithm's search behaviour on certain functions and dimensions. Furthermore, the first use cases for dynAS emerge from analysing the data.

Table 3 highlights the relative ERTs for each algorithm in different dimensions, aggregated per function group. This helps us to better understand algorithmic search behaviour on certain function-dimensions pairs, while quantifying the variation of performances within our portfolio. For example, we can observe that CMA-ES outperforms the other algorithms in the function groups f6–f9 and f10–f14 in all dimensions. However, DE and PSO achieve the lowest relERT in other use cases. Especially the good performance of PSO for functions f15–f24 in lower dimensions is notable, since hill-climbing methods as employed by BFGS and MLSL were expected to excel compared to population algorithms. It is also worth noting that BFGS achieves the best relERT only once, while MLSL's performance lags behind on every function group-dimension pair. Nevertheless, the performance measure applied here is based on the algorithm's ability to approach a target precision of 10^{-8}. That said, it is still possible that any algorithm performs extremely well in the first part of the optimization, making it a suitable candidate as A_1 algorithm in a dynamic algorithm selection model.

Table 3: relERT per function group and dimension for all five algorithms in the portfolio. The best performing algorithm for the respective function group is highlighted in bold green font. The relERT for algorithms that did not reach target precision in any of their runs on a certain function-dimension pair was set to the tenfold of the highest relERT within that function group and dimension. Values for f15-f19 are not defined because none of the algorithms reached target precision for that function group in dimension 20.

Dimension	Functions	BFGS	MLSL	PSO	CMA-ES	DE
2	f1-f5	858.36	1717.84	106.17	30.88	**15.14**
	f6-f9	158.61	27.61	179.57	**3.15**	5.55
	f10-f14	1510.60	204.68	4594.73	**1.33**	4.09
	f15-f19	543.08	300.19	**1.32**	140.22	147.14
	f20-f24	1345.61	1348.97	**9.33**	61.87	460.66
3	f1-f5	4156.34	4367.73	103.94	**28.39**	29.94
	f6-f9	3308.18	48.08	3709.61	**4.48**	11.71
	f10-f14	3990.81	1484.01	5439.09	**1.00**	1.76
	f15-f19	46.11	46.11	2.40	2.51	**1.63**
	f20-f24	950.86	990.36	**17.63**	571.79	488.30
5	f1-f5	1763.63	2648.38	96.35	1771.89	**35.32**
	f6-f9	2993.33	3199.64	5985.66	**2.83**	3005.46
	f10-f14	1133.16	1188.87	1885.81	**1.00**	4.92
	f15-f19	22.00	22.00	18.04	13.61	**9.40**
	f20-f24	179.79	136.24	94.87	96.08	**48.10**
10	f1-f5	2007.56	3014.67	2130.04	2017.02	**1096.10**
	f6-f9	222.88	444.76	444.76	**3.93**	238.84
	f10-f14	853.22	1065.87	1065.87	**1.00**	874.01
	f15-f19	10.00	10.00	10.00	10.00	**8.20**
	f20-f24	40.57	25.36	24.85	24.82	**17.05**
20	f1-f5	**4096.80**	6151.21	4439.86	4105.76	6205.40
	f6-f9	1260.32	2519.65	2519.65	**634.11**	1332.56
	f10-f14	10.00	10.00	10.00	**1.00**	10.00
	f15-f19	not defined	not defined	not defined	not defined	not defined
	f20-f24	35.65	28.98	29.23	35.65	**28.72**

Reviewing the algorithms' ECDF charts as depicted in Figure 10 sheds light on their performance throughout the optimization process. It becomes apparent that BFGS solves the most function-dimension pairs in the lower range of available budget, an indication for suitability as A_1 algorithm. Likewise, DE seems to perform exceptionally well if more budget is available, particularly in lower dimensions, while CMA-ES shows similar advantages in higher dimensions. Thus, we hypothesize that both DE and CMA-ES are good candidates when switching towards an A_2 algorithm. Yet the question that remains open is whether the search behaviour outlined in the ECDF chart is a result of aggregating vastly different performances on vari-

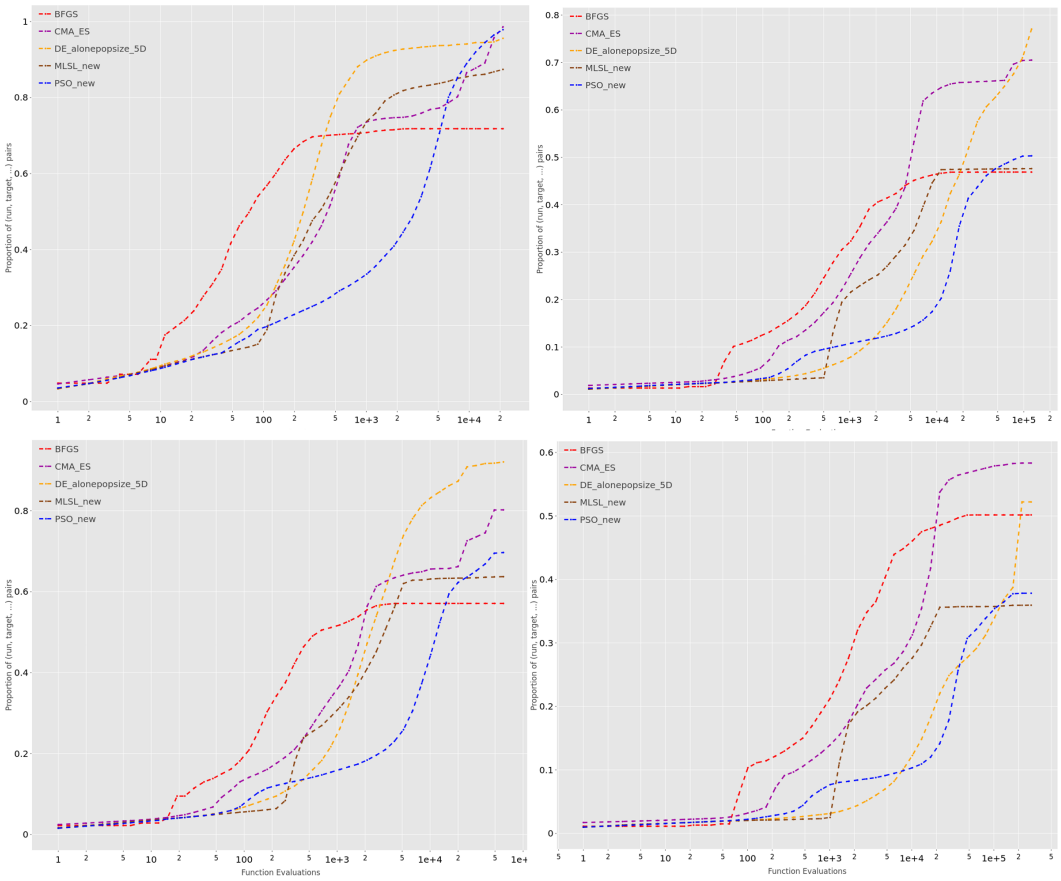


Figure 10: Empirical Cumulative Distribution Function (ECDF) curves, aggregated over 24 BBOB functions in dimensions 2 (top left), 5 (bottom left), 10 (top right), and 20 (bottom right). The *bbob targets* option in IOHprofiler has been turned on, which includes 51 log-linear targets between 10^2 and 10^{-8}. The vertical axis shows the proportion of function-dimension-target pairs that the algorithm has solved, given the respective budget on the horizontal axis.

ous functions, or if similar behaviour can be actually observed when zooming in on single function-dimension pairs.

As an example, Figure 11 illustrates the ERT charts for CMA-ES and BFGS on function 14 in dimension 5 (left) and dimension 20 (right). The figures show that BFGS indeed performs better in the early part of the optimization, that is, the algorithm reaches a target precision of the magnitude 10^{-5} to 10^{-6} with less function evaluations than CMA-ES. But from that point onward, CMA-ES requires less function evaluations to reach the final target precision of 10^{-8}. If we started the optimization with BFGS and switched to CMA-ES at the optimal switch point, we could reach target precision faster. Again, the question is whether these gains materialize, given that up to the switch point, CMA-ES has continuously adapted its internal parameters, thereby obtaining information about the solution landscape that may not be available from the parameters maintained by BFGS. This is in essence what we like to study with our experiments in the following sections.

Figure 12 shows another example that emerges from the data, a potential use case for the algorithms MLSL and PSO. Prior to implementing a switch between the different solvers though, we will identify more such use cases of algorithm combinations on particular function-dimension pairs, following a data-driven approach.

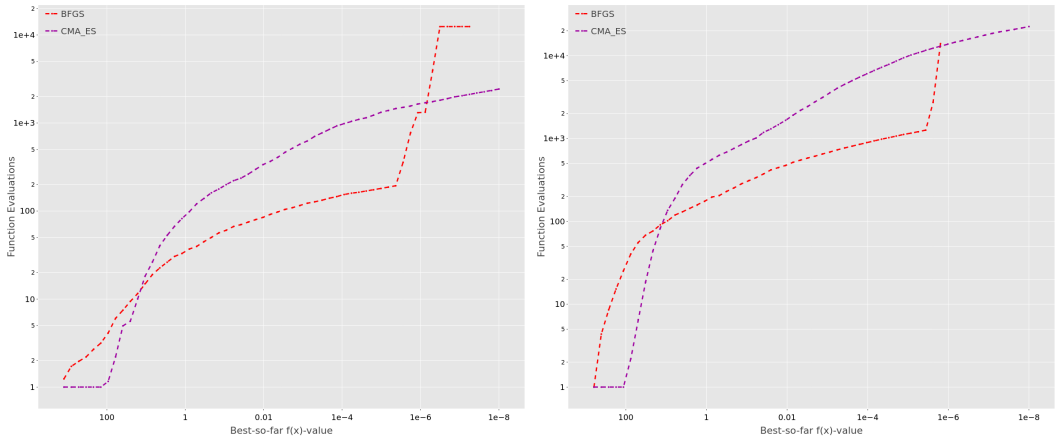


Figure 11: ERT charts for CMA-ES and BFGS on function 14 in dimension 5 (left) and 20 (right). All algorithms were run according to the setting outlined in Section 4.3.3.

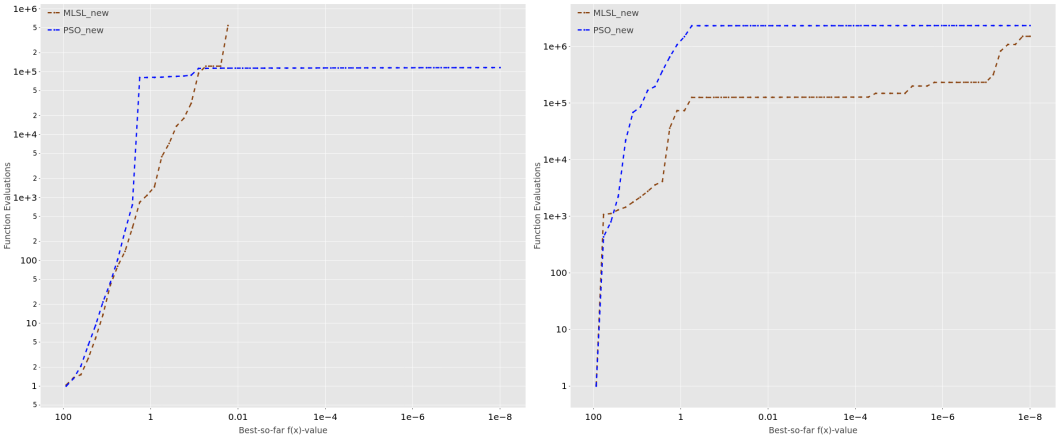


Figure 12: ERT charts for MLSL (brown) and PSO (blue) on function 24 in dimension 2 (left) and function 21 in dimension 20 (right). All algorithms were run according to the setting outlined in Section 4.3.3.

5 Experimental setup

5.1 Experimental settings

If not otherwise stated, we employ the same experimental settings on the BBOB test suite as outlined in Section 4.3.3 for all subsequent experiments, that is, five runs per instance on the first five instances, a target precision of $\phi = 10^{-8}$, the dimensionality set to $d \in \{2, 3, 5, 10, 20\}$, and a budget of 10,000$d$ function evaluations.

5.2 Identifying use cases

Based on the previously generated performance data (see Section 4.3.3), we follow the approach by Vermetten, Wang, et al. (2020) to identify use cases for dynAS within our portfolio. The approach is described in more detail in Section 4.1. For each function-dimension pair, we obtain $\text{VBS}_{\text{dyn}}(f, d)$, that is, a combination of A_1 and A_2 algorithms and the respective switch point τ that is expected to yield the highest ERT improvement, compared to the portfolio's static virtual best solver $\text{VBS}_{\text{static}}$. For a target precision of $\phi = 10^{-8}$, the analysis results in 87 potential algorithm combinations, out of the total 120 function-dimension pairs[11]. For the remaining pairs, no combination is available because none of the algorithms reached target precision, or none of the combinations outperformed $\text{VBS}_{\text{static}}$.

Table 4 shows the number of resulting use cases for each algorithm combination in our portfolio. Interestingly, almost half of the use cases are based on the combination of BFGS and CMA-ES. The remaining use cases are evenly spread across other combinations. It is worth noting that all five algorithms appear at least once as A_1 and A_2 algorithm. Another observation from the data is that the calculated switch point τ varies widely within the range $[10^2, 1.58 \cdot 10^{-8}]$, depending on the respective function-dimension pair.

Figure 13 depicts the potential ERT improvements within our algorithm portfolio for all functions and dimensions. In lower dimensions, potential ERT improvements are detected for almost all functions, with up to 92% for switching from PSO to DE on function 22 in dimension 2. Moreover, the function group of unimodal functions with high conditioning, f10–f14, seems particularly interesting with substantial improvement potential across all dimensions. In higher dimensions, we observe less use cases to improve performance with VBS_{dyn}, which is partly caused by less algorithms reaching target precision[12]. However, on functions 17, 21, and 22 in dimensions 5 to 20, the data shows the highest improvement potential within our portfolio, with up to 97% improvement when switching from BFGS to CMA-ES.

[11] resulting from 24 functions times 5 dimensions

[12] potentially larger improvements can be observed if we ease the target value. However, this is not our interest in this context.

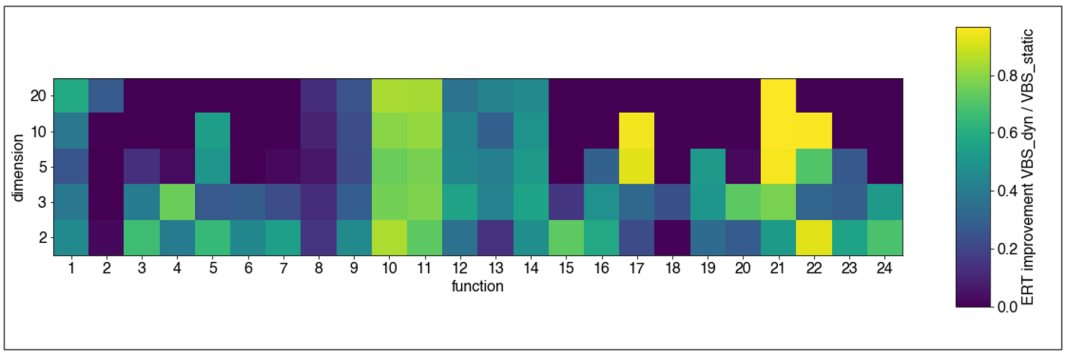


Figure 13: Potential ERT improvements of VBS_{dyn} over $\text{VBS}_{\text{static}}$ within our portfolio for all functions and dimensions. Percental improvement is calculated as (ERT($\text{VBS}_{\text{static}}$) - ERT($\text{VBS}_{\text{dyn}}$)) / ERT($\text{VBS}_{\text{static}}$).

Table 4: The number of use cases VBS_{dyn} for each algorithm combination in our portfolio. The target precision has been set to $\phi = 10^{-8}$. For more details, please refer to Appendix 4.

A_1	A_2	**Number of use cases**	**Functions and dimensions**
CMA-ES	DE	3	f16, f19, f23 [d5]
CMA-ES	PSO	0	-
CMA-ES	MLSL	0	-
CMA-ES	BFGS	12	f1 [d10, d20], f8, f9 [all dimensions]
PSO	CMA-ES	8	f3, f4, f15-f17 [d2], f19 [d2, d3], f5 [d5]
PSO	DE	5	f3, f7, f17, f22 [d2], f7 [d3]
PSO	MLSL	0	-
PSO	BFGS	1	f22 [d3]
MLSL	CMA-ES	4	f7 [d5], f16 [d2], f23 [d2, d3]
MLSL	DE	1	f21 [d2]
MLSL	PSO	3	f18, f24 [d2], f20 [d5]
MLSL	BFGS	6	f1 [d5], f2 [d20], f15 [d2], f20 [d2, d3], f21 [d3]
DE	CMA-ES	3	f5, f13 [d2], f18 [d3]
DE	PSO	4	f3, f4, f17 [d5], f24 [d3]
DE	MLSL	0	-
DE	BFGS	2	f1 [d3], f4 [d2]
BFGS	CMA-ES	30	f5, f6, f10-f14, f21, f22 [various dimensions]
BFGS	DE	0	-
BFGS	PSO	0	-
BFGS	MLSL	4	f2, f10, f11 [d2], f21 [d20]

5.3 Switch routine

To realize a single switch dynAS process, we have developed an algorithmic routine illustrated in Algorithm 7. We run the first algorithm, A_1, until a pre-defined target precision τ is reached. Meanwhile, we store the observed search history, such as the trajectory of sampled points or the progression of internal parameters. Together with A_1's current state $s_t(A_1)$ at the switch point $t = \tau$, the search history works as input for warmstarting the second algorithm, A_2. That is, we predict its state $s_t^*(A_2)$ at $t = \tau$ and set the internal parameters accordingly. Afterwards, we run the second algorithm until the final target precision ϕ has been reached.

For practical reasons during experimentation, we also include a maximum function evaluation budget as stopping criterion. While we limit ourselves to a single-switch

Algorithm 7 Single-switch dynamic algorithm selection

1: **procedure** SWITCH ROUTINE
2: Set target ϕ, switch point τ, budget B
3: $t \leftarrow 0$ ▷ iteration counter
4: Initialize A_1, $s_{t=0}(A1)$ ▷ Algorithm 1
5: $\mathcal{H}_{t=0} \leftarrow \emptyset$ ▷ Search and parameters history
6: **while not** ($\phi_{\text{best-so-far}} \leq \tau$ **or** evaluations $\geq B$) **do**
7: Run one iteration of A_1
8: Update $\mathcal{H}_t$
9: Update $s_t(A_1)$ ▷ Internal state description at time step t
10: $t \leftarrow t + 1$
11: **end while**
12: Initialize A_2 ▷ Algorithm 2
13: $s_t^*(A_2) \leftarrow \text{warmstarting}(A_1, A_2, \mathcal{H}_t, s_t(A_1))$ ▷ State prediction for A_2
14: $s_t(A_2) \leftarrow s_t^*(A_2)$ ▷ Initialize A_2 with s_t^* (warmstarting)
15: **while not** ($\phi_{\text{best-so-far}} \leq \phi$ **or** evaluations $\geq B$) **do**
16: Run one iteration of A_2
17: Update $\mathcal{H}_t$
18: Update $s_t(A_2)$
19: $t \leftarrow t + 1$
20: **end while**
21: **return** best found solution x
22: **end procedure**

model in this work, the outlined routine can be easily extended by repeating lines 10 to 17 for various switch points τ_i and algorithms A_i.

5.4 Warmstarting

As mentioned previously, this work aims to explore how we can switch between vastly different algorithms, and if we can realize the aspired ERT improvements derived from performance data. In doing so, warmstarting procedures are primarily developed per use case, that is, specific combinations of A_1 and A_2 algorithms. We leave a more generalized approach that implements warmstarting from any solver to any other open for future work on dynAS. Nonetheless, the algorithm specific routines presented here will establish a first step in this direction.

As a preparation for our experiments, we analyse each algorithm from our portfolio in detail with regards to their internal parameters. If the algorithm is set as A_1, we are interested in the information that is maintained during search to potentially use it as input for our warmstarting routine. While the common information carried by all algorithms are the trajectory of points and their respective fitness values, some algorithms maintain additional parameters. For example, BFGS maintains an approximate inverse Hessian matrix, whereas CMA-ES includes the estimated correlation between search variables.

From an A_2 perspective, the critical question is which parameters need to be warmstarted to resume search without performance loss. While answering this question is only possible with thorough experimentation, the table in Appendix 2 provides us with first indications by highlighting the parameters relevant for warmstarting in each algorithm. In many cases, parameters do not need to be warmstarted, since

they are re-calculated in the algorithm's main iteration loop before actually being applied. Examples are the step size α_k in BFGS or the critical distance r_k in MLSL. Other parameters however, such as the covariance matrix C in CMA-ES or particle velocities in PSO, are eligible for warmstarting. The table also reveals that initializing the population, or the first point in BFGS respectively, is a key element in warmstarting any algorithm within our portfolio.

5.5 Limitations

An unbiased view on our experimental results requires a brief discussion on the limitations associated with the dynAS approach presented in this work. To begin with, our approach is based on analysing empirical performance data for each algorithm in our portfolio. Naturally, this data is not always available in a real-world black box optimization scenario. This affects the selection of A_1 and A_2 as well as the pre-defined switch point τ. An operational dynamic algorithm selector would ideally determine the subsequent algorithm as well as the moment to switch automatically. Nevertheless, the BBOB suite contains optimization problems that typically occur in real-world applications as well. Therefore, our findings might still be actionable on other optimization tasks.

Another limitation related to relying on empirical performance data is the stochastic nature of iterative search heuristics. The high variability of hitting times for individual algorithm runs may result in a substantial variance of the recorded ERT values, depending on the respective experiment. Consequently, the data-driven selection of A_1, A_2, and τ may not always be ideal.

Next, we only consider a single-switch model in this work. However, the data reveals that on some use cases, switching multiple times and between multiple different algorithms could potentially lead to even higher performance improvements.

An additional limitation is the lack of tuning hyperparameters within our approach. Since we are more focused on a general proof of concept for warmstarting, rather than optimizing performance, we do not tune the algorithms' hyperparameters. Likewise, we refrain from tuning hyperparameters introduced in the warmstarting procedure. However, automatic hyperparameter tuning often impacts algorithm performance significantly, which could alter the selection of A_1 and A_2 algorithms and the performance of algorithm switches if applied to our portfolio.

Our fixed-target approach predicates that once a certain target precision is hit, an algorithm switch leads to advantageous search behaviour. This is based on the assumption that the first algorithm has already found an interesting region within the solution landscape, but fails to converge to the optimum as fast as the second algorithm. Especially for use cases with early switch points, this assumption may be flawed. After all, the first algorithm could be trapped in local optima, while the global optimum is located somewhere else. This is a direct consequence of the fixed target perspective, since target precision does not contain any information on proximity in the feature space. Moreover, target precision values may not always be available in real-world applications, since the exact location of the optimum and the respective function value are typically unknown.

6 Experimental findings

6.1 BFGS to CMA-ES

The highest number of use cases within our portfolio has been identified for switching from BFGS to CMA-ES (see Section 5.2). Out of the total 30 use cases for this algorithm combination, 22 include a unimodel, ill-conditioned function from the group f10–f14. Therefore, we start our experiments on a specific function from this group, namely f14 in dimension 2. The solution landscape is depicted in Figure 1. Based on our previous data analysis, we expect an ERT improvement of 48% when switching from BFGS to CMA-ES on this function-dimension pair, compared to $\text{VBS}_{\text{static}}$.

Reviewing the ERT charts of both algorithms as shown in Figure 17 provides us with a first intuition why dynAS may improve performance in this case. Initially, BFGS requires fewer evaluations to reach a target precision of $\tau = 3.98 \cdot 10^{-6}$. Beyond that point however, the ERT curve leaps drastically. Further analysis reveals that most BFGS runs terminate after reaching τ, with only 6 out of 25 runs arriving at the final target precision $\phi = 10^{-8}$. CMA-ES on the other hand converges more reliably, with all 25 runs reaching target precision. That is, CMA-ES shows advantageous exploitation behaviour compared to BFGS, which is likely caused by the ill-conditioned solution landscape around the optimum. With the subsequent experiments, we will examine if and how the outlined differences in search behaviour can be utilized in a dynAS process in order to improve performance.

6.1.1 Warmstart mean of the population distribution

To begin with, we develop a method to initialize the population in CMA-ES. Without warmstarting, CMA-ES would sample a new population at random in the search space, which contradicts our idea of resuming search where the first algorithm ended. As outlined in Section 4.2.4, CMA-ES samples its population around the distribution mean m. By setting the distribution mean at the switch point $m^{(\tau)}$ to the best point found by A_1, $x_{\text{opt,BFGS}}$, we effectively create a population in the neighbourhood of that point. Note that $x_{\text{opt,BFGS}}$ is also the last point sampled in BFGS, since we terminate the algorithm after reaching τ.

We run the combination BFGS-CMAES with warmstarted $m^{(\tau)}$ according to the settings outlined in Section 5.1. The blue line in Figure 17 depicts the resulting ERT curve. It follows the ERT curve of BFGS up to the switch point τ, after which it leaps. Eventually, it reaches and matches the ERT curve of CMA-ES. The final target precision $\phi = 10^{-8}$ is hit by all 25 runs, but the resulting ERT value is slightly worse than $\text{ERT}(\text{VBS}_{\text{static}})$.

To understand why we cannot realize the calculated ERT improvements with this method alone, we visualize the mutation distribution of CMA-ES right after the switch by sampling 10,000 additional points according to the algorithm's mutation operator, thereby depicting the shape of the distribution. As illustrated in Figure 14, the distribution is centred around $x_{\text{opt,BFGS}}$, as we intended. However, the plot

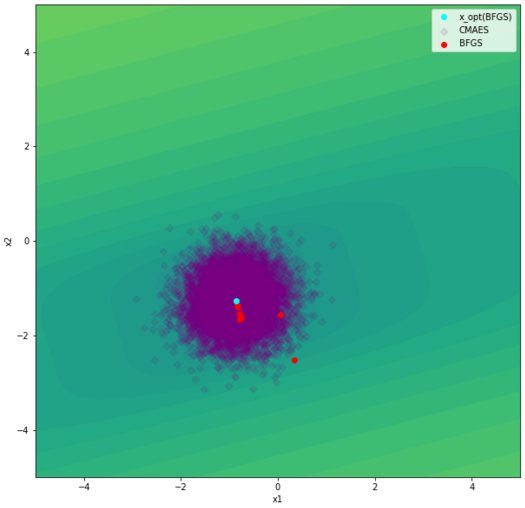


Figure 14: Contour plot of running dynAS(BFGS, CMA-ES, f14, d2, $\tau = 3.98 \cdot 10^{-6}$, $\phi = 10^{-8}$). Darker green-blue colors indicate smaller function values, that is, better target precisions. The initial distribution mean $m^{(\tau)}$ in CMA-ES has been set to $x_{\text{opt,BFGS}}$. The purple diamonds are points sampled by CMA-ES right after the switch, plotted by increasing population size to $\lambda = 10{,}000$. The plot has been created by 1 run on instance 1.

reveals that the mutation distribution is isotropic with a large diameter that even covers areas previously explored by BFGS.

We compare this plot to an ideal-typical[13] mutation distribution of CMA-ES at the switch point. We retrieve this by running CMA-ES until τ and saving all internal parameters. Then, we warmstart CMA-ES with the saved parameters and increase the population size again to $\lambda = 10{,}000$, while setting $m^{(\tau)} = x_{\text{opt,BFGS}}$. Figure 15 (left) shows that the ideal-typical distribution of CMA-ES at the switch point is much narrower than the mutation distribution we obtained in Figure 14. By zooming in, Figure 15 (right) reveals that the distribution is anisotropic, that is, shaped like an ellipsoid. Moreover, it is rotated to match the contour lines of increasingly small function values.

In CMA-ES, the mutation distribution's size, shape and rotation are influenced mainly by the parameters *step size* σ and *covariance matrix* C. Without warmstarting at the switch point, they are initialized as $\sigma^{(\tau)} = 0.5$ and $C^{(\tau)} = I$ in our implementation. Based on the contour plot comparison, we have to include both parameters in our warmstarting routine to achieve the aspired performance gains.

[13] By ideal-typical, we mean a distribution that is typical for an ideal distribution, which does not mean that it is ideal for every problem or problem instance.

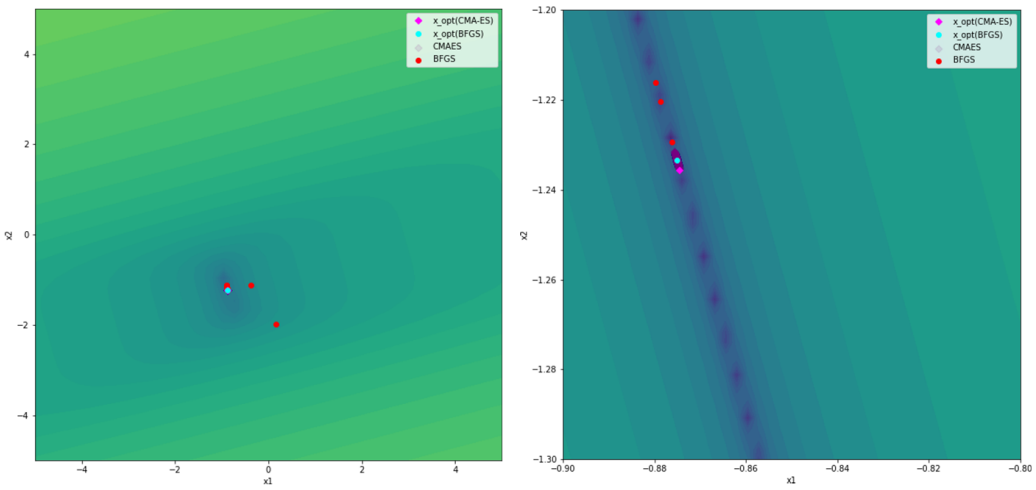


Figure 15: Contour plot of running dynAS(BFGS, CMA-ES, f14, d2, $\tau = 3.98 \cdot 10^{-6}$, $\phi = 10^{-8}$). Darker green-blue colors indicate smaller function values, that is, better target precisions. The initial distribution mean $m^{(\tau)}$ in CMA-ES has been set to $x_{\text{opt,BFGS}}$. All other parameters have been initialized according to saved internal parameters when running CMA-ES until τ. The purple diamonds are points sampled by CMA-ES right after the switch, plotted by increasing population size to $\lambda = 10{,}000$. The plot has been created by 1 run on instance 1. Left image shows the entire search space, while the right image zooms in on the area of the optimum.

6.1.2 Step size prediction

BFGS and CMA-ES both maintain step size as an internal parameter. Consequently, the intuitive procedure is to initialize the step size in CMA-ES with the equivalent value observed in BFGS at the switch point. However, the step sizes in both algorithms vastly differ in their meaning and the way they are calculated. While α_k in BFGS scales the gradient vector and results from independently performed line searches in each generation, σ in CMA-ES regulates the length of a multi-variate Gaussian mutation and is an adaptive parameter that is *learned* over time. Figure 16 highlights how both parameters evolve throughout the optimization process: In BFGS, α_k fluctuates strongly and does not converge to a particular value. On the other hand, σ in CMA-ES increases initially, but then steadily decreases and converges to $\sigma < 0.1$.

Based on these observations, we include a threshold parameter ϵ in the procedure to warmstart σ. If $\alpha_k^{(\tau)}$ is below this threshold, we set $\sigma^{(\tau)} = \alpha_k^{(\tau)}$. Otherwise, we set $\sigma^{(\tau)} = \epsilon$. This way, we can still use the available information from runs where the final α_k value is very low, without passing on undesirably high values in other runs.

We initialize $m^{(\tau)}$ as outlined in the previous section and set $\epsilon = 0.01$. Running BFGS-CMAES with this warmstarting routine on function 14 in dimension 2 results in a 6% improved ERT compared to ERT($\text{VBS}_{\text{static}}$). The ERT curve is indicated by the dark green line in Figure 17. Even though a small ERT improvement can be realized, it is still far off the calculated potential improvement of 48% for this function-dimension pair. Another problem is that ϵ has been arbitrarily selected, making it unlikely to be transferable to other functions and dimensions.

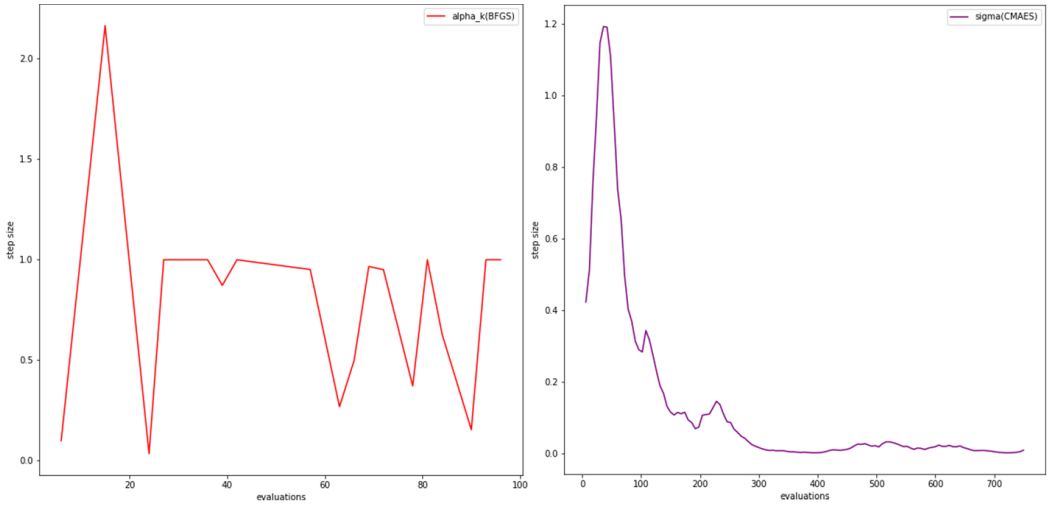


Figure 16: Step size evolving over the number of evaluations on function 14 in dimension 2. Left image shows α_k in BFGS. Right image shows σ in CMA-ES. Both charts have been generated by 1 run on instance 1. In our experiments, additional runs resulted in similar charts.

Therefore, we develop an alternative warmstarting routine to predict $\sigma^{(\tau)}$ that does not rely on α_k or a threshold value. For this method, we hand-over the trajectory of all points sampled by BFGS. Note that this only includes points evaluated as x_k, not the ones evaluated by performing line search or the finite difference method (more details in Section 4.2.1). We calculate the average Euclidean distance between the last n points in the trajectory and set this as our prediction for $\sigma^{(\tau)}$, given by:

$$\sigma^{(\tau)} = \frac{\sum_{j=0}^{n-2} \|x_j - x_{j+1}\|}{n-1} \tag{9}$$

where x_j is the j-th element in the trajectory of points X_{hist}, sorted so that x_0 is the last point sampled by BFGS. The idea behind this approach is that the closer BFGS converges to τ, the smaller is typically the distance between sampled points, which may predict an appropriate scaling for CMA-ES started at the switch point.

We set the number of points $n = 3$ and initialize $m^{(\tau)} = x_{\text{opt,BFGS}}$. Indicated by the bright green line in Figure 17, the ERT of running BFGS-CMAES with this step size prediction method is slightly worse than both $\text{VBS}_{\text{static}}$ (-2%) and the threshold method (-8%). Nonetheless, this method is less dependent on function-specific threshold values and might therefore be advantageous when applied to more function-dimension pairs. After all, the small differences in performance could be caused by the algorithms' stochastic nature.

Even though both methods to warmstart step size in CMA-ES slightly improve performance compared to just warmstarting the distribution mean, we still experience a significant leap in ERT when switching from BFGS to CMA-ES.

6.1.3 Warmstart covariance matrix

The next experiment is focused on warmstarting the covariance matrix in CMA-ES. For convex-quadratic functions, the covariance matrix directly relates to the objective function's inverse Hessian matrix. That is, setting $C = H^{-1}$ is equivalent

to optimizing a simple sphere function (Glasmachers and Krause 2020), which has been demonstrated to yield the optimal progress as it is one of the most extensively studied functions (Hansen 2016). Shir and Yehudayoff (2020) provide proof that in $(1, \lambda)$-evolution strategies on quadratic functions, the learned covariance matrix C is proportional to a *scaled* version of H^{-1} with increasing population size λ.

Obviously, not all BBOB functions belong to the class of convex-quadratic functions. However, if the mutation distribution only samples in a small neighborhood of the current search point, the local landscape will usually resemble that of a convex-quadratic functions, given some continuity assumptions on the objective function (e.g. *Lipschitz continuity*). This approach, called *active encoding*, is outlined in more detail in Hansen, Finck, et al. (2009).

Consequently, we hypothesize that the covariance matrix in CMA-ES can be warm-started by setting $C = \beta H^{-1}$, where β is a scaling factor. Moreover, we suggest that this approach only yields performance improvements on non-convex-quadratic BBOB functions if the step size at the switch point is sufficiently small, meaning that the optimization process is local enough to validate our assumptions above. This directly relates to our switch approach as well, because we expect the search behaviour of A_2 to be more localized the later we switch, or for smaller values for τ, respectively. For this particular algorithm combination, we can obtain the local Hessian matrix at the switch point directly from BFGS, as the algorithm maintains an approximate inverse Hessian matrix $B_k \approx H^{-1}$(see Section 4.2.1). We test this approach by setting $C^{(\tau)} = \beta B_k^{(\tau)}$. The scaling factor is set to $\beta = 0.1$ based on our intuition from prior experiments with alternating values. Additionally, we implement the previous warmstarting routines, namely $m^{(\tau)} = x_{\text{opt,BFGS}}$ and the prediction method for $\sigma^{(\tau)}$ with $n = 3$. As indicated by the pink line in Figure 17, this approach yields a significant ERT improvement (61%) compared to $\text{VBS}_{\text{static}}$, which is even 13 percentage points higher than the anticipated improvement for this use case.

By reviewing Figure 18, it becomes apparent that with the outlined warmstarting routine, the ellipsoid mutation distribution of CMA-ES right after the switch matches the ideal-typical distribution from Figure 15. The ellipsoid seems to be correctly sized, shaped, and rotated to find the optimum, indicated by the pink diamond, efficiently. Yet, the contour plots have been created by only one run on the first instance. Therefore, we need to test this approach with more runs on several instances and apply it to different use cases.

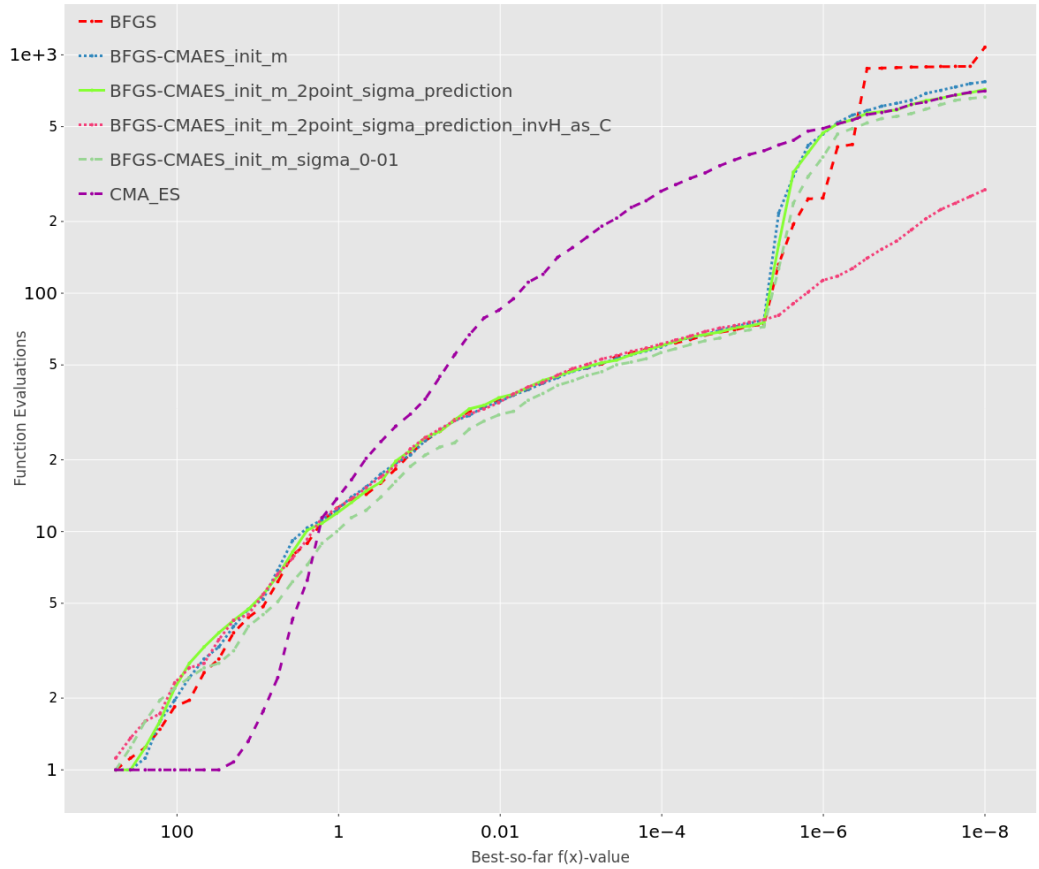


Figure 17: ERT chart for BFGS, CMA-ES and BFGS-CMAES with different warmstarting routines on function 14, dimension 2. All algorithms were run according to the experimental settings outlined in Section 5.1.

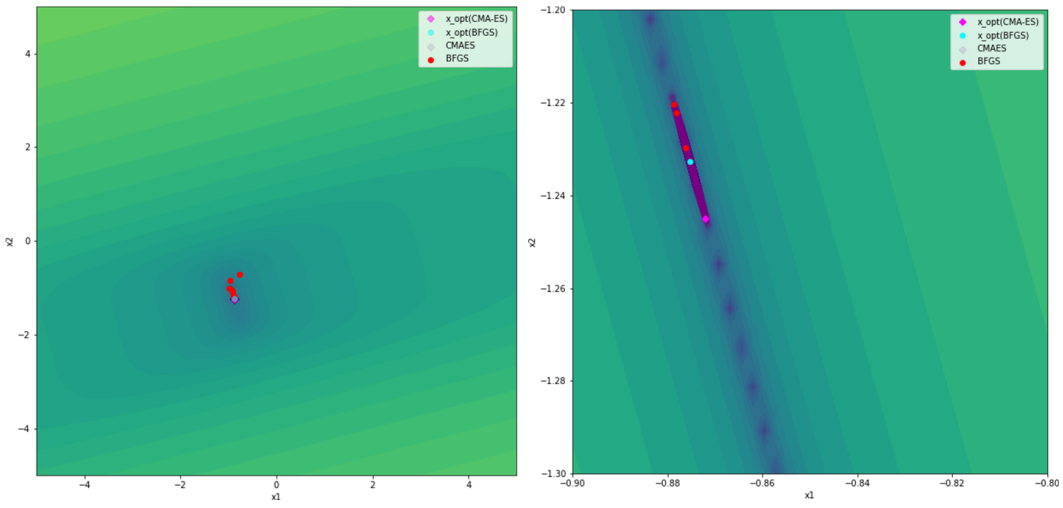


Figure 18: Contour plot of running dynAS(BFGS, CMA-ES, f14, d2, $\tau = 3.98 \cdot 10^{-6}$, $\phi = 10^{-8}$). Darker green-blue colors indicate smaller function values, that is, better target precisions. CMA-ES has been warmstarted according to the settings outlined in Section 6.1.3. The purple diamonds are points sampled by CMA-ES right after the switch, plotted by increasing population size to $\lambda = 10{,}000$. The plot has been created by 1 run on instance 1.

6.1.4 Validate results

We run the combination BFGS-CMAES with the warmstarting routine outlined in Algorithm 8 on all 30 identified use cases (see Section 5.2). Thereby, we aim to validate the promising results from applying this procedure on function 14 in dimension 2. Table 5 shows the corresponding ERT values for a selection of function-dimension pairs. It highlights that our warmstarting routine performs exceptionally well within the function group f10–f14. With 83%, the highest improvement is recorded for functions 10 and 11 in dimension 20.

On some functions, such as f13 or f14 in dimension 10, the actual ERT is even better than the expected ERT value derived from the data analysis. In a few other cases, the improvement is not as high as expected, such as for function 12 in dimension 2. Averaged over all 22 use cases within the function group f10–f14, the actual ERT(VBS_{dyn}) is 60% better than ERT($\text{VBS}_{\text{static}}$) and 5% better than the expected ERT(VBS_{dyn}).

On functions that do not belong to this group, the results are mixed. On function 6, our approach yields a 45% improvement in dimension 2, while performance deteriorates by 8% in dimension 3. On functions 21 and 22, the ERT of switching from BFGS to CMA-ES is considerably worse than ERT($\text{VBS}_{\text{static}}$). A possible explanation is that τ in the respective function-dimension pairs is comparatively large, i.e. the switch happens early during the optimization. This is problematic considering our earlier claim that initializing the covariance matrix with the approximate Hessian matrix from BFGS only works if the optimization is already zoomed in on a narrow area on the solution landscape. Another reason could be the multimodality of functions 21 and 22, leading BFGS to approach a local rather than the global optimum as it reaches τ.

Algorithm 8 Warmstarting routine BFGS-CMAES

procedure WARMSTART CMA-ES($s_t(BFGS)$, $\mathcal{H}_t$)
  Save $x_{\text{opt,BFGS}}$, X_{hist}, $B_k^{(\tau)}$ ▷ Observed by running BFGS
  $n \leftarrow 3$ ▷ Number of points for sigma prediction
  $\beta \leftarrow 0.1$ ▷ Scaling factor for approximate inverse Hessian
  $m^{(\tau)} \leftarrow x_{\text{opt,BFGS}}$ ▷ Warmstart distribution mean
  $\sigma^{(\tau)} \leftarrow \frac{\sum_{j=0}^{n-2} \|(x_j - x_{j+1})\|}{n-1}$ ▷ Warmstart step size
  $C^{(\tau)} \leftarrow \beta B_k^{(\tau)}$ ▷ Warmstart covariance matrix
end procedure

Table 5: ERT values for switching from BFGS to CMA-ES on selected function-dimension pairs. The target precision has been set to $\phi = 10^{-8}$. Percental improvements are calculated as (ERT($\text{VBS}_{\text{static}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / ERT($\text{VBS}_{\text{static}}$), and (expected ERT($\text{VBS}_{\text{dyn}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / expected ERT($\text{VBS}_{\text{dyn}}$), respectively.

A1	**A2**	**f**	**d**	τ	**ERT ($\text{VBS}_{\text{static}}$)**	**Expected ERT (VBS_{dyn})**	**Actual ERT (VBS_{dyn})**	**ERT impr. actual vs. $\text{VBS}_{\text{static}}$**	**ERT impr. actual vs. expected**
BFGS	CMA-ES	6	2	$6.31 \cdot 10^{-8}$	589.56	**324.33**	327.2	45%	-1%
BFGS	CMA-ES	6	3	$1.58 \cdot 10^{-7}$	942.56	**672.32**	1018.36	-8%	-51%
BFGS	CMA-ES	10	3	$1.58 \cdot 10^{-5}$	1135.92	**267.18**	267.36	76%	0%
BFGS	CMA-ES	10	20	$2.51 \cdot 10^{-5}$	19825.48	**3096.4**	3341	83%	-8%
BFGS	CMA-ES	11	10	$2.51 \cdot 10^{-4}$	5797.32	**1072.12**	1226.2	79%	-14%
BFGS	CMA-ES	11	20	$1.58 \cdot 10^{-4}$	15457	**2540.52**	2553.4	83%	-1%
BFGS	CMA-ES	12	2	$1.58 \cdot 10^{-6}$	1237.07	**791.6**	1032.04	17%	-30%
BFGS	CMA-ES	12	20	$6.31 \cdot 10^{-5}$	26711.84	16854.47	**15785.28**	41%	6%
BFGS	CMA-ES	13	5	$2.51 \cdot 10^{-3}$	3938.08	2294.00	**2111.24**	46%	8%
BFGS	CMA-ES	13	10	$2.51 \cdot 10^{-3}$	15987.36	11193.2	**9209.24**	42%	18%
BFGS	CMA-ES	14	2	$3.98 \cdot 10^{-6}$	705	364.72	**271.64**	61%	26%
BFGS	CMA-ES	14	10	$3.98 \cdot 10^{-6}$	6750.84	3414.2	**2399.08**	64%	30%
BFGS	CMA-ES	21	5	1	49843.62	**2258.5**	1184041.5	-2276%	-52326%
BFGS	CMA-ES	21	10	1	294066.29	**9924.00**	732071	-149%	-7277%
BFGS	CMA-ES	22	10	1.58	318490.5	**12056.5**	563100.88	-77%	-4571%

Figure 19 shows two more examples of how the warmstarting procedure affects performance when switching from BFGS to CMA-ES. In both function-dimension pairs, f10 in dimension 20 (left) and f13 in dimension 5 (right), the ERT curve of VBS_{dyn} follows the ERT curve of BFGS up to the switch point. Instead of leaping after the switch, as observed for warmstarting without initialized C matrix in Figure 17, the ERT curve follows the course of CMA-ES in the range from τ to ϕ. It is important to note the logarithmic scaling of the y-axis in these charts, which may lead to the conclusion that the ERT curve of BFGS-CMAES rises faster than the ERT curve of CMA-ES after the switch, which is not the case.

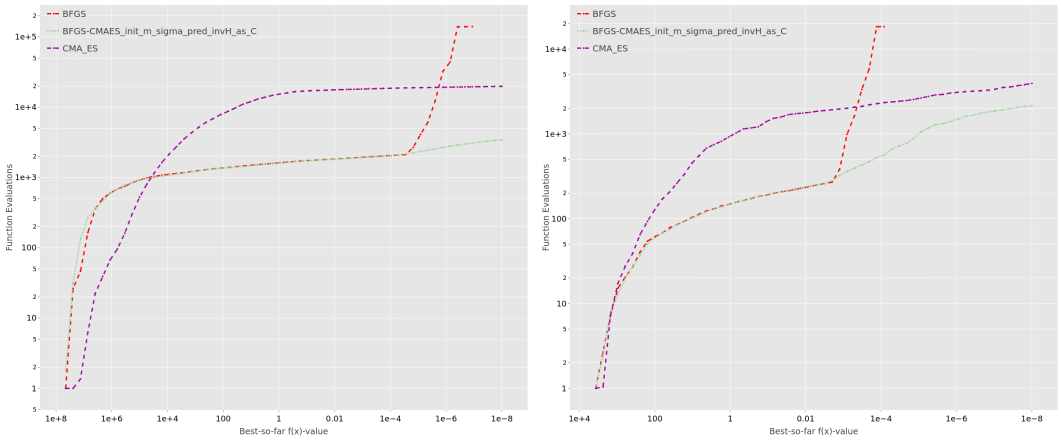


Figure 19: ERT charts for BFGS, CMA-ES and BFGS-CMAES with warmstarting routine according to Algorithm 8 on function 10, dimension 20 (left) and function 13, dimension 5 (right). All algorithms were run according to the experimental settings outlined in Section 5.1.

6.1.5 Conclusion

We have developed a warmstarting routine that realizes substantial performance gains when switching from BFGS to CMA-ES on several different function-dimension pairs. Our approach seems to perform especially well on unimodal, ill-conditioned functions (f10–f14). We have learned that from all examined parameters, warmstarting the covariance matrix had the largest effect on ERT. However, setting C to an estimated local inverse Hessian matrix may become exceedingly difficult in algorithm combinations where the A_1 algorithm, unlike BFGS, does not maintain this parameter. Moreover, the selection of parameters introduced in the warmstarting methods, like the number of points n in our sigma prediction method, or the inverse Hessian scaling factor β, still needs to be investigated. We expect that tuning them would allow even higher performance gains.

6.2 CMA-ES to BFGS

Inspecting the use cases for switching the other way round, i.e. from CMA-ES to BFGS, leads to a notable finding. For all function-dimension pairs that involve either function 8 or 9, this algorithm combination is expected to yield the highest performance gains over $\text{VBS}_{\text{static}}$. Figure 20 depicts function 8 (*Rosenbrock function*), where the optimum is located on a bent ridge. Function 9 is a rotated version of the Rosenbrock function. To help us explain this finding, we run each algorithm individually on function 9 in dimension 2 and plot all sampled points during search together with the solution landscape's contour lines of uniform target precision values.

As illustrated in Figure 21, BFGS follows the course of the ridge. However, many points are sampled on an indirect route towards the optimum as the algorithm's search direction aligns with the bend of the ridge. CMA-ES on the other hand, as shown in Figure 22, quickly approaches an area of the ridge closer to the optimum, but then samples many points before reaching the final target. This is likely caused by continuously updating and rotating the mutation distribution, determined by

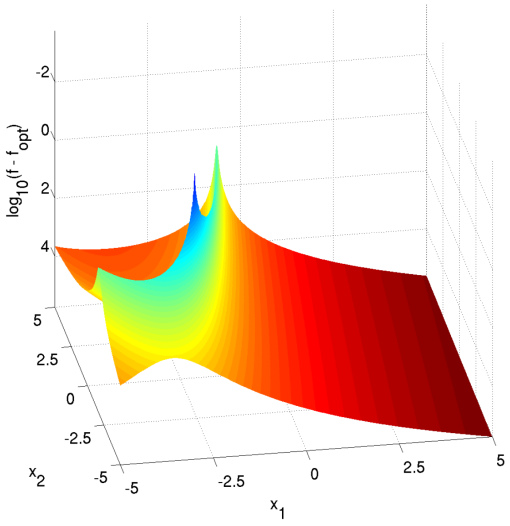


Figure 20: Solution landscape for BBOB function 8 (Rosenbrock function) in dimension 2. Image taken from Hansen, Finck, et al.(2009).

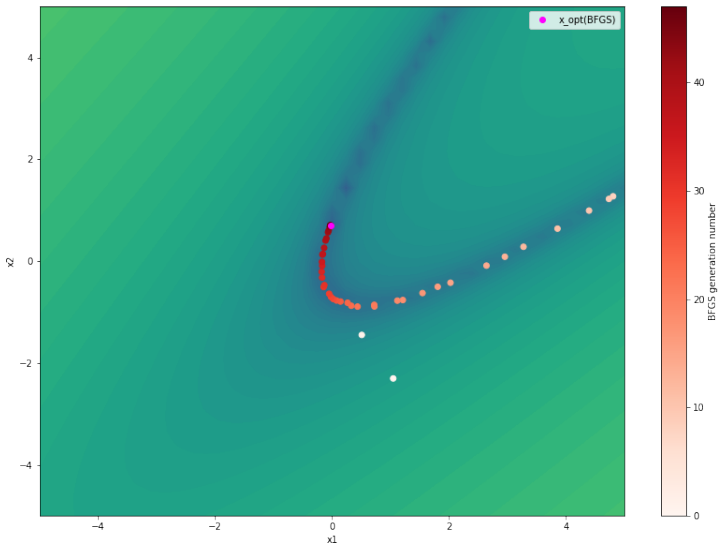


Figure 21: Contour plot of running BFGS on function 9, dimension 2, $\phi = 10^{-8}$. Darker green-blue colors indicate smaller function values. Darker red color indicates higher generation numbers of points x_k sampled by the algorithm. The target precision has been hit at $x_{\text{opt,BFGS}}$, highlighted in magenta. The plot has been created by 1 run on instance 1.

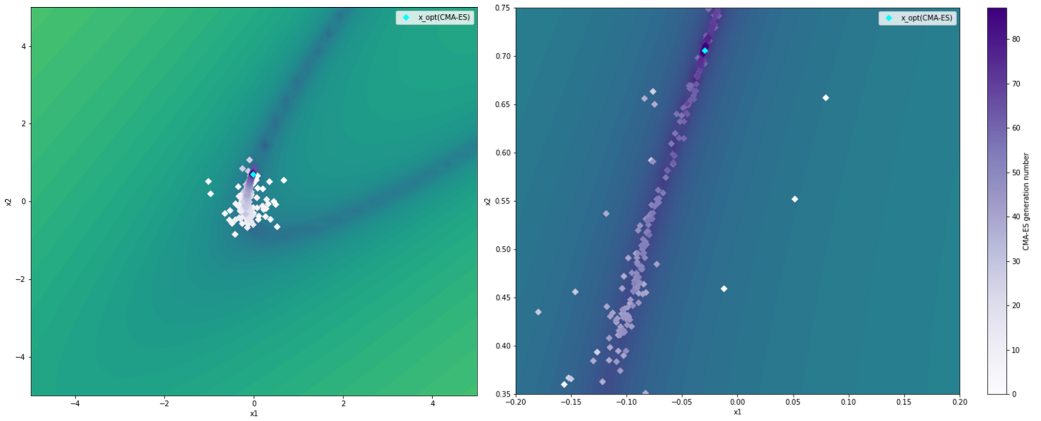


Figure 22: Contour plot of running CMA-ES on function 9, dimension 2, $\phi = 10^{-8}$. Darker green-blue colors indicate smaller function values. Darker purple color indicates higher generation numbers of points sampled by the algorithm. The target precision has been hit at $x_{\text{opt,CMAES}}$, highlighted in cyan. The plot has been created by 1 run on instance 1. Left image shows entire search space, while right image zooms in on the area of the optimum.

the covariance matrix, to sample solution candidates on the thin path of the ridge. Based on these learnings, we infer the following hypothesis: Switching from CMA-ES to BFGS on the Rosenbrock function improves performance, compared to $\text{VBS}_{\text{static}}$, because CMA-ES ($A_1$) reaches the area of the ridge that contains the optimum faster, while BFGS (A_2) converges to the optimum quicker as it requires less evaluations to follow the course of the ridge.

6.2.1 Warmstart initial point

The first parameter we consider for warmstarting BFGS is x_0, which is the initial point where BFGS starts its search. We set x_0 to the best point found by CMA-ES, $x_{\text{opt,CMAES}}$. We run this configuration on function 9 in dimension 2 with $\tau = 1.58$ and on function 8 in dimension 10 with $\tau = 10^2$, according to the experimental settings outlined in Section 5.1. Figure 24 depicts the resulting ERT charts. On both use cases, switching from CMA-ES to BFGS results in improved ERT values, accounting for 52% improvement on function 9 in dimension 2 (left image), and 37% on function 8 in dimension 10 (right image). The ERT curve of VBS_{dyn}, indicated by the light blue line, matches the ERT curve of CMA-ES up to the switch point, after which it continues similar to the course of the ERT curve of BFGS.

To test if the observed performance gains can be explained by our hypothesis, we again plot all sampled points during search together with the contour lines of equal target precision values, this time for the combination CMAES-BFGS. As shown in Figure 23 (left), CMA-ES requires three iterations to sample a point (cyan diamond) on the ridge that is already close to the optimum. Starting there, BFGS only samples very few points that are all positioned on the ridge, until it arrives at the target precision. Figure 23 (right) highlights this behaviour in more detail as it zeros in on the area of the optimum. Even though the plots presented here have been generated by only one run on the first instance, we observed the outlined behaviour on several independent runs on different instances throughout our experiments. Therefore, we rate these findings as support of our initial hypothesis.

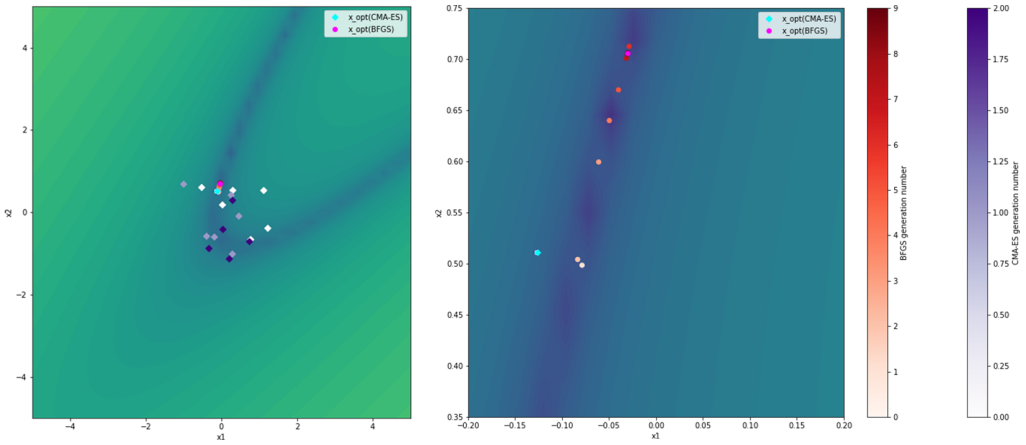


Figure 23: Contour plot of running CMAES-BFGS on function 9, dimension 2, $\phi = 10^{-8}$, $\tau = 1.58$. Darker green-blue colors indicate smaller function values. The target precision has been hit at $x_{\text{opt,BFGS}}$, highlighted in magenta. The plot has been created by 1 run on instance 1. Left image shows the entire search space, while right image zooms in on the area of the optimum.

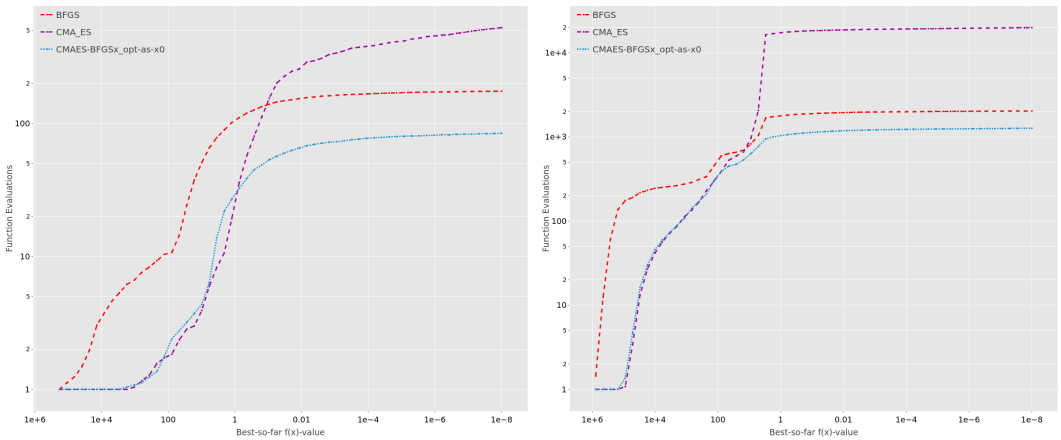


Figure 24: ERT charts for CMA-ES, BFGS and CMAES-BFGS with warmstarting routine for x_0 on function 9, dimension 2 (left) and function 8, dimension 10 (right). All algorithms were run according to the experimental settings outlined in Section 5.1.

6.2.2 Warmstart approximate inverse Hessian matrix

Another parameter in BFGS that is available for warmstarting is the approximate inverse Hessian matrix B_k. Without warmstarting, it will be initialized as $B_k = I$. Based on the relation between the covariance matrix and the inverse Hessian matrix as outlined in Section 6.1.3, we set the approximate inverse Hessian matrix at the switch point to $B_k^{(\tau)} = C^{(\tau)}$. We run this configuration on selected function-dimension pairs according to the experimental settings outlined in Section 5.1. Figure 25 depicts how the hitting times to reach target precision for individual runs differ on various function-dimension pairs, depending on whether or not B_k has been warmstarted. It becomes apparent that warmstarting B_k with this method does not have a large impact on hitting times. Only on function 9 in dimension 20, the ERT of CMAES-BFGS with initialized B_k is slightly better than without initialization.

This observation can be explained by the early switch, that is, large values for τ in all use cases. Prior to the switch, CMA-ES has only performed very few iterations. Consequently, the covariance matrix has not yet been learned properly, which results in a rather isotropic mutation distribution. As this is similar to setting B_k to the identity matrix, there is little effect on the performance of BFGS after the switch.

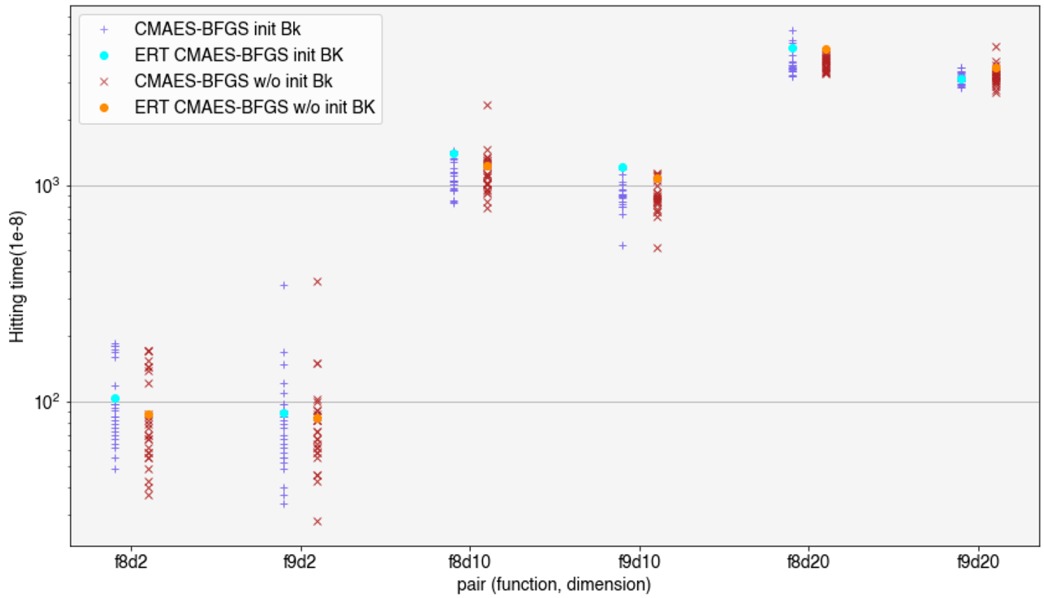


Figure 25: Hitting times to reach $\phi = 10^{-8}$ for individual runs of CMAES-BFGS. Blue plus signs indicate hitting time for runs of CMAES-BFGS with initialized approximate Hessian matrix. Red cross signs indicate hitting time for runs of CMAES-BFGS with approximate Hessian matrix initialized as identity matrix. The resulting ERT values are highlighted with light blue discs (with initial Bk) and orange discs (without initial Bk), respectively.

6.2.3 Validate results

We run the switch from CMA-ES to BFGS with warmstarted x_0 on all 12 use cases derived from the data analysis outlined in Section 5.2. On functions 8 and 9 combined, we achieve an average ERT improvement of 30% over $\text{VBS}_{\text{static}}$, 13% over the expected ERT, respectively. The highest improvement has been achieved on function 9 in dimension 2, accounting for 52%. Only on function 8 in dimension 5, the actual ERT is worse than $\text{ERT}(\text{VBS}_{\text{static}})$ (-16%).

On the two use cases that involve function 1, our approach shows inferior performance. A possible explanation is that as quadratic function, f1 is the ideal case for which BFGS, or generally Newton's method, is designed for, leading to a superlinear convergence rate without any restarting mechanism. Therefore, starting the optimization with CMA-ES does not yield the calculated improvements.

Table 6: ERT values for switching from CMAES to BFGS on selected function-dimension pairs. The target precision has been set to $\phi = 10^{-8}$. Percental improvements are calculated as (ERT($\text{VBS}_{\text{static}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / ERT($\text{VBS}_{\text{static}}$), and (expected ERT($\text{VBS}_{\text{dyn}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / expected ERT($\text{VBS}_{\text{dyn}}$), respectively.

A1	**A2**	**f**	**d**	τ	**ERT ($\text{VBS}_{\text{static}}$)**	**Expected ERT (VBS_{dyn})**	**Actual ERT (VBS_{dyn})**	**ERT impr. actual vs. $\text{VBS}_{\text{static}}$**	**ERT impr. actual vs. expected**
CMA-ES	BFGS	8	2	10	155.08	131.68	**87.52**	44%	34%
CMA-ES	BFGS	8	3	3.98	277.32	242.16	**204.92**	26%	15%
CMA-ES	BFGS	8	5	100	500.83	**472.11**	580.26	-16%	-23%
CMA-ES	BFGS	8	10	100	1963.2	1771.28	**1234.21**	37%	30%
CMA-ES	BFGS	8	20	100	4746.16	**4145.44**	4282.82	10%	-3%
CMA-ES	BFGS	9	2	1.58	175.24	93.92	**84.64**	52%	10%
CMA-ES	BFGS	9	3	2.51	259.24	184.8	**163.36**	37%	12%
CMA-ES	BFGS	9	5	100	551.8	428.56	**301.42**	45%	30%
CMA-ES	BFGS	9	10	100	1645.09	1270.73	**1080.77**	34%	15%
CMA-ES	BFGS	9	20	6.31	4994.96	3745.76	**3502.26**	30%	7%
CMA-ES	BFGS	1	10	100	35.76	**22.00**	47.52	-33%	-116%
CMA-ES	BFGS	1	20	100	74.92	**30.84**	117.4	-57%	-281%

6.2.4 Conclusion

On the Rosenbrock function, we are able to realize performance gains when switching from CMA-ES to BFGS. Our warmstarting procedure simply starts the search of BFGS at the best point found by CMA-ES. Thereby, dynamic algorithm selection enables us to combine distinct advantages of both algorithms, that is, advantageous exploration behaviour by CMA-ES and superior exploitation behaviour by BFGS. However, the performance gains can potentially also be realized by a more efficient sampling method in BFGS to begin with. On most use cases, warmstarting the approximate inverse Hessian matrix B_k in BFGS with the covariance matrix maintained by CMA-ES does not lead to performance improvements.

6.3 MLSL to PSO

Switching from MLSL to PSO accounts for three use cases according to our data analysis from Section 5.2. However, only on function 24 in dimension 2, we expect a substantial ERT improvement when combining both algorithms (69%).[14] Therefore, we focus our experiments on this function-dimension pair, with the switch point set to $\tau = 1.58$.

As shown in Appendix 3, f24 is a multimodal function where the optimum is located within a separate funnel in the solution landscape's global structure. We hypothesize that switching from MLSL to PSO improves ERT compared to $\text{VBS}_{\text{static}}$ because MLSL finds this funnel more efficiently, while PSO is better suited to converge towards the optimum once the swarm arrives at the funnel.

[14] On function 18, dimension 2 the expected ERT improvement accounts for only 1%, on function 20, dimension 5 only 2%, respectively.

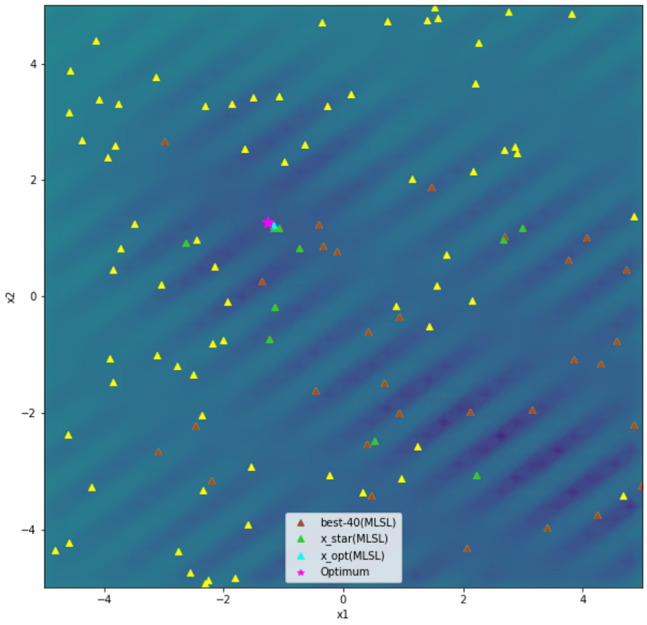


Figure 26: Contour plot of running MLSL on function 24, dimension 2. The algorithm has been stopped after reaching $\tau = 1.58$. Each triangle is a point sampled by MLSL over the course of optimization. The best point found, $x_{\text{opt,MLSL}}$ (cyan), is also part of the points returned by local search (green triangles), which in turn are included in the best 40 points, i.e. the points with the lowest function values (brown triangles). The pink star indicates the location of the global optimum. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1.

6.3.1 Warmstart particle swarm

When switching to the population-based algorithm PSO, we have to initialize the swarm of particles. Without warmstarting, particle positions would be sampled at random in the search space, which is similar to restarting the optimization (see Section 4.2.3). Based on the available information from running MLSL until the switch point, we devise three different approaches to warmstart the particle swarm: (1) set particle positions according to the best n points found by MLSL, where n equals the swarm size, (2) initialize the swarm in the neighbourhood of the best point found by MLSL, $x_{\text{opt,MLSL}}$, or (3) initialize swarm particles in the neighbourhood of the points returned by local search in MLSL, stored in X^* (see Section 4.2.2).

We run both algorithms individually on function 24 to learn more about their search behaviour. Figure 26 illustrates the state of MLSL at the switch point τ. The best point found so far, $x_{\text{opt,MLSL}}$, is located near the global optimum. The points returned by local search X^* as well as the best $n = 40$ points are distributed across the entire search space. As shown in Figure 27 (left), at the switch point, the swarm in PSO is similarly dispensed over the entire search space. However, when reaching target precision, at least part of the swarm accumulates in the area of the global optimum, depicted in Figure 27 (right). Based on these observations, we suggest that the second approach, initializing the swarm in the neighbourhood of $x_{\text{opt,MLSL}}$, yields the highest ERT improvement.

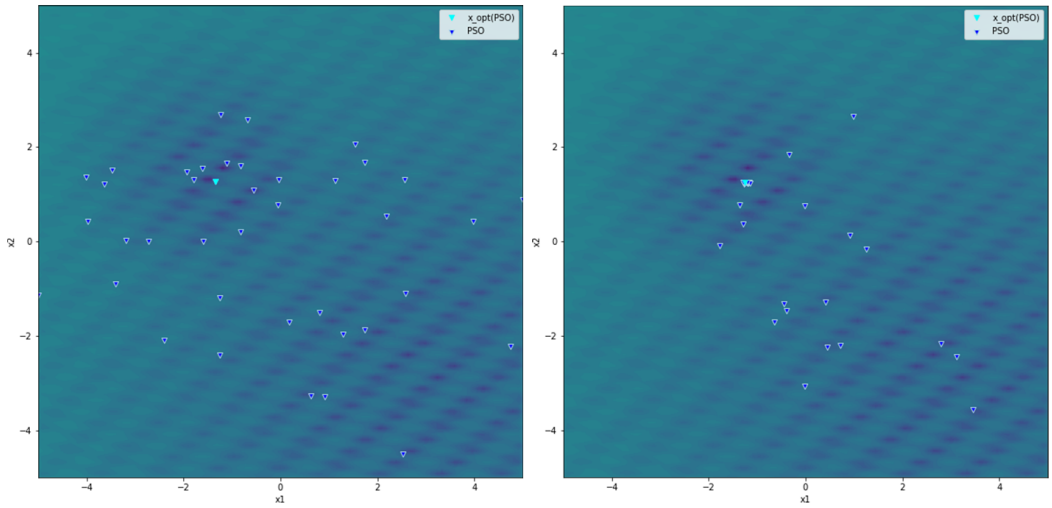


Figure 27: Contour plot of running PSO on function 24, dimension 2. The blue downward triangles indicate the particle positions of the final swarm after reaching $\tau = 1.58$ (left image) and $\phi = 10^{-8}$ (right image). Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1.

To test this hypothesis, we implement the *neighbourhood method* according to Algorithm 9. The particle positions are initialized around $x_{\text{opt,MLSL}}$ within a radius η. Early experiments provided us with the intuition to set $\eta = 0.1$. The particle velocities are set randomly in $[-1, 1]^d$, equivalent to the settings outlined in Section 4.3.1. We run MLSL-PSO with this configuration on function 24 in dimension 2 according to the experimental setting outlined in Section 5.1. As indicated by the orange line in Figure 29, the resulting ERT is considerably lower than ERT($\text{VBS}_{\text{static}}$) (86% improvement), which is even 55% better than the expected ERT. A notable observation from the data is that this improvement can be explained by the number of successful runs, which accounts for 18 out of 25 for the switch, but only 4 for PSO, and 0 for MLSL, respectively. Therefore, combining MLSL and PSO with the outlined warmstarting procedure does not only reduce the number of required function evaluations, but also increases the probability to reach target precision.

Figure 28 illustrates the contour plot for switching from MLSL to PSO. Within two iterations, MLSL reaches the switch point close to the global optimum. The swarm is subsequently initialized in the same area. In the first few PSO iterations, some swarm particles move to distant areas in the search space. However, the majority of particles stay near the global optimum and iteratively converge closer to it, until target precision is hit. Compared to Figure 27 (right), it becomes apparent that in MLSL-PSO, the entire swarm exploits the basin of attraction at convergence, rather than just part of the swarm when running PSO individually. This could explain the increased success probability to hit target precision when combining both algorithms.

We compare this result with warmstarting the swarm according to the best n points found by MLSL. For this method, the particle positions x_i are set to the respective x-values of the $n = 40$ points with the lowest function values returned by MLSL (brown triangles in Figure 26). Even though we achieve an improved ERT on function 24, dimension 2 (75%), indicated by the light green line in Figure 29, the improvement is not as high as with the neighbourhood method.

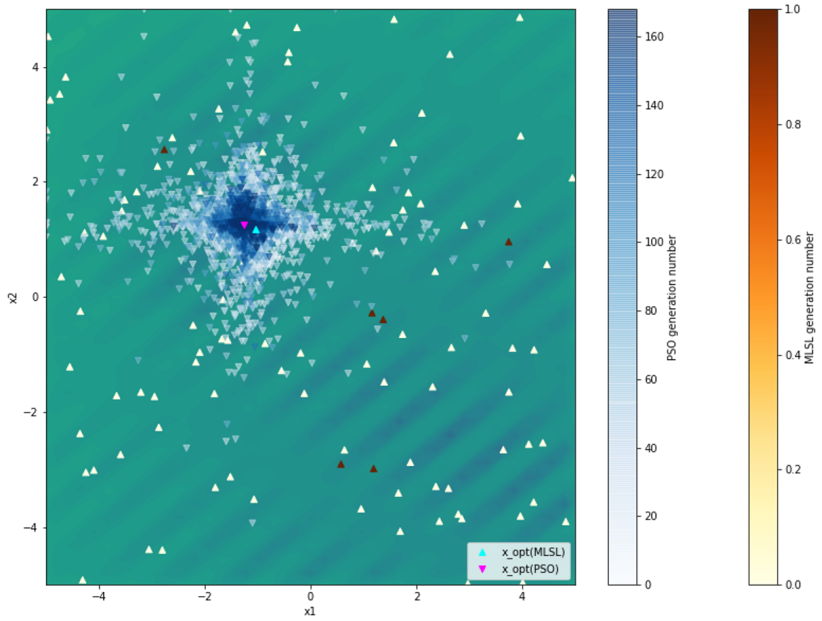


Figure 28: Contour plot of running MLSL-PSO with the warmstarting procedure outlined in Algorithm 9 on function 24, dimension 2, $\tau = 1.58$, $\phi = 10^{-8}$. Upward triangles indicate points sampled by MLSL, while downward triangles indicate points sampled by PSO. Darker yellow-brown and blue colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1. The target precision has been reached at the magenta colored point.

Algorithm 9 Warmstarting routine MLSL-PSO

procedure WARMSTART PSO($s_t(MLSL)$, $\mathcal{H}_t$)
  $\eta \leftarrow 0.1$ ▷ population distribution radius
  for $i \leftarrow 1$ **to** swarm size **do**
    $x_i \leftarrow x_{\text{opt,MLSL}} +$ random vector in $[-\eta, \eta]^d$ ▷ particle position
    $v_i \leftarrow$ random vector in $[v_{0,min}, v_{0,max}]^d$ ▷ particle velocity
    if $f_i < f_{gbest}$ **then**
      $f_{gbest} \leftarrow f_i$
      $x_{gbest} \leftarrow x_i$
    end if
  end for
end procedure

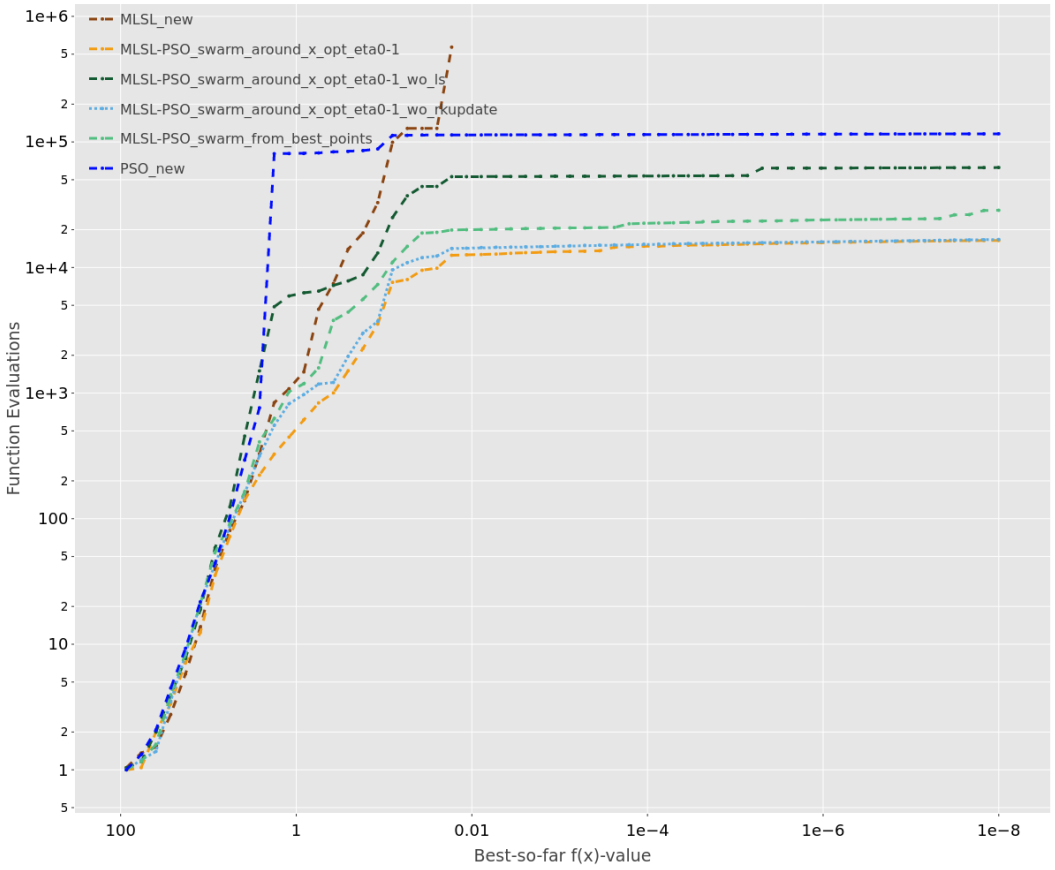


Figure 29: ERT charts for MLSL, PSO and MLSL-PSO with different warmstarting routines on function 24, dimension 2, $\tau = 1.58$. All algorithms were run according to the experimental settings outlined in Section 5.1.

6.3.2 Impact of clustering and local search

A possible explanation for the observed performance improvement is that MLSL finds the basin of attraction, or *funnel*, more efficiently than PSO. However, it is also imaginable that the same results can be achieved by an improved initialization method in PSO. Put differently, the question is if the unique features of MLSL contribute to the performance advantages of this particular algorithm combination on function 24 in dimension 2. Therefore, we conduct the following experiments.

First, the local search routine in MLSL (Algorithm 3, lines 9–15) is disabled. Effectively, MLSL in this configuration just iteratively samples $N = 50d$ random points in the search space until a target precision of τ is hit. We run the combination MLSL-PSO with this configuration and the warmstarting procedure outlined in Algorithm 9. The resulting ERT curve is depicted by the dark green line in Figure 29. Even though ERT is improved by 46% compared to ERT($\text{VBS}_{\text{static}}$), it is almost four times higher than running MLSL-PSO with local search routine turned on.

Second, we deactivate the clustering mechanism in MLSL by setting $r_k = 0$. As a result, a local search routine starts at every point included in the reduced sample X_r, not only at the best point in each cluster. Then, we run MLSL-PSO with the outlined warmstarting procedure on function 24 in dimension 2. The resulting ERT, as illustrated by the light blue line in Figure 29, only slightly deviates from the ERT we achieved with MLSL-PSO where clustering is turned on.

Based on these observations, we conclude that the local search routine in MLSL is contributing to the performance gains obtained when switching from MLSL to PSO, while the clustering mechanism does not seem to have a substantial impact on ERT.

6.3.3 Warmstart particle velocities

Another aspect to consider when warmstarting PSO is how to set the initial particle velocities. Without warmstarting, particle velocities are set randomly in $[-1, 1]^d$ (see Section 4.3.1). Our previous experiments on function 24 have shown that when switching from MLSL to PSO, the swarm is initialized within the basin of attraction. Therefore, the first intuition is to set lower particle velocities after the switch as compared to running PSO individually, since slower particles are more likely to stay within that area.

We run MLSL-PSO with the warmstarting procedure outlined in Algorithm 9 on function 24 in dimension 2, while changing the initial velocities determined by $[v_{0,min}, v_{0,max}]$. Table 7 lists the resulting ERT values. Setting lower particle velocities after the switch, e.g. $v_0 \in [-0.1, 0.1]$, leads to an increased ERT compared to our default settings. A possible explanation is that even though slower particles stay within the basin of attraction, the likelihood to hit target precision is diminished. Higher initial particle velocities, e.g. $v_0 \in [-5, 5]$, yield increased ERT values as well.

Table 7: ERT values for different initial velocity settings in MLSL-PSO on function 24, dimension 2, $\tau = 1.58$, $\phi = 10^{-8}$. All algorithms were run according to the experimental settings outlined in Section 5.1.

$[v_{0,min}, v_{0,max}]$	ERT(10^{-8})
$[-0.1, 0.1]$	44067.44
$[-0.5, 0.5]$	20149.69
$[-1, 1]$	16435.78
$[-2, 2]$	28554.92
$[-5, 5]$	27795.31

6.3.4 Compare to MLSL-CMAES and MLSL-DE

When switching from MLSL to PSO on function 24, the basin of attraction is found by MLSL, which enables us to initialize the population in PSO in the respective neighbourhood. Based on these properties, it is reasonable to assume that similar results can be obtained when switching to another population-based algorithm with good convergence properties. In other words, by switching from MLSL to other algorithms on the same use case, we can examine if the previously observed performance improvements are related specifically to the algorithmic procedure in PSO.

To begin with, we experiment with switching from MLSL to CMA-ES. Similar to our approach in Section 6.1.1, we set the distribution mean at the switch point $m^{(\tau)} = x_{\text{opt,MLSL}}$. By default, CMA-ES will sample the initial population around m. The other parameters are kept in their original settings, that is, $\sigma^{(\tau)} = 0.5$ and $C^{(\tau)} = I$. We run this configuration on function 24 in dimension 2, with $\tau = 1.58$ and $\phi = 10^{-8}$. With only 2 successful runs, the resulting ERT is worse than ERT($\text{VBS}_{\text{static}}$) (-103%), thus far worse than the ERT when switching from MLSL to PSO. Depicted in Figure 30 (left), during successful runs, CMA-ES is able to stay within the area of the global optimum while reaching target precision after a few iterations. However, in most of the runs, CMA-ES departs from that area and performs many iterations, repeatedly updating the covariance matrix and step size without hitting target precision, as illustrated by Figure 30 (right).

Next, we combine MLSL with DE. Similar to the *neighbourhood method* outlined in Section 6.3.1, we initialize the population in DE around the optimal point found by MLSL with $x_i = x_{\text{opt,MLSL}}+$ random vector in $[-\eta, \eta]^d$. We set $\eta = 0.1$ and run this configuration on function 24 in dimension 2, with $\tau = 1.58$ and $\phi = 10^{-8}$. The resulting ERT is twice as high as ERT($\text{VBS}_{\text{static}}$) (99.9% performance deterioration), thus again inferior to setting PSO as A_2. Figure 31 shows a successful MLSL-DE run. The initial population is sampled around the best point found by MLSL, as we intended, and target precision is hit a few iterations after the switch. The problem in unsuccessful runs, as illustrated by Figure 32, is that $x_{\text{opt,MLSL}}$ is not located on the same ridge of decreasing function values as the global optimum. As a consequence, DE does not reach the correct ridge and samples a multitude of points in an incorrect location, until the evaluation budget runs out. This observation could explain the unique advantages of switching to PSO on this use case: Due to the initially high particle velocities and the resulting vibrant swarm behaviour, the correct ridge is found, even if the center of the initial population, determined by $x_{\text{opt,MLSL}}$, is located elsewhere.

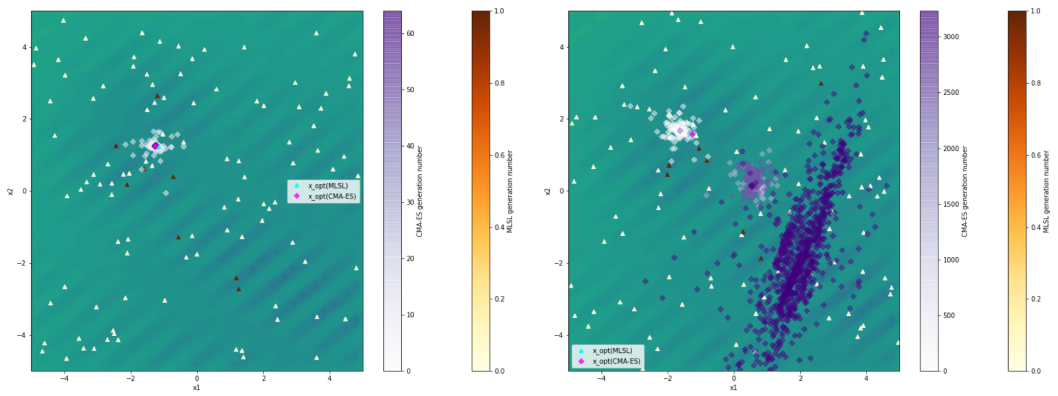


Figure 30: Contour plot of running MLSL-CMAES on function 24, dimension 2, $\tau = 1.58$, $\phi = 10^{-8}$. Triangles indicate points sampled by MLSL, while diamonds indicate points sampled by CMA-ES. Darker yellow-brown and purple colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1. Left image shows a successful run, i.e. target precision has been reached at the magenta colored point. The right image shows an unsuccessful run.

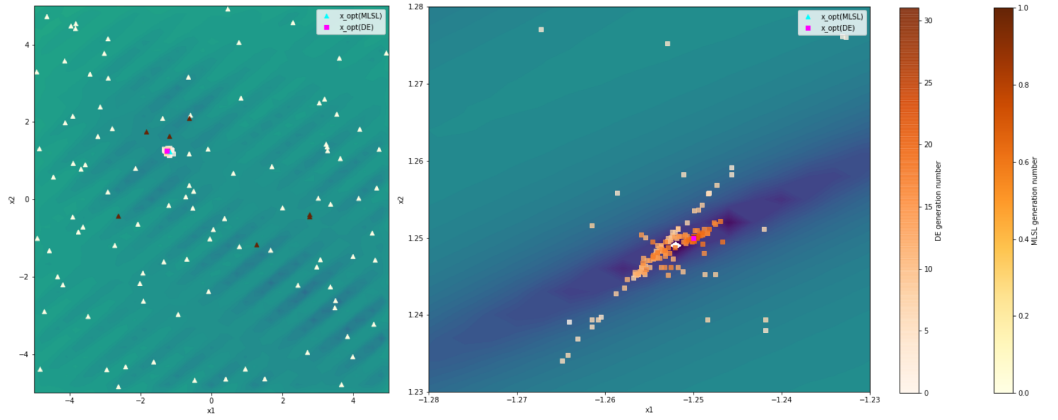


Figure 31: Contour plot of running MLSL-DE on function 24, dimension 2, $\tau = 1.58$, $\phi = 10^{-8}$. Triangles indicate points sampled by MLSL. Squares indicate points sampled by DE. Darker yellow-brown and orange colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1. Left image shows entire search space, while right image zooms in on the area of the best sampled point. Target precision has been hit at the magenta colored point.

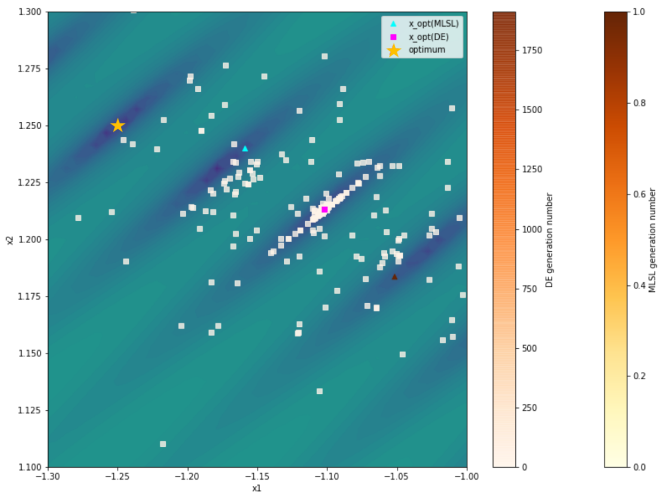


Figure 32: Contour plot of an unsuccessful run of MLSL-DE on function 24, dimension 2, $\tau = 1.58$, $\phi = 10^{-8}$. Triangles indicate points sampled by MLSL. Squares indicate points sampled by DE. Darker yellow-brown and orange colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The star indicates the location of the global optimum. The plot has been created by 1 run on instance 1.

6.3.5 Validate results

To validate our findings, we run MLSL-PSO on additional use cases. Since our data analysis did not return more than the previously mentioned function-dimension pairs, the procedure outlined in Section 5.2 is repeated while limiting the algorithm portfolio to only include MLSL and PSO. It must be noted that as a result, $\text{VBS}_{\text{static}}$ is either MLSL or PSO instead of any of the five algorithms within our portfolio.

Table 8 lists the ERT values for running MLSL-PSO according to the warmstarting procedure outlined in Algorithm 9 on selected use cases. Averaged over all function-dimension pairs, the ERT is improved by 45%. A notable observation is that improvements are achieved for various functions from different function groups, such as multimodal functions with weak global structure (e.g. f23) and multimodal functions with adequate global structure (e.g. f16). On function 21 in dimension 5, the highest improvement is recorded, accounting for 89%. Only on function 15 in dimension 2 and function 24 in dimension 3, ERT(MLSL-PSO) is worse than $\text{ERT}(\text{VBS}_{\text{static}})$, which may be related to the high values for τ in these use cases.

Compared to the expected improvement, the results are mixed. On some use cases, the actual improvement is substantially better than the expected value, such as on function 23 in dimension 2 or on function 21 in dimension 5. On other use cases, we are not able to achieve the predicted ERT improvement. For example, on function 21 in dimension 20, the actual $\text{ERT}(\text{VBS}_{\text{dyn}})$ is 83% better than $\text{ERT}(\text{VBS}_{\text{static}})$, but 97% worse than the expected ERT. This raises the question how accurately a data-driven approach based on ERT values is able to predict the empirical performance of dynamic algorithm selectors.

Table 8: ERT values for switching from MLSL to PSO on selected function-dimension pairs. The target precision has been set to $\phi = 10^{-8}$. The data analysis has been limited to PSO and MLSL. Therefore, $\text{VBS}_{\text{static}}$ is either of the two algorithms. Percental improvements are calculated as ($\text{ERT}(\text{VBS}_{\text{static}})$ - actual $\text{ERT}(\text{VBS}_{\text{dyn}})$) / $\text{ERT}(\text{VBS}_{\text{static}})$, and (expected $\text{ERT}(\text{VBS}_{\text{dyn}})$ - actual $\text{ERT}(\text{VBS}_{\text{dyn}})$) / expected $\text{ERT}(\text{VBS}_{\text{dyn}})$, respectively.

A1	**A2**	**f**	**d**	τ	**ERT ($\text{VBS}_{\text{static}}$)**	**Expected ERT (VBS_{dyn})**	**Actual ERT (VBS_{dyn})**	**ERT impr. actual vs. $\text{VBS}_{\text{static}}$**	**ERT impr. actual vs. expected**
MLSL	PSO	2	3	$2.51 \cdot 10^{-4}$	8003.16	**2176.36**	6061.08	24%	-178%
MLSL	PSO	2	20	$3.98 \cdot 10^{-4}$	203640.71	**7418.79**	50493.52	75%	-581%
MLSL	PSO	12	2	$3.98 \cdot 10^{-5}$	17320.85	14447.14	**9139.72**	47%	37%
MLSL	PSO	15	2	1.58	6016.24	**5532.4**	6340	-5%	-15%
MLSL	PSO	16	2	$1.00 \cdot 10^{-2}$	7030.21	**4829.68**	5829.67	17%	-21%
MLSL	PSO	17	2	$1.58 \cdot 10^{-1}$	7639.88	**7213.4**	7443.16	3%	-3%
MLSL	PSO	20	2	$3.98 \cdot 10^{-1}$	7647.52	5706	**5038**	34%	12%
MLSL	PSO	21	5	$6.31 \cdot 10^{-1}$	49843.62	12614.84	**5322.32**	89%	58%
MLSL	PSO	21	20	$6.31 \cdot 10^{-5}$	1507443	**133682.63**	263427.17	83%	-97%
MLSL	PSO	22	5	$3.98 \cdot 10^{-1}$	42582.86	12528.71	**11127**	74%	11%
MLSL	PSO	23	2	$2.51 \cdot 10^{-1}$	92379.4	87941.44	**21294.72**	77%	76%
MLSL	PSO	24	2	1.58	116140.75	36253.11	**16435.78**	86%	55%
MLSL	PSO	24	3	2.51	749916	**662482.46**	893445	-19%	-35%

6.3.6 Conclusion

By warmstarting the initial population in the neighbourhood of the best point found by A_1, we achieve substantial ERT improvements when switching from MLSL to PSO on several use cases. This *neighbourhood method* is not only applicable when A_1 maintains a population, but for single solution algorithms like BFGS as well. Moreover, it is potentially transferable to other population based algorithms. With the warmstarting procedure outlined in Algorithm 9, we improved ERT up to 89% compared to $\text{ERT}(\text{VBS}_{\text{static}})$. In particular, our approach proved to be effective on function 24, which poses the special challenge for an algorithm to converge within a certain funnel in the global solution landscape. Therefore, this approach may be practicable on problems with similar solution landscape structures.

For the warmstarting procedure, the distribution radius parameter η has been introduced. We expect that tuning this parameter will further improve performance. According to our experiments, particle velocities have to be considered for warmstarting to achieve optimal performance as well. However, for the use cases presented here, the default settings already resulted in the highest ERT improvements.

Another key finding is that MLSL appears often as A_1 algorithm in dynAS combinations (see Section 4.1) because the algorithm is able to identify areas worth exploiting in the solution landscape, especially by performing its local search routine. Finally, our experiments revealed that ERT improvements in dynamic algorithm selection may not only be related to a reduction in required function evaluations, but also to an increased success probability to reach target precision for individual runs.

6.4 PSO to DE

From all algorithm combinations that include DE as A_2 algorithm, switching from PSO to DE accounts for the highest number of use cases (see Section 5.2). We start our experiments on the multimodal function 17 in dimension 2 with the switch point set to $\tau = 2.51 \cdot 10^{-5}$.

6.4.1 Impact of population distribution radius

Since all other parameters are constants or based on per-iteration calculations, the only parameter to consider for warmstarting DE is the initial population. Again, we employ the neighbourhood method introduced in Section 6.3.1. That is, initial solution candidates are sampled around the best point found by PSO, $x_{\text{opt,PSO}}$, within a radius η. We set $\eta = 0.1$ and run PSO-DE on function 17 in dimension 2 according to the settings outlined in Section 5.1. As illustrated in Figure 35, the resulting ERT curve (dark green) leaps after the switch point, such that the empirical performance to reach target precision is worse than both PSO and DE run separately.

Further data analysis reveals that PSO-DE in this configuration reaches target precision only in 3 out of 25 runs. By comparing the contour plots of successful and unsuccessful runs, we learn why the current warmstarting procedure leads to inferior performance. As depicted in Figure 33, during successful runs, the DE population is initialized around $x_{\text{opt,PSO}}$ and stays within that area, reaching target precision within a few generations. In most of the runs however, DE fails to hit target precision despite initialization in the correct area. As shown in Figure 34, in unsuccessful runs, DE repeatedly samples points in an area close to the global optimum, exploiting a local optimum instead. As outlined in Section 4.2.5, being trapped in a local optimum is a known weakness of differential evolution. On this specific use case, this is likely caused by the highly asymmetric shape of the solution landscape of function 17.

Figure 34 also highlights that the best point found by PSO is sampled in the basin of attraction. Therefore, by lowering the population distribution radius η, the DE population is initialized closer to that point, thus exploiting this specific area rather than diverging to an immediate neighbourhood. We set $\eta = 10^{-5}$ and repeat the previous experimental procedure. With the adapted distribution radius, the switch PSO-DE reaches target precision in all 25 runs. The resulting ERT, as indicated by the red line in Figure 35, is 24% better than ERT($\text{VBS}_{\text{static}}$), which is 2 percentage points better than the expected ERT improvement.

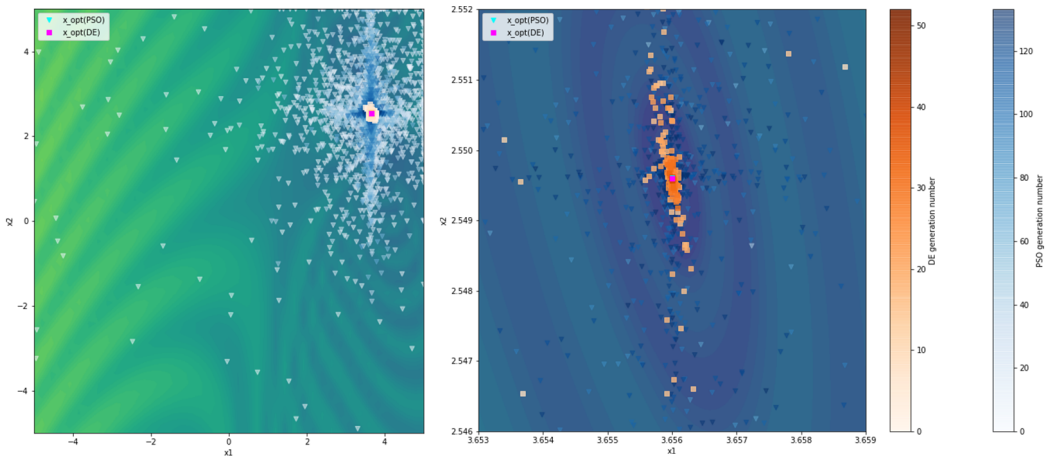


Figure 33: Contour plot of a successful run of PSO-DE on function 17, dimension 2, $\tau = 2.51 \cdot 10^{-5}$, $\phi = 10^{-8}$. Downward triangles indicate points sampled by PSO. Squares indicate points sampled by DE. Darker orange and blue colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1. Left image shows the entire search space, while rights image zooms in on the area of the global optimum.

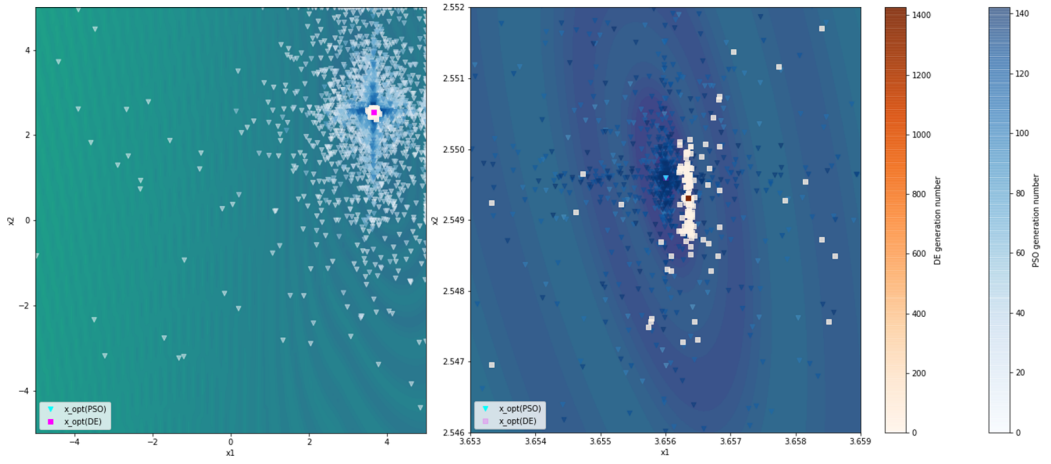


Figure 34: Contour plot of an unsuccessful run of PSO-DE on function 17, dimension 2, $\tau = 2.51 \cdot 10^{-5}$. The target precision of $\phi = 10^{-8}$ has not been reached. Downward triangles indicate points sampled by PSO. Squares indicate points sampled by DE. Darker orange and blue colors indicate higher generation numbers. Darker green-blue colors indicate smaller function values. The plot has been created by 1 run on instance 1. Left image shows the entire search space, while rights image zooms in on the area of the global optimum.

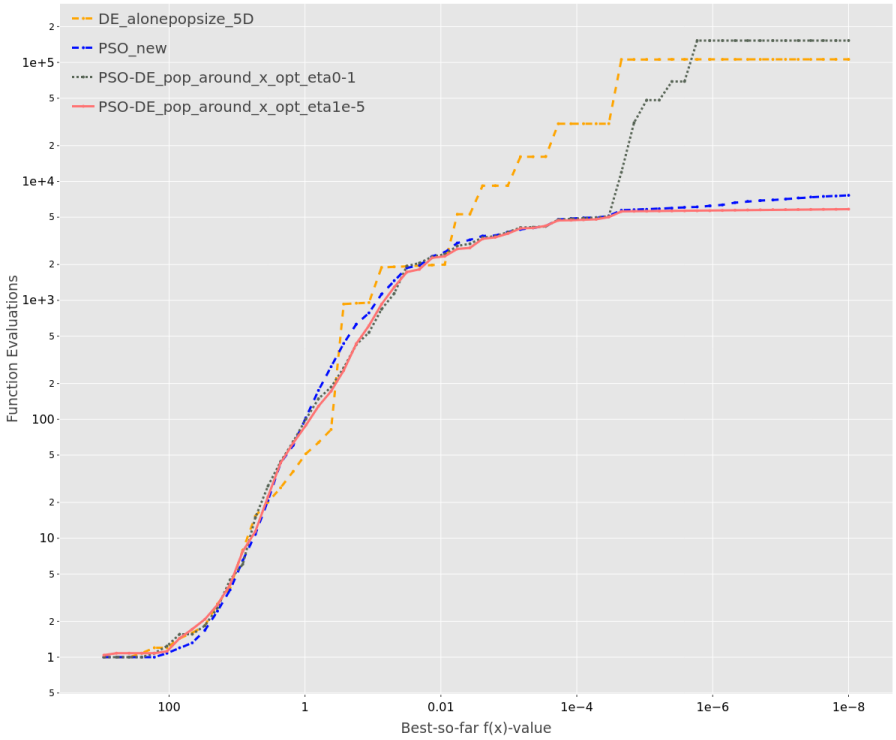


Figure 35: ERT charts for PSO, DE and PSO-DE with warmstarting the initial population in the neighbourhood of $x_{\text{opt,PSO}}$, $\eta = 0.1$ (dark green line) and $\eta = 10^{-5}$ (red line) on function 17, dimension 2, $\tau = 2.51 \cdot 10^{-5}$. All algorithms were run according to the experimental settings outlined in Section 5.1.

6.4.2 Validate results

To verify that warmstarting DE with the outlined procedure results in ERT improvements, we run PSO-DE on all five uses cases with different values for η. Table 9 lists the resulting ERT values. On all use cases in dimension 2, we achieve performance improvements, with up to 92% improvement on function 22. Compared to the expected improvements, the results only slightly deviate, i.e. in the range between -6% and 8%.

On function 7 in dimension 3 however, switching from PSO to DE does not yield the predicted performance gains. Even with different values for η, i.e. $\eta = 0.01$ and $\eta = 0.5$, the ERT of PSO-DE is higher than ERT($\text{VBS}_{\text{static}}$).

Table 9: ERT values for switching from PSO to DE on selected function-dimension pairs. The target precision has been set to $\phi = 10^{-8}$. Percental improvements are calculated as (ERT($\text{VBS}_{\text{static}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / ERT($\text{VBS}_{\text{static}}$), and (expected ERT($\text{VBS}_{\text{dyn}}$) - actual ERT($\text{VBS}_{\text{dyn}}$)) / expected ERT($\text{VBS}_{\text{dyn}}$), respectively.

A1	**A2**	**f**	**d**	τ	η	**ERT ($\text{VBS}_{\text{static}}$)**	**Expected ERT (VBS_{dyn})**	**Actual ERT (VBS_{dyn})**	**ERT impr. actual vs. $\text{VBS}_{\text{static}}$**	**ERT impr. actual vs. expected**
PSO	DE	3	2	$6.31 \cdot 10^{-1}$	10^{-1}	5668.56	1894.23	**1735.88**	69%	8%
PSO	DE	7	2	$3.98 \cdot 10^{-3}$	10^{-1}	2752.12	1254.65	**1208.08**	56%	4%
PSO	DE	7	3	$1.00 \cdot 10^{-2}$	10^{-1}	4518.76	**3488.29**	8739.14	-93%	-151%
PSO	DE	7	3	$1.00 \cdot 10^{-2}$	10^{-2}	4518.76	**3488.29**	10548.14	-133%	-202%
PSO	DE	7	3	$1.00 \cdot 10^{-2}$	$5 \cdot 10^{-1}$	4518.76	**3488.29**	10435.4	-131%	-199%
PSO	DE	17	2	$2.51 \cdot 10^{-5}$	10^{-1}	7639.88	**5930.37**	152803	-1900%	-2477%
PSO	DE	17	2	$2.51 \cdot 10^{-5}$	10^{-5}	7639.88	5930.37	**5844.04**	24%	1%
PSO	DE	22	2	$6.31 \cdot 10^{-1}$	10^{-1}	3343.28	**256.03**	270.84	92%	-6%

6.4.3 Impact of switch point

We further examine the performance data obtained from running PSO, DE and PSO-DE on function 7 in dimension 3 to understand why the previous approach did not return the expected improvements. In particular, we zoom in on the individual runs for DE and PSO, illustrated in Figure 36. It becomes apparent that within the range of target precisions $\phi \in [10^{-2}, 10^{-6}]$, the blue lines, indicating hitting times of individual PSO runs, progress horizontally. That is, even after hitting the calculated switch point $\tau = 10^{-2}$ for this use case, PSO only requires very few function evaluations to reach lower target precision values. Only after approaching $\phi = 10^{-6}$, hitting times in PSO leap, whereas in DE, target precision is reached without a vast amount of additional function evaluations. The only reason why ERT(DE) is worse than ERT(PSO) is that DE reaches target precision with only 22 out of 25 runs, while PSO hits target precision with every run.

Keeping in mind that DE is prone to being trapped in local optima, these observations lead to the idea that switching later than the calculated switch point, that is, lowering the value of τ, may improve the performance of PSO-DE on this use case. Without many additional function evaluations, a lower target precision is reached by PSO, such that DE is more likely to be initialized near the basin of attraction. On the other hand, since proximity to the optimum in function value does not necessarily relate to the respective proximity in the variables space, DE might still exploit a local optimum instead.

We test this approach by running PSO-DE on function 7 in dimension 3 while decreasing the switch point to $\tau = 10^{-5}$ and setting $\eta = 0.1$. As illustrated by the red line in Figure 37, switching later indeed exhibits a positive effect on ERT. The ERT(PSO-DE) with $\tau = 10^{-5}$ accounts for 3865.68, which is 56% better than the same combination with $\tau = 10^{-2}$, and even accounts for a 14% improvement compared to ERT($\text{VBS}_{\text{static}}$).

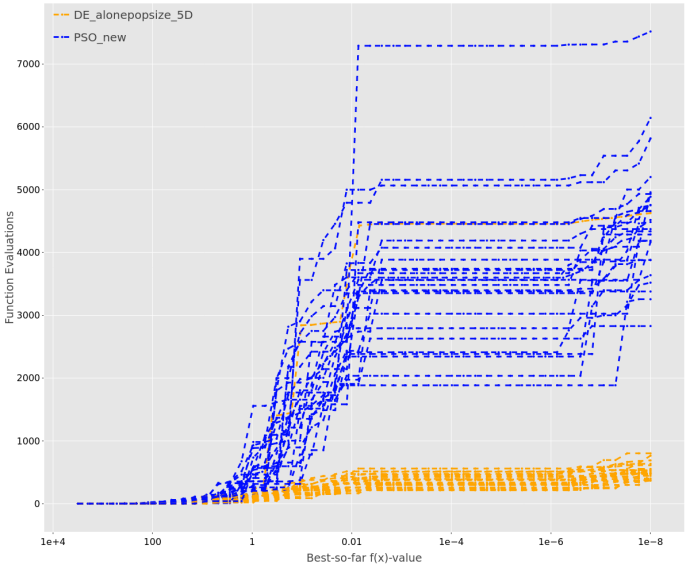


Figure 36: Hitting times for individual runs of PSO and DE on function 7, dimension 3. All algorithms were run according to the experimental settings outlined in Section 5.1.

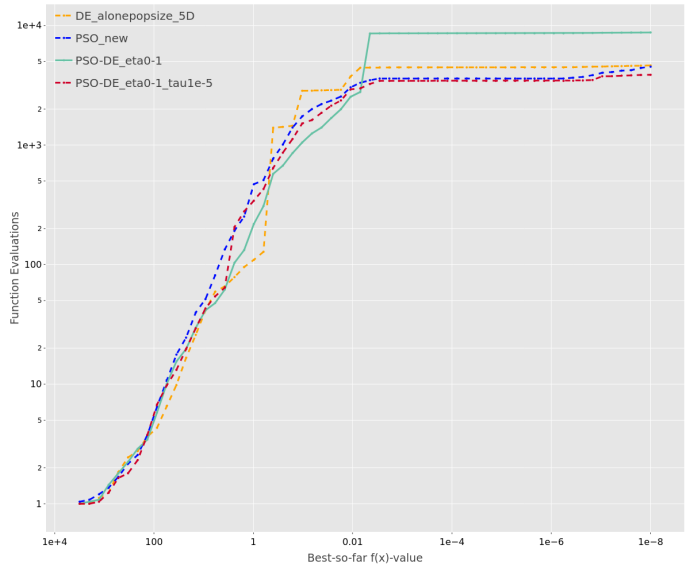


Figure 37: ERT charts for PSO, DE and PSO-DE with warmstarting the initial population in the neighbourhood of $x_{\text{opt, PSO}}$, with $\eta = 0.1$, on function 7 in dimension 3. The switch point is set to $\tau = 10^{-2}$ (green line), $\tau = 10^{-5}$ (red line), respectively. All algorithms were run according to the experimental settings outlined in Section 5.1.

6.4.4 Conclusion

By initializing the population in DE in the neighbourhood of the best point found by PSO after the switch, we were able to realize substantial performance improvements for all five function-dimension pairs obtained from the data analysis in Section 5.2. The experiments have demonstrated that setting warmstarting parameters like the distribution radius η is key to achieve these improvements, which underscores the importance of tuning.

As expected from the data analysis in Section 4.1, DE performs well as A_2 algorithm in a dynAS combination, mainly due to its exploiting search behaviour. Therefore, we suspect that differential evolution will prove efficient in algorithm combinations with other A_1 algorithms as well. Likewise, it is imaginable that similar results can be achieved when switching from PSO to other algorithms with good convergence properties.

Finally, our experiments show that the selection of the switch point τ affects the performance of a dynAS model. In some cases, it may prove useful to switch earlier or later compared to the switch point derived from the empirical performance data. Inspecting individual algorithm runs revealed that the optimal switch point probably even differs depending on the optimization progress during each run.

7 Discussion of results

In this work, a diverse portfolio of five algorithms has been assembled and run on the BBOB test suite. Afterwards, 87 use cases, consisting of function-dimension pairs and the respective switch points, have been derived from the empirical performance data. By examining four algorithm combinations in more detail, we have realized ERT improvements of up to 92% and developed warmstarting procedures for all algorithms but MLSL. The experiments have revealed that warmstarting algorithmic parameters after the switch is crucial to realize performance improvements, especially initializing the population, or first sample point, respectively. Furthermore, this work has shown that on several use cases, chaining algorithms with a fixed-target approach returns a merged ERT curve, following the course of A_1 until the switch point, after which it resumes the course of A_2. Our experiments have also highlighted that hyperparameter tuning as well as changing the switch point impact the performance of a dynamic algorithm combination.

The complete experimental results from running dynAS on all identified use cases can be found at Schröder (2021). Appendix 4 provides a summary of the most important performance metrics for each use case. Out of the 87 use cases (see Section 5.2), we run a single-switch algorithm combination on 83 function-dimension pairs. The remaining four use cases include MLSL as A_2 algorithm, for which no warmstarting routine has been developed yet.

Figure 38 depicts the resulting ERT improvements of VBS_{dyn} over $\text{VBS}_{\text{static}}$. It shows that for the functions f8–f14, substantial improvements were achieved across various dimensions, based on the bidirectional combination of BFGS and CMA-ES. However, the same combination did not reach the calculated performance improvements on multimodal functions, such as f21 and f22 in dimensions 5 and 10, which is likely related to the large value for τ in these use cases. The early switch point does not only lead to an isotropic covariance matrix at the switch, but also increases the risk of BFGS to reach the switch point at a local optimum.

On other multimodal functions, for example f15–f19, dynAS led to superior performance only in lower dimensions. As discussed in Section 4.4, a possible explanation is that the algorithms within our portfolio have difficulties to reach target precision on this particular function group in higher dimensions.

By comparing the actual to the potential improvements shown in Figure 13, we can identify examples of function-dimension pairs where an algorithm switch performed either better or worse than expected. For instance, on f19 in dimension 3, switching from PSO to CMA-ES led to a 70% ERT improvement, which is 19 percentage points higher than the calculated improvement. Likewise, on function 4 in dimension 5, switching from DE to PSO resulted in a 33% improvement, which is eleven times better than the expected value. In both examples, the positive results were achieved even though we did not investigate these algorithm combinations (in this order) in more detail, which means the warmstarting procedures presented here are applicable for more algorithm combinations than just the ones outlined in Section 6. As a negative example, switching from PSO to CMA-ES on function 16 in dimension 3 resulted in a 10% performance decrease, compared to an expected improvement

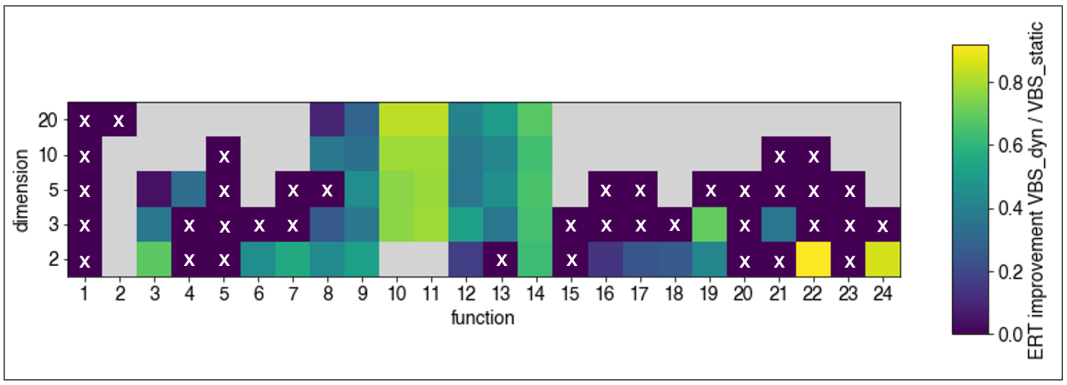


Figure 38: Actual ERT improvements of VBS_{dyn} over $\text{VBS}_{\text{static}}$ within our portfolio for all functions and dimensions. Percental improvement is calculated as (ERT($\text{VBS}_{\text{static}}$) - ERT($\text{VBS}_{\text{dyn}}$)) / ERT($\text{VBS}_{\text{static}}$). Improvements are capped at 0. Boxes with 'x' indicate function-dimension pairs where ERT(VBS_{dyn}) decreased performance, or ERT $= \infty$. Grey boxed indicate function-dimension pairs that were not part of any use case according to Section 5.2. Algorithms were warmstarted according to the settings outlined in the validation sub-sections of Section 6, except for CMA-ES, where we applied the threshold method with $\epsilon = 0.1$.

of 49%. In total, the dynAS procedure outperformed the portfolio's virtual best static solver on 45 use cases.

Figure 39 sheds light on the performance of different combinations of A_1 and A_2 algorithms. Unsurprisingly, the ERT speed-up peaks for the algorithm combinations that we have investigated in more detail in Section 6. We assume that studying the remaining algorithm combinations and use cases in more depth will lead to similarly positive results. As a promising example, switching from PSO to CMA-ES already leads to an average speed up larger than 1, even though we have only applied the warmstarting routine developed for a different combination.

From Figure 39, we cannot infer which algorithms are generally favorable candidates as A_1 or A_2 algorithms, since the data for each combination is not only limited to certain functions and dimensions, but also differs in the amount of available use cases. However, it is worth noting that from all combinations that include BFGS as A_2, only CMAES-BFGS resulted in improvements. Thus, switching towards BFGS in our fixed-target approach is not as straightforward as in fixed-budget approaches, where similar algorithms are applied at the end of optimization to further exploit the optimum. For switching from MLSL to BFGS this finding matches our expectations, since MLSL already runs a similar algorithm within its local search phase. Further data analysis shows that for the remaining use cases, i.e. when switching from PSO or DE, the switch happens early, with $\tau > 1.0$. This is problematic due to the already discussed weakness of BFGS to be trapped in local optima.

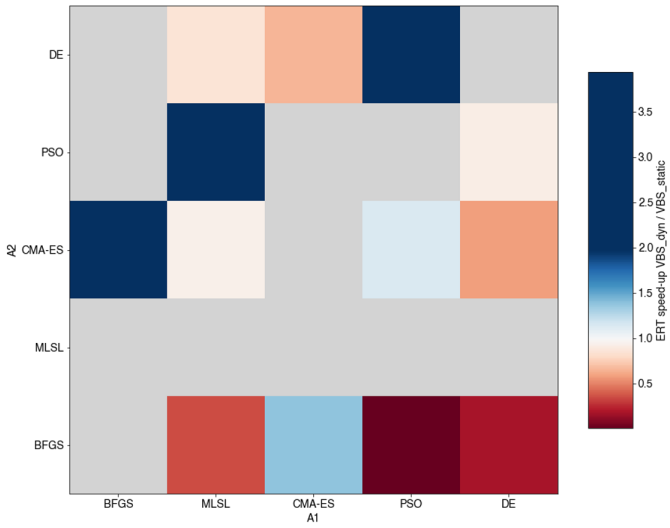


Figure 39: Heatmap indicating the ERT speedup (ERT($\text{VBS}_{\text{static}}$) / ERT($\text{VBS}_{\text{dyn}}$)) per A_1–A_2 algorithm combination, averaged over all available function-dimensions pairs for the respective combination (see experimental results in Appendix 4). Grey boxes indicate combinations where no experimental results are available.

Lastly, Figure 40 provides an overview of the performance of VBS_{dyn} over $\text{VBS}_{\text{static}}$ across all BBOB functions in dimensions 2 (top) and 10 (bottom). Moreover, the charts indicate the difference between calculated ERT values (see Section 5.2) and the actually achieved values from our experiments. It becomes apparent that on some function-dimension pairs, the actual results almost perfectly match the expected values, for example on f6, f7 and f22 in dimension 2, and on f10 in dimension 10. However, on most use cases, we observe a discrepancy between both values. For instance, on f4 and f15 in dimension 2, the ERT of VBS_{dyn} is much higher than the calculated ERT, while on f14 and f24 in dimension 2, it is lower. In conclusion, even though performance improvements were achieved for various use cases and the respective algorithm combinations and switch points, the accuracy of the data-driven approach applied in this work needs to be further investigated.

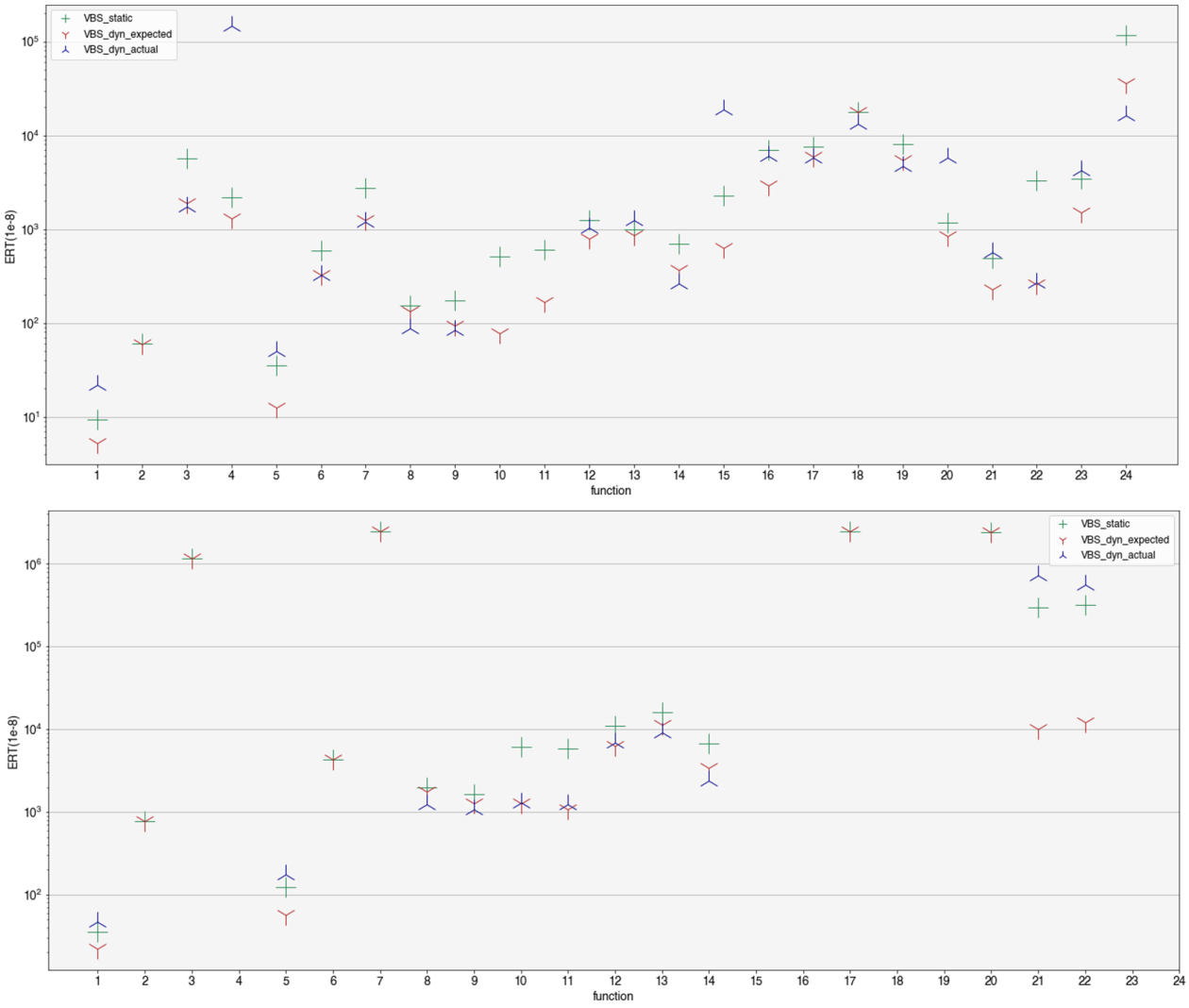


Figure 40: ERT values for $\text{VBS}_{\text{static}}$ (green *plus*), VBS_{dyn} actual (blue *tri*) and VBS_{dyn} expected (red *tri*) on all BBOB functions in dimension 2 (top image) and dimension 10 (bottom image). Algorithm combinations and respective switch points can be found in Appendix 4. All algorithms were run according to the settings outlined in Section 5.1.

8 Conclusion and future work

In this thesis, we have shown that implementing a single-switch dynAS routine for a small portfolio of continuous black-box solvers leads to performance improvements over the portfolio's virtual static best solver on various function-dimension pairs. In that regard, this work serves as a proof of concept for the approach to switch between algorithms after a certain function value has been reached, as previously suggested by Vermetten, Wang, et al. (2020). On approximately half of the identified use cases, we were able to realize at least the theoretical performance improvement derived from analyzing the corresponding empirical performance data. For these use cases, the resulting ERT curve against target values often resembles a continuation of the static ERT curve of the first algorithm with that of the second one, matching our initial expectations. However, if the second algorithm is not properly warmstarted, the ERT curve of an algorithm combination leaps after the switch point. Therefore, our experimental findings confirm the importance of warmstarting to realize performance improvements with dynAS.

We have demonstrated that performance improvements result from combining the advantages of different algorithmic search behaviours. For example, on unimodal, ill-conditioned functions, BFGS requires less evaluations to approach the optimum, whereas CMA-ES is faster to converge within the basin of attraction. We expect that further research on different algorithm combinations and the associated search behaviours will reveal similar mechanics. In addition, our experiments illustrated that the observed performance improvements are not only related to a more efficient search behaviour, but also to an increased success probability to hit target precision.

Following a data-driven approach, we were able to identify algorithm combinations and related switch points that lead to performance improvements over $\text{VBS}_{\text{static}}$. However, the predictions were not always accurate, so that we observed large discrepancies, both positive and negative, between actual and predicted improvements. Consequently, it is possible that other algorithm combinations within our portfolio perform even better than the combinations identified in Section 5.2. Adding to the discussion of limitations of the approach (see Section 5.5), we experienced difficulties related to the performance metric ERT. Especially on difficult-to-solve, high-dimensional problems, the absolute ERT values are very large, which can be explained by the low number of successful runs. As a result, only one additional successful run has a large impact on ERT. Given the stochastic nature of IOHs, we conjecture that repeating the experiments may lead to different algorithm combinations and switch points for the aforementioned problems. In the future, this problem can be mitigated by investigating several algorithm combinations and switch points per function-dimension pair.

To process and evaluate experimental data, we made use of the IOHprofiler tool. Especially the feature to inspect ERT curves for different algorithms and their individual runs proved to be useful. Moreover, visualizing the search history of individual algorithms and algorithm combinations with contour plots helped us to understand the course of optimization in detail. Thus, utilizing the IOHprofiler and extending it with features for search history visualization, especially for high-dimensional problems, is recommended for further research on dynAS.

In the near term, this work can be extended by including additional algorithms in the portfolio. In particular, it would be worth investigating if the findings presented here can be transferred to other algorithms from the same class, such as different versions of CMA-ES or other Quasi-Newton methods. Likewise, the portfolio could be extended by algorithms from different algorithm families, such as Nelder-Mead or Variable Neighbourhood Search. Moreover, as more warmstarting procedures become available, an advanced switch procedure would allow switching from any to any other algorithm on multiple different switch points. Thereby, future research could investigate the effect of varying values for τ on performance, as well as examining the performance of all algorithm combinations within the portfolio.

As previously discussed, we expect that tuning algorithmic hyperparameters as well as parameters for warmstarting, such as the scaling factor β for warmstarting the covariance matrix, will have a large impact on the performance of dynAS. Initially, tuning will lead to different ERT values for the individual algorithm in the portfolio, changing the choice of A_1 and A_2 algorithms and the respective switch point per function-dimension pair. Improved warmstarting parameters will probably lead to even better performances of algorithm combinations. Another interesting research direction for tuning would be to adapt algorithmic parameters that influence search behaviour based on the algorithm's assignment as first or second algorithm in a dynAS process. For example, in the modular CMA-ES framework, the parameter *ps_factor* determines if the algorithm favours exploitation or exploration behaviour. If CMA-ES is set as A_2, it may be beneficial to change this parameter in favour of exploitative behaviour.

In this work, we applied our approach to the problems available within the BBOB test suite. Even though it contains various continuous functions with different properties, our findings need to be further validated. For instance, the approach presented here could be tested on the *Nevergrad* platform, an optimization and benchmarking library provided by Facebook AI (Rapin and Teytaud 2018). Another option would be to test dynamic algorithm selection on real-world problems.

By investigating selected algorithm combinations in more detail, we have developed warmstarting routines that in some cases rely on information that is only available from running specific A_1 algorithms. Thus, it is questionable if our warmstarting routines can be applied to other algorithm combinations as well. We expect that some routines can be generalized, such as the neighbourhood method to initialize the population in DE and PSO (see Section 6.3.1). However, other routines need to be refined to make them less dependent on the parameters that are maintained by the A_1 algorithm. For instance, initializing the covariance matrix in CMA-ES as outlined in Section 6.1.3 depends on the availability of the local inverse Hessian matrix at the switch point. If BFGS is set as first algorithm, we can directly obtain this parameter, since the algorithm maintains an approximate inverse Hessian matrix over the course of optimization. To warmstart the covariance matrix in algorithm combinations with different A_1 algorithms, the local inverse Hessian needs to be supplied in a different way. The approach outlined by Mohammadi, Riche, and Touboul (2015), where the Hessian matrix and step size are calculated based on a Gaussian process model of the already sampled points, seems to be a promising direction for further research in that regard.

Another focus area for future research on dynAS is the selection of the switch point. In this work, we have only investigated single-switch algorithm combinations. However, reviewing the ERT charts showed that switching multiple times between algorithms may improve performance even more (e.g. Figure 17). Moreover, our experiments revealed that the right switch point may differ depending on individual runs. Therefore, initiating the switch automatically, e.g. based on the current search progress of A_1, could prove to be an interesting research direction.

Finally, this work is based on the assumption that the algorithms involved in a dynAS process are determined prior to the optimization. In the long term, a true dynamic algorithm selector may not only initiate the switch automatically, but also select which algorithm to switch to. For example, this could be based on knowledge about the solution landscape, obtained from computing landscape features from the points already sampled by A_1.

References

[Bau14] Petr Baudis. COCOpf: An Algorithm Portfolio Framework. In: *CoRR* abs/1405.3487 (2014). arXiv: 1405.3487.

[Bie+19] André Biedenkapp et al. Towards White-box Benchmarks for Algorithm Control. 2019. arXiv: 1906.07644 [cs.LG].

[BP11] Thomas Bartz-Beielstein and Mike Preuss. Experimental Analysis of Optimization Algorithms: Tuning and Beyond. In: Jan. 2011. DOI: 10.1007/978-3-642-33206-7_10.

[Bro70] C. G. Broyden. The Convergence of a Class of Double-rank Minimization Algorithms: 2. The New Algorithm. In: *IMA Journal of Applied Mathematics* 6.3 (Sept. 1970), pp. 222–231. ISSN: 0272-4960. DOI: 10.1093/imamat/6.3.222. eprint: https://academic.oup.com/imamat/article-pdf/6/3/222/1848059/6-3-222.pdf.

[BWB20] Rick Boks, Hao Wang, and Thomas Bäck. A Modular Hybridization of Particle Swarm Optimization and Differential Evolution. 2020. arXiv: 2006.11886 [cs.NE].

[Doe+18] Carola Doerr et al. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. In: *CoRR* abs/1810.05281 (2018). arXiv: 1810.05281.

[EK09] Mohammed El-Abd and Mohamed S. Kamel. Black-box optimization benchmarking for noiseless function testbed using particle swarm optimization. In: *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, Companion Material*. Ed. by Franz Rothlauf. ACM, 2009, pp. 2269–2274. DOI: 10.1145/1570256.1570316.

[Fle70] R. Fletcher. A new approach to variable metric algorithms. In: *The Computer Journal* 13.3 (Jan. 1970), pp. 317–322. ISSN: 0010-4620. DOI: 10.1093/comjnl/13.3.317. eprint: https://academic.oup.com/comjnl/article-pdf/13/3/317/988678/130317.pdf.

[GK20] Tobias Glasmachers and Oswin Krause. The Hessian Estimation Evolution Strategy. In: *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*. Ed. by Thomas Bäck et al. Vol. 12269. Lecture Notes in Computer Science. Springer, 2020, pp. 597–609. DOI: 10.1007/978-3-030-58112-1_41.

[Gol70] Donald Goldfarb. A Family of Variable-Metric Methods Derived by Variational Means. In: *Mathematics of Computation* 24.109 (1970), pp. 23–26. ISSN: 00255718, 10886842.

[Han+09] Nikolaus Hansen, Steffen Finck, et al. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829. INRIA, 2009.

[Han+10] Nikolaus Hansen, Anne Auger, Raymond Ros, et al. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. In: *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 1689–1696. ISBN: 9781450300735. DOI: 10.1145/1830761.1830790.

[Han+16] Nikolaus Hansen, Anne Auger, Olaf Mersmann, et al. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. In: *CoRR* abs/1603.08785 (2016). arXiv: 1603.08785.

[Han16] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. In: *CoRR* abs/1604.00772 (2016). arXiv: 1604.00772.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. In: *Evol. Comput.* 9.2 (2001), pp. 159–195. DOI: 10.1162/106365601750190398.

[HO96] Nikolaus Hansen and Andreas Ostermeier. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In: *Proceedings of 1996 IEEE International Conference on Evolutionary Computation, Nayoya University, Japan, May 20-22, 1996*. Ed. by Toshio Fukuda and Takeshi Furuhashi. IEEE, 1996, pp. 312–317. DOI: 10.1109/ICEC.1996.542381.

[KE95] James Kennedy and Russell Eberhart. Particle swarm optimization. In: *Proceedings of International Conference on Neural Networks (ICNN'95), Perth, WA, Australia, November 27 - December 1, 1995*. IEEE, 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[KT19] Pascal Kerschke and Heike Trautmann. Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning. In: *Evolutionary Computation* 27.1 (Mar. 2019), pp. 99–127. ISSN: 1530-9304. DOI: 10.1162/evco_a_00236.

[KT87] Alexander H. G. Rinnooy Kan and G. T. Timmer. Stochastic global optimization methods part II: Multi level methods. In: *Math. Program.* 39.1 (1987), pp. 57–78. DOI: 10.1007/BF02592071.

[MRT15] Hossein Mohammadi, Rodolphe Le Riche, and Eric Touboul. Making EGO and CMA-ES Complementary for Global Optimization. In: *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*. Ed. by Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion. Vol. 8994. Lecture Notes in Computer Science. Springer, 2015, pp. 287–292. DOI: 10.1007/978-3-319-19084-6_29.

[Nob+21] Jacob de Nobel et al. Tuning as a Means of Assessing the Benefits of New Ideas in Interplay with Existing Algorithmic Modules. 2021. arXiv: 2102.12905 [cs.NE].

[Pál13] László Pál. Benchmarking a hybrid multi level single linkage algorithm on the bbob noiseless testbed. In: *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material Proceedings*. Ed. by Christian Blum and Enrique Alba. ACM, 2013, pp. 1145–1152. DOI: 10.1145/2464576.2482692.

[PK12] Petr Posik and Vaclav Klema. Benchmarking the differential evolution with adaptive encoding on noiseless functions. In: *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*. Ed. by Terence Soule and Jason H. Moore. ACM, 2012, pp. 189–196. DOI: 10.1145/2330784.2330813.

[Ric76] John R Rice. The Algorithm Selection Problem. Vol. 15. Advances in Computers Volume 15. Elsevier, 1976, pp. 65–118. ISBN: 9780120121151.

[Rij+16] Sander van Rijn et al. Evolving the structure of Evolution Strategies. In: *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*. IEEE, 2016, pp. 1–8. DOI: 10.1109/SSCI.2016.7850138.

[RT18] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad. 2018.

[Sch21] Dominik Schröder. Dynamic Algorithm Selection - Experimental Data. Zenodo, Apr. 2021. DOI: 10.5281/zenodo.4721299.

[SEB18] Jörg Stork, A. E. Eiben, and Thomas Bartz-Beielstein. A new Taxonomy of Continuous Global Optimization Algorithms. In: *CoRR* abs/1808.08818 (2018). arXiv: 1808.08818.

[Sha70] D. F. Shanno. Conditioning of Quasi-Newton Methods for Function Minimization. In: *Mathematics of Computation* 24.111 (1970), pp. 647–656. ISSN: 00255718, 10886842.

[SP97] Rainer Storn and Kenneth V. Price. Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. In: *J. Glob. Optim.* 11.4 (1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[Spe+20] David Speck et al. Learning Heuristic Selection with Dynamic Algorithm Configuration. 2020. arXiv: 2006.08246 [cs.AI].

[SY20] Ofer M. Shir and Amir Yehudayoff. On the covariance-Hessian relation in evolution strategies. In: *Theor. Comput. Sci.* 801 (2020), pp. 157–174. DOI: 10.1016/j.tcs.2019.09.002.

[Ver+19] Diederick Vermetten, Sander van Rijn, et al. Online selection of CMA-ES variants. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Anne Auger and Thomas Stützle. ACM, 2019, pp. 951–959. DOI: 10.1145/3321707.3321803.

[Ver+20] Diederick Vermetten, Hao Wang, et al. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In: *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*. Ed. by Carlos Artemio Coello Coello. ACM, 2020, pp. 654–662. DOI: 10.1145/3377930.3390189.

[Vir+20] Pauli Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[Vog+12] Costas Voglis et al. MEMPSODE: comparing particle swarm optimization and differential evolution within a hybrid memetic global optimization framework. In: *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*. Ed. by Terence Soule and Jason H. Moore. ACM, 2012, pp. 253–260. DOI: 10.1145/2330784.2330821.

[Wan+20] Hao Wang et al. IOHanalyzer: Performance Analysis for Iterative Optimization Heuristic. 2020. arXiv: 2007.03953 [cs.NE].

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893.

Appendices

Appendix 1 - ERT comparison charts

Figure 1: ERT charts for our BFGS implementation (red) compared to Baudis 2014 (pink) on all 24 BBOB functions in dimension 2.

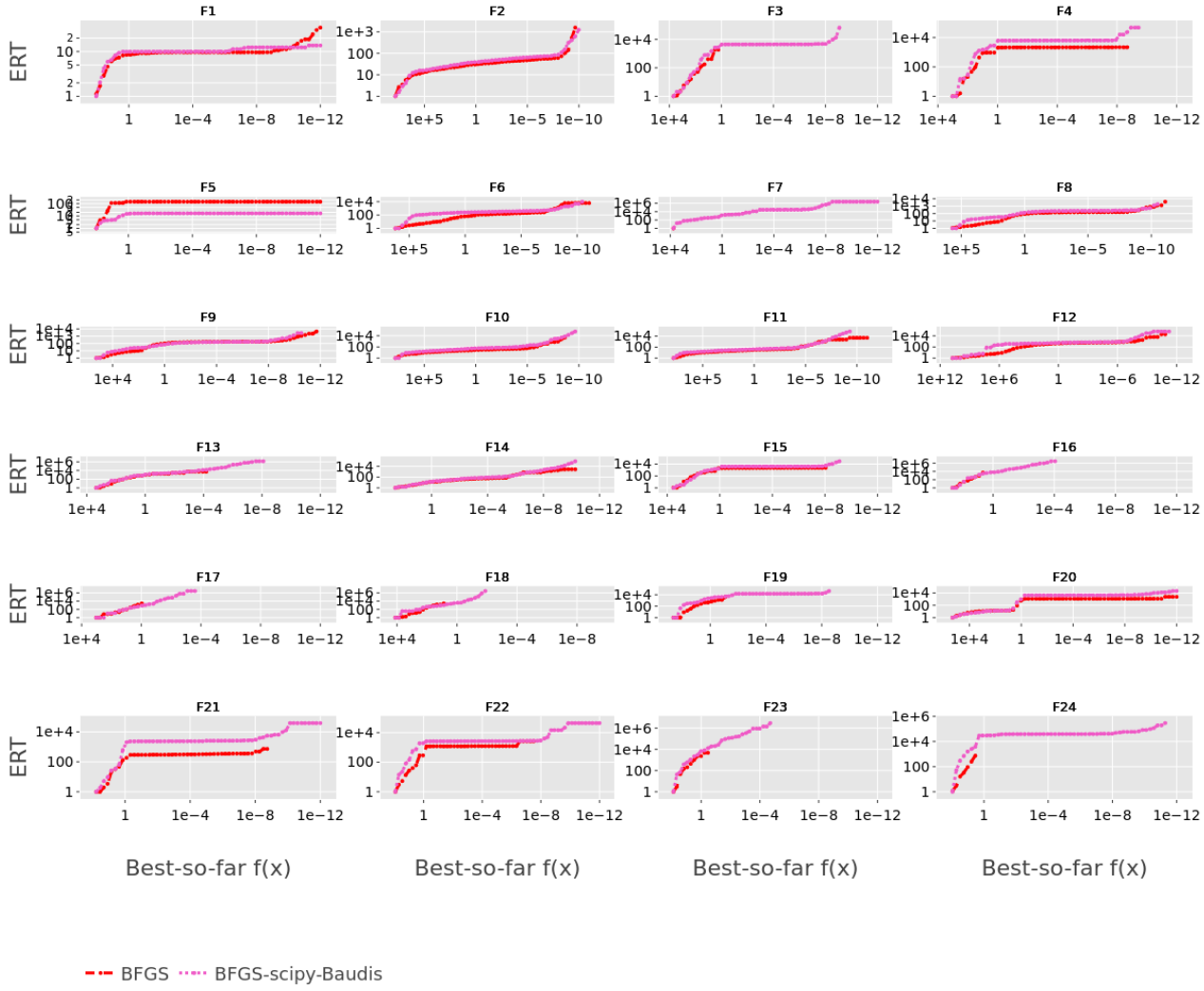


Figure 2: ERT charts for our BFGS implementation (red) compared to Baudis 2014 (pink) on all 24 BBOB functions in dimension 5.

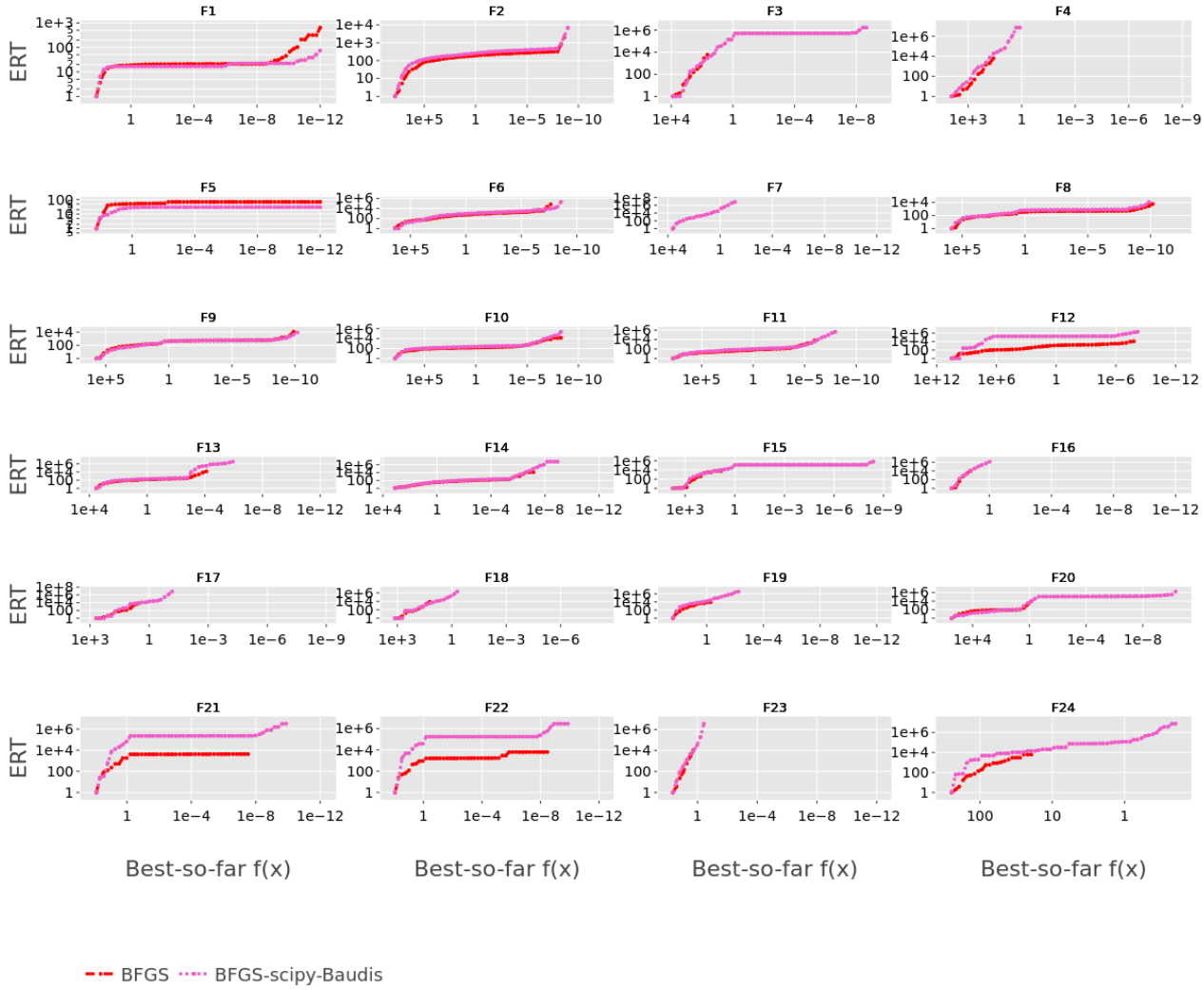


Figure 3: ERT charts for our MLSL implementation (brown) compared to Pál 2013 (green) on all 24 BBOB functions in dimension 2.

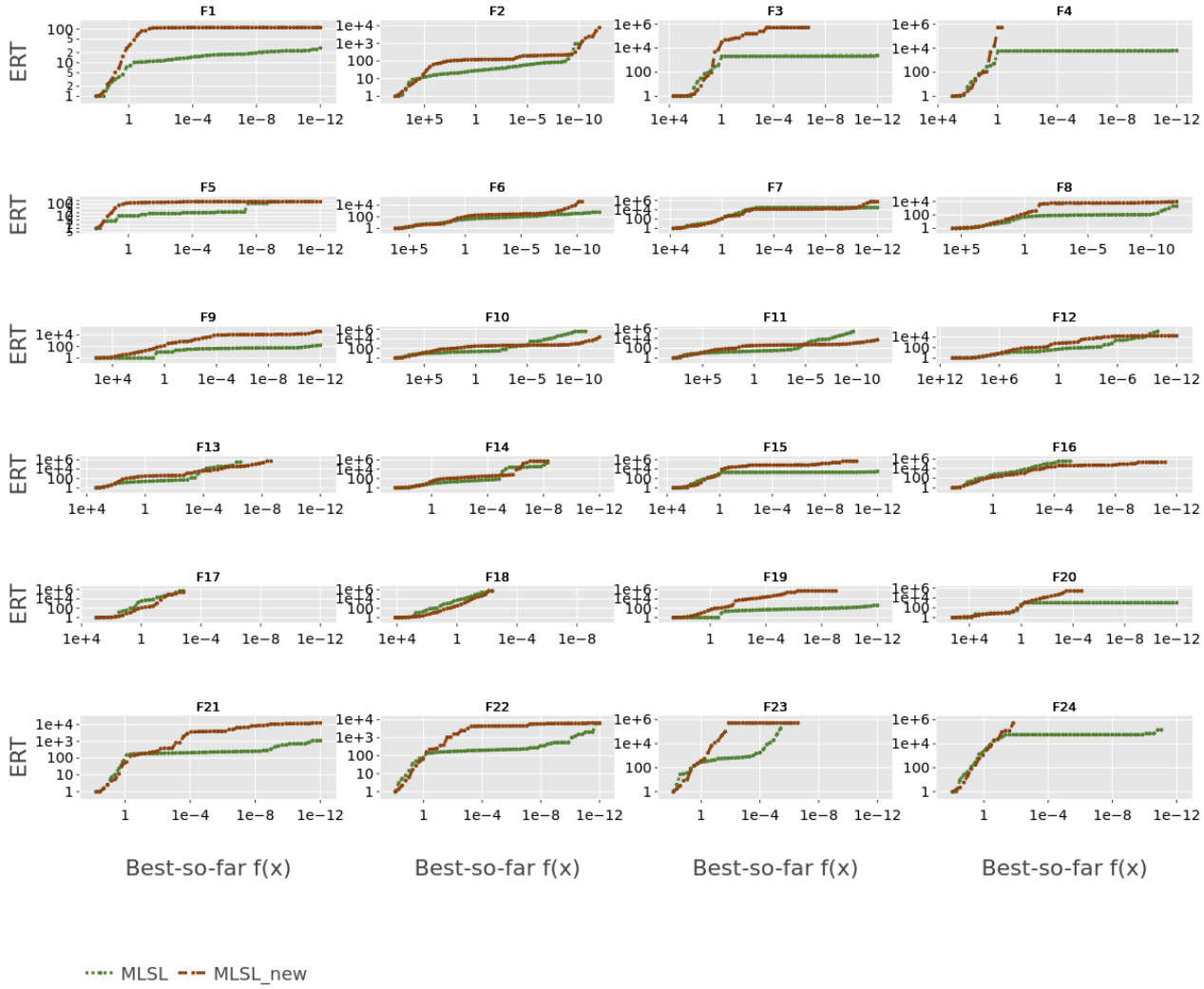


Figure 4: ERT charts for our MLSL implementation (brown) compared to Pál 2013 (green) on all 24 BBOB functions in dimension 10.

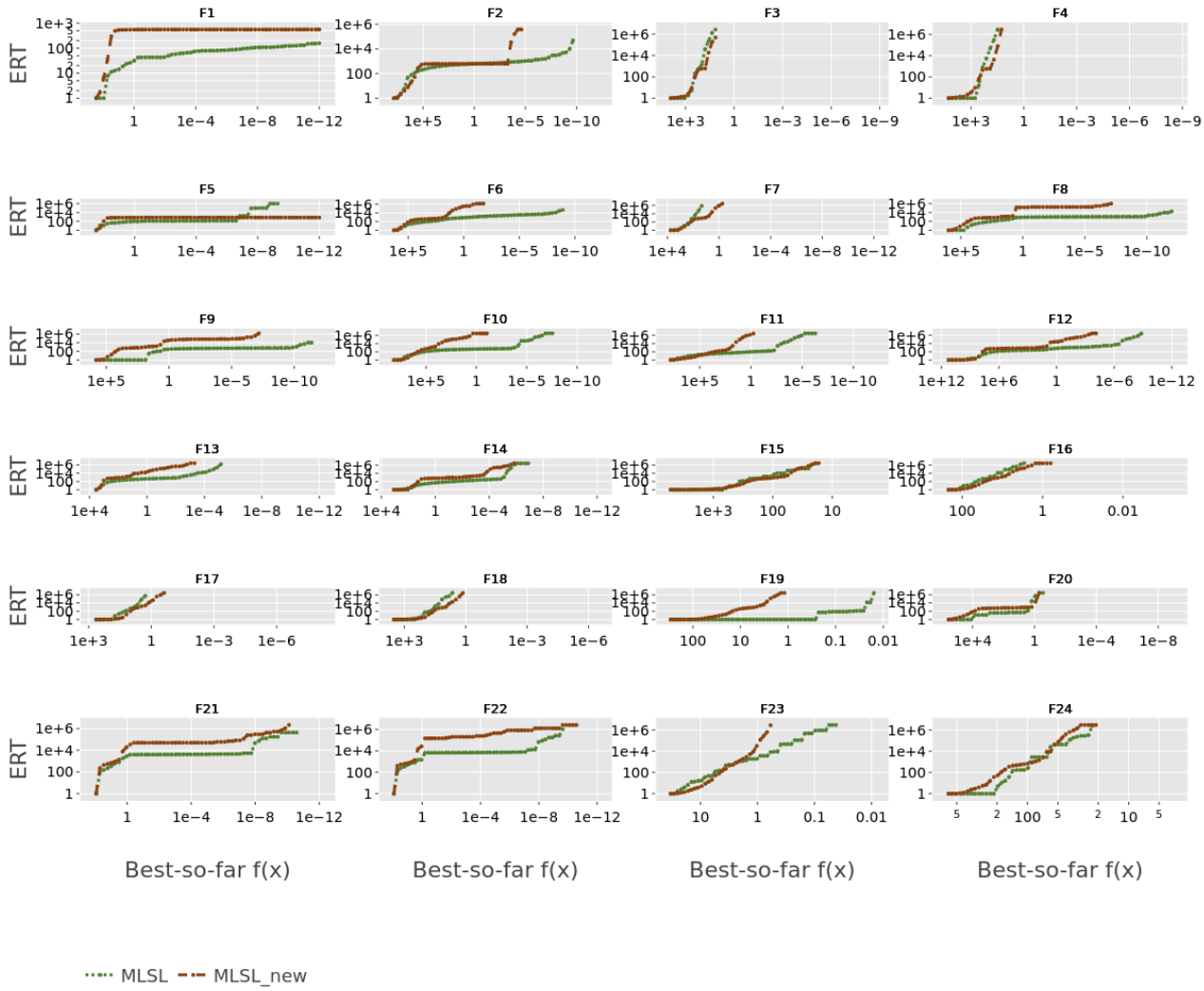


Figure 5: ERT charts for our PSO implementation (dark blue) compared to El-Abd and Kamel 2009 (light blue) on all 24 BBOB functions in dimension 2.

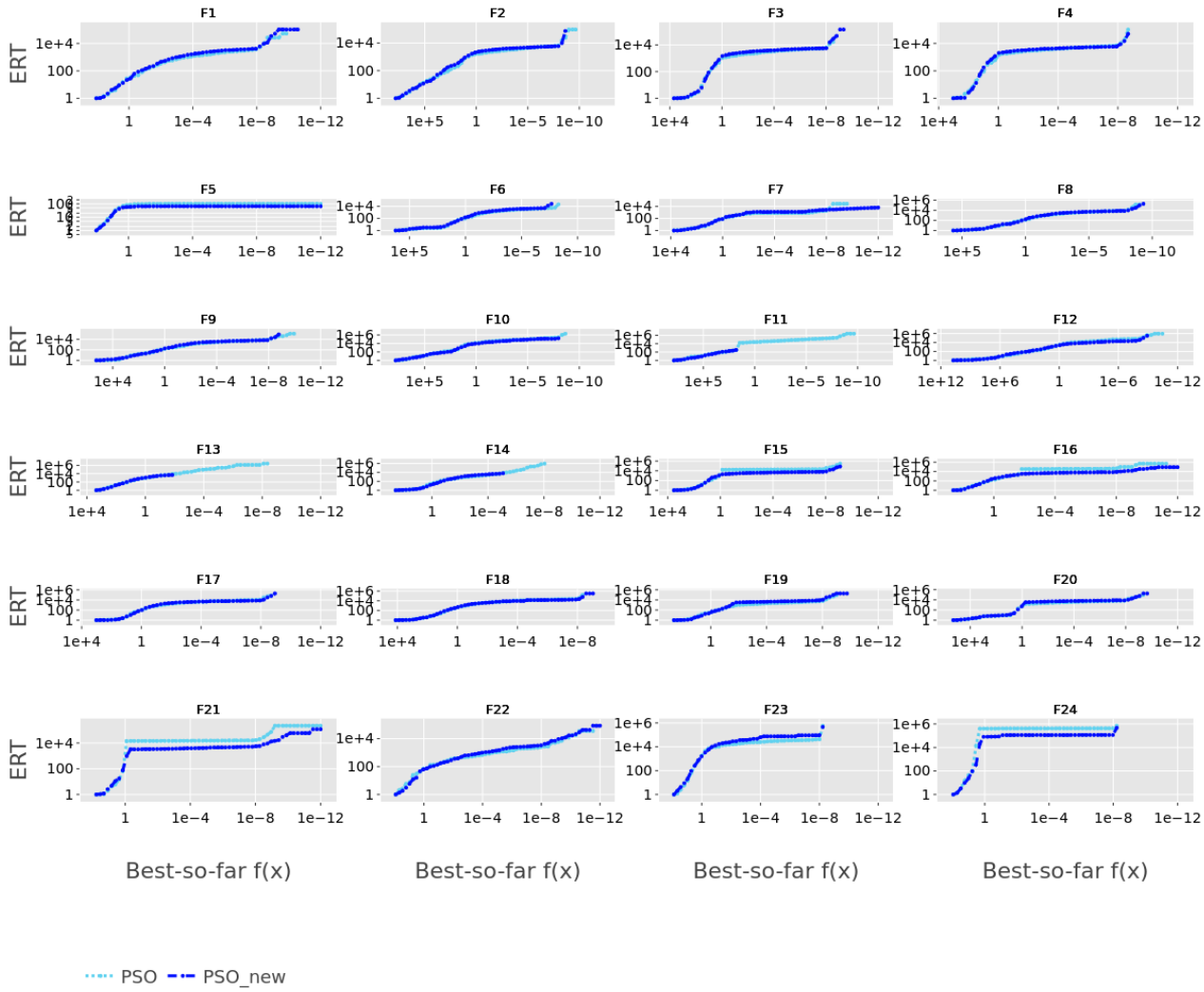


Figure 6: ERT charts for our PSO implementation (dark blue) compared to El-Abd and Kamel 2009 (light blue) on all 24 BBOB functions in dimension 10.

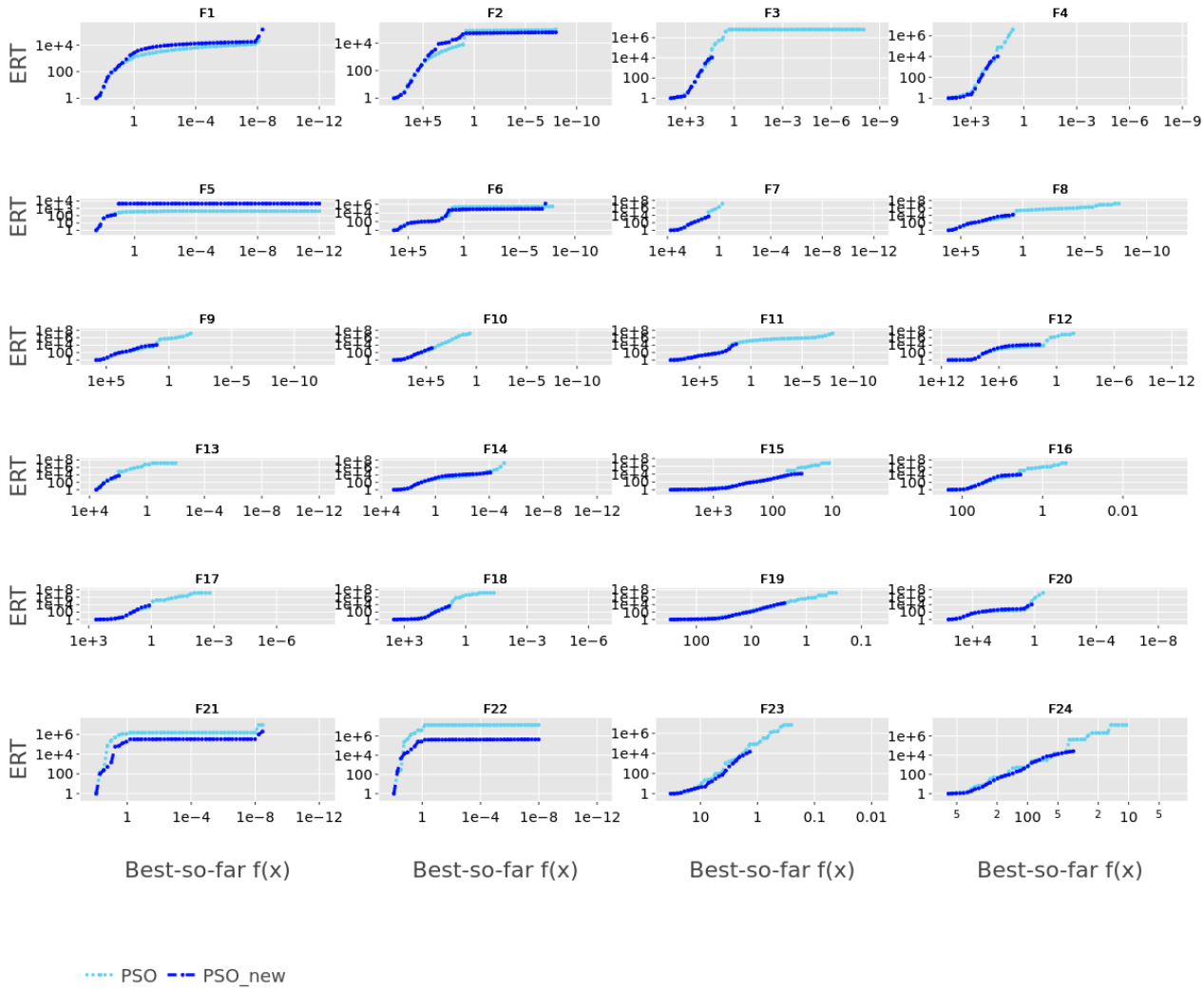


Figure 7: ERT charts for our DE implementation (orange) compared to Posik and Klema 2012 (brown) on all 24 BBOB functions in dimension 2.

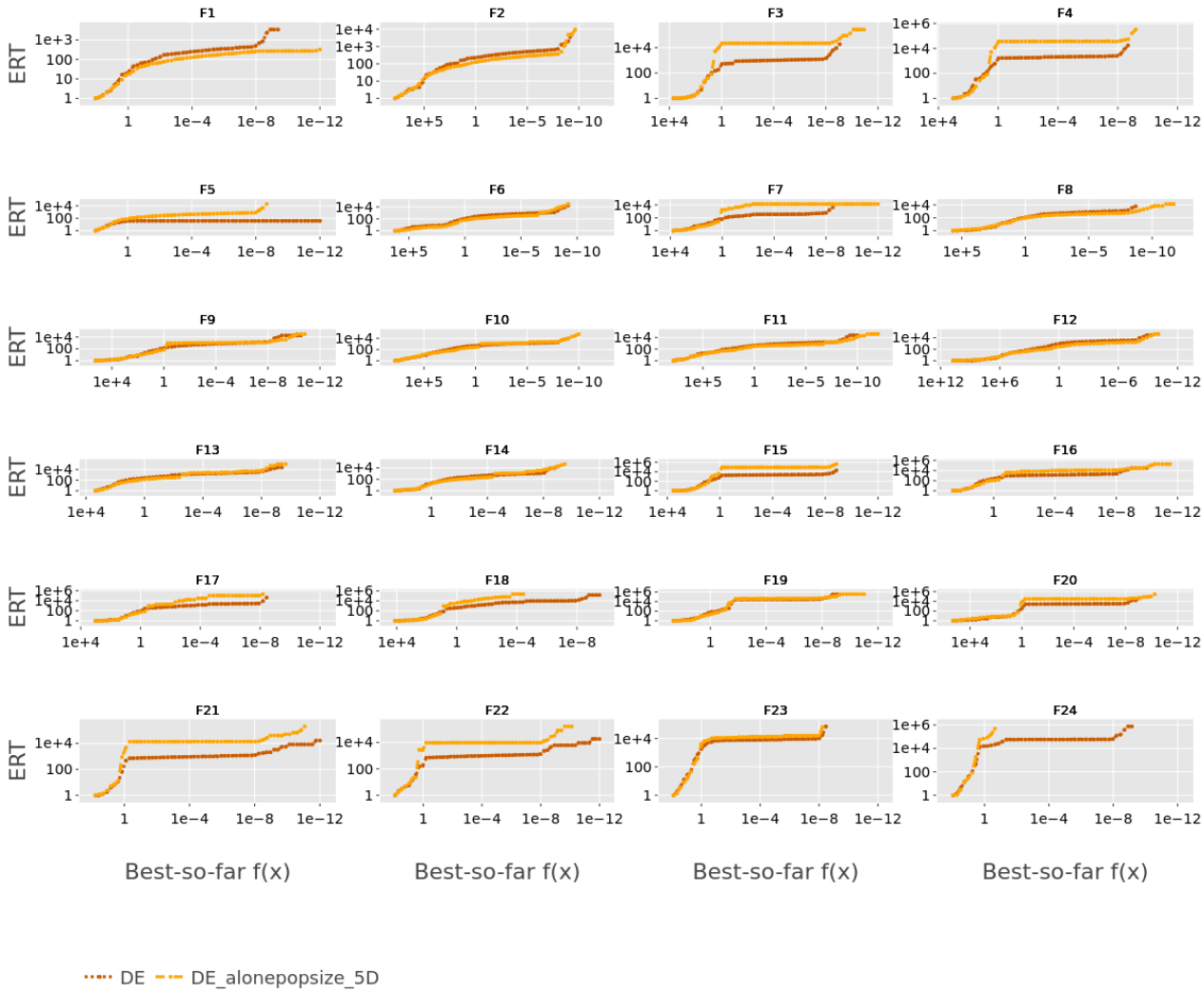
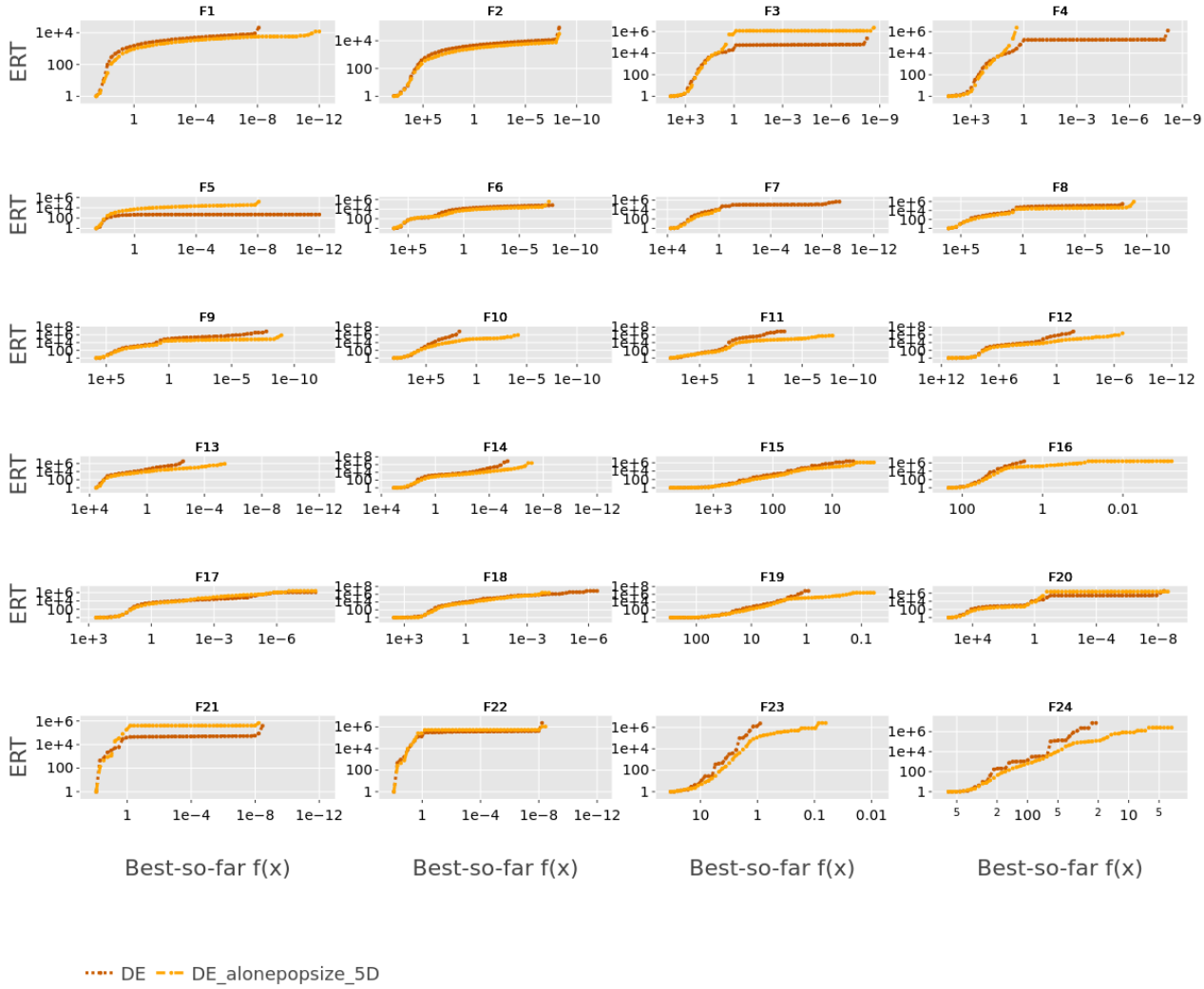


Figure 8: ERT charts for our DE implementation (orange) compared to Posik and Klema 2012 (brown) on all 24 BBOB functions in dimension 10.

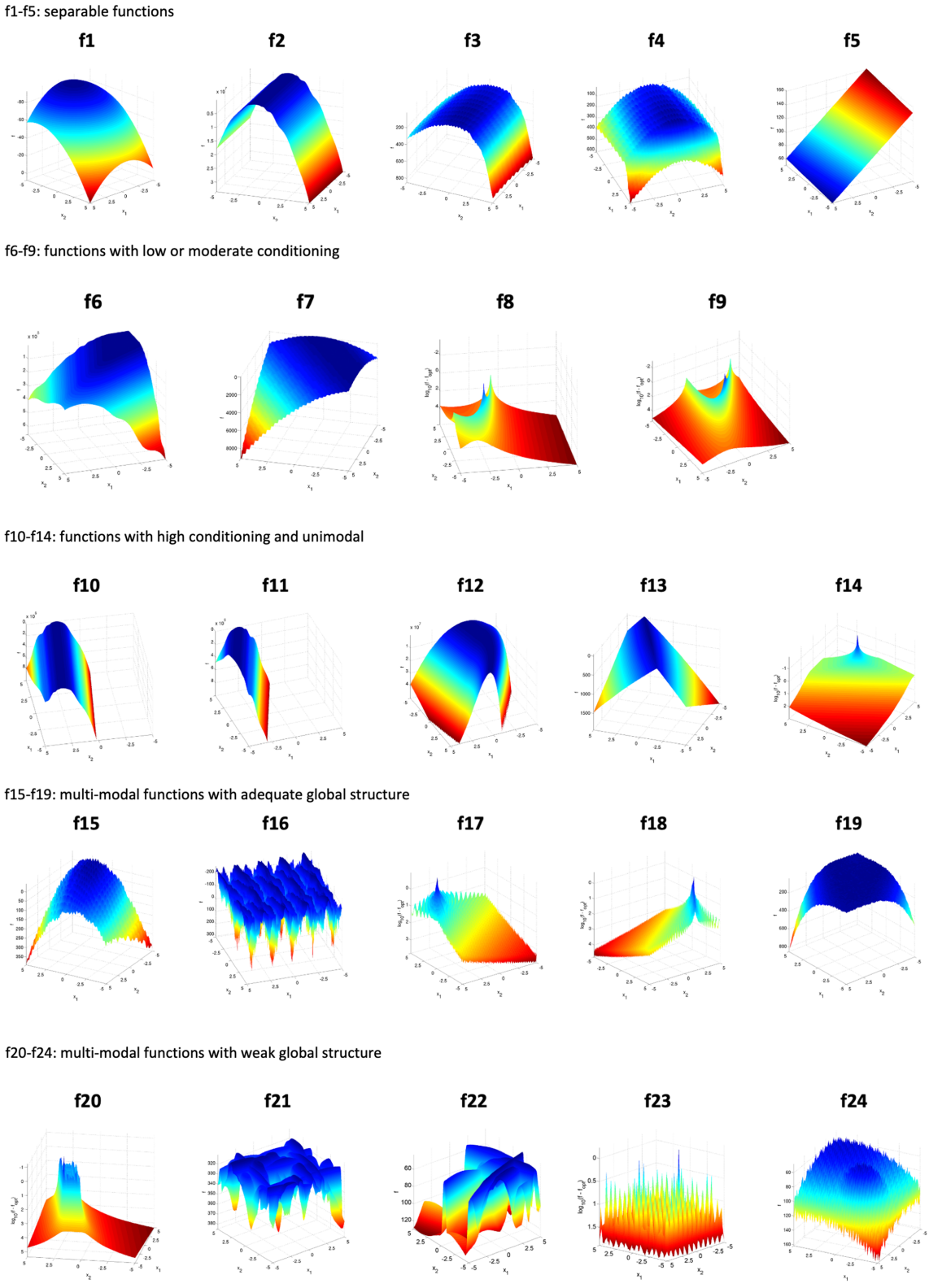


Appendix 2 - Parameters for warmstarting

Algorithm	**Available information**	**Parameters available for warmstarting**	**Parameters *not* relevant for warmstarting**
All algorithms	- best point found so far $\mathbf{x_{opt}}$ - fitness value for best point $\mathbf{f_{opt}}$ - trajectory of sampled points $\mathbf{X_{hist}}$ - trajectory of fitness values $\mathbf{f_{hist}}$ - iteration counter $\mathbf{k}$		
BFGS	- step size $\alpha_\mathbf{k}$ - inverse Hessian approximation matrix $\mathbf{H_k}$ - gradient vector at best point so far $\nabla\mathbf{f}(\mathbf{x_{opt}})$	- inverse Hessian approximation matrix $\mathbf{B_k}$ - initial point $\mathbf{x_0}$	- step size $\alpha_\mathbf{k}$
MLSL	- critical distance measure $\mathbf{r_k}$ - reduced sample $\mathbf{X_r}$ and corresponding fitness values $\mathbf{f_r}$ - best points found by local search, $\mathbf{X}^*$ and corresponding fitness values $\mathbf{f}^*$	- initial population $\mathbf{X}$ - iteration counter $\mathbf{k}$	- critical distance measure $\mathbf{r_k}$
CMA-ES	- current distribution mean $\mathbf{m}$ - step size σ - covariance matrix $\mathbf{C}$ - recent evolution paths $\mathbf{p_c}$ and $\mathbf{p_s}$	- distribution mean $\mathbf{m}$ - step size σ - covariance matrix $\mathbf{C}$ - recent evolution paths $\mathbf{p_c}$ and $\mathbf{p_s}$	
PSO	- particle velocities $\mathbf{v_i}$ - inertia weight ω	- initial swarm positions $\mathbf{x_i}$ - particle velocities $\mathbf{v_i}$	- inertia weight ω
DE		- initial population of points $\mathbf{x_i}$	

Appendix 3 - BBOB functions solution landscapes

Figure 9: Solution landscapes for all 24 BBOB functions depicted by 3D surface plots in dimension 2, taken from Hansen et al. 2009.



Appendix 4 - Experimental results

function	dimension	tau	VBS_static	ERT (VBS_static)	A1	A2	ERT(VBS_dyn) expected	ERT(VBS_dyn) actual	Expected improvement	Improvement vs. VBS_static
1	2	1.00E+01	BFGS	9.64	CMA-ES	BFGS	5.2	22	46%	-128%
2	2	3.98E-08	BFGS	60.96	BFGS	MLSL	59.76	n.a. (MLSL as A2)	2%	n.a.
3	2	6.31E-01	PSO	5668.56	PSO	DE	1894.23	1735.88	67%	**69%**
4	2	6.31E+00	BFGS	2172	DE	BFGS	1308.6	146864.33	40%	-6662%
5	2	1.58E+01	CMA-ES	35.6	DE	CMA-ES	12.6	50.6	65%	-42%
6	2	6.31E-08	CMA-ES	589.56	BFGS	CMA-ES	324.33	327.2	45%	**45%**
7	2	3.98E-03	PSO	2752.12	PSO	DE	1254.65	1208.08	54%	**56%**
8	2	1.00E+01	BFGS	155.08	CMA-ES	BFGS	131.68	87.52	15%	**44%**
9	2	1.58E+00	BFGS	175.24	CMA-ES	BFGS	93.92	84.64	46%	**52%**
10	2	1.00E-05	MLSL	512.52	BFGS	MLSL	77.74	n.a. (MLSL as A2)	85%	n.a.
11	2	1.00E-04	MLSL	603.84	BFGS	MLSL	167.88	n.a. (MLSL as A2)	72%	n.a.
12	2	1.58E-06	BFGS	1237.07	BFGS	CMA-ES	791.6	1032.04	36%	**17%**
13	2	1.00E-02	CMA-ES	986.84	DE	CMA-ES	850.84	1238.2	14%	-25%
14	2	3.98E-06	CMA-ES	705	BFGS	CMA-ES	364.72	261.64	48%	**63%**
15	2	1.58E+00	BFGS	2295	MLSL	BFGS	624.08	18759.15	73%	-717%
16	2	1.00E-02	PSO	7030.21	MLSL	CMA-ES	2923.94	6073.77	58%	**14%**
17	2	2.51E-05	PSO	7639.88	PSO	DE	5930.37	5844.04	22%	**24%**
18	2	1.00E+00	PSO	17718.72	MLSL	PSO	17576.2	13147.18	1%	**26%**
19	2	1.00E-03	PSO	8142	PSO	CMA-ES	5400.98	4688.52	34%	**42%**
20	2	1.00E+00	BFGS	1171	MLSL	BFGS	842.38	5774.5	28%	-393%
21	2	6.31E-01	BFGS	486.67	MLSL	DE	229.61	565.52	53%	-16%
22	2	6.31E-01	PSO	3343.28	PSO	DE	256.03	270.84	92%	**92%**
23	2	6.31E-01	CMA-ES	3418.24	MLSL	CMA-ES	1504.24	4232.04	56%	-24%
24	2	1.58E+00	PSO	116140.75	MLSL	PSO	36253.11	16435.78	69%	**86%**
1	3	1.58E+01	BFGS	13.16	DE	BFGS	8.08	39.48	39%	-200%
2	3	6.31E+00	BFGS	124.08	BFGS	BFGS	124.08	n.a. (self-switch)	0%	n.a.
3	3	6.31E-01	PSO	9608.13	PSO	CMA-ES	5728.21	6051.29	40%	**37%**
4	3	1.58E+00	PSO	17943.7	PSO	CMA-ES	4529.98	349526.5	75%	-1848%
5	3	1.00E+01	BFGS	27.33	BFGS	CMA-ES	19.92	Inf	27%	n.a.
6	3	1.58E-07	CMA-ES	942.56	BFGS	CMA-ES	672.32	1018.36	29%	-8%
7	3	1.00E-02	PSO	4518.76	PSO	DE	3488.29	8739.14	23%	-93%
8	3	3.98E+00	BFGS	277.32	CMA-ES	BFGS	242.16	204.92	13%	**26%**
9	3	2.51E+00	BFGS	259.24	CMA-ES	BFGS	184.8	163.36	29%	**37%**
10	3	1.58E-05	CMA-ES	1135.92	BFGS	CMA-ES	267.18	267.36	76%	**76%**

11	3	1.00E-04	CMA-ES	1257	BFGS	CMA-ES	278.52	260.76	78%	**79%**
12	3	2.51E-05	CMA-ES	3907.8	BFGS	CMA-ES	1728.4	1877.76	56%	**52%**
13	3	1.58E-03	CMA-ES	1722.44	BFGS	CMA-ES	994.16	1105.68	42%	**36%**
14	3	2.51E-06	CMA-ES	1225.12	BFGS	CMA-ES	539.95	434.64	56%	**65%**
15	3	6.31E-01	PSO	66221.78	PSO	CMA-ES	56086.07	69934.38	15%	-6%
16	3	2.51E-01	CMA-ES	21897.59	PSO	CMA-ES	11168.86	24126.38	49%	-10%
17	3	6.31E-06	PSO	29942.44	PSO	CMA-ES	20210.11	362341	33%	-1110%
18	3	1.00E-05	DE	160478	DE	CMA-ES	122482.8	722747	24%	-350%
19	3	1.58E-02	DE	79498.86	PSO	CMA-ES	39273.11	23854.88	51%	**70%**
20	3	6.31E-01	BFGS	4192	MLSL	BFGS	1156.92	53574.33	72%	-1178%
21	3	6.31E-01	BFGS	3028	MLSL	BFGS	687.12	1925.78	77%	**36%**
22	3	2.51E+00	BFGS	999.33	PSO	BFGS	683.37	345377	32%	-34461%
23	3	1.00E+00	CMA-ES	10859.12	MLSL	CMA-ES	7662.4	12969.83	29%	-19%
24	3	6.31E-02	PSO	749916	DE	PSO	358100	Inf	52%	n.a.
1	5	3.98E+01	BFGS	20.92	MLSL	BFGS	15.52	279	26%	-1234%
2	5	1.00E+02	BFGS	319.96	BFGS	BFGS	319.96	n.a. (self-switch)	0%	n.a.
3	5	3.98E+00	PSO	54564.14	DE	PSO	47473.12	52535.43	13%	**4%**
4	5	1.00E+01	PSO	122262.25	DE	PSO	118712.69	82144.18	3%	**33%**
5	5	6.31E+01	CMA-ES	65.56	PSO	CMA-ES	32.52	137.96	50%	-110%
6	5	1.00E-01	CMA-ES	1893.64	CMA-ES	CMA-ES	1893.64	n.a. (self-switch)	0%	n.a.
7	5	1.58E+00	CMA-ES	180029.33	MLSL	CMA-ES	175356.12	Inf	3%	n.a.
8	5	1.00E+02	BFGS	500.83	CMA-ES	BFGS	472.11	580.26	6%	-16%
9	5	1.00E+02	BFGS	551.8	CMA-ES	BFGS	428.56	301.42	22%	**45%**
10	5	2.51E-04	CMA-ES	2263.28	BFGS	CMA-ES	574.16	533.68	75%	**76%**
11	5	2.51E-04	CMA-ES	2259.44	BFGS	CMA-ES	526.35	506.08	77%	**78%**
12	5	3.98E-05	CMA-ES	5345.48	BFGS	CMA-ES	2966.4	3427.64	45%	**36%**
13	5	2.51E-03	CMA-ES	3938.08	BFGS	CMA-ES	2294.00	2111.24	42%	**46%**
14	5	3.98E-06	CMA-ES	2434.08	BFGS	CMA-ES	1158.56	825.44	52%	**66%**
15	5	1.00E+02	BFGS	Inf	DE	BFGS	Inf	Inf	n.a.	n.a.
16	5	2.51E-04	DE	268768.5	CMA-ES	DE	188280.15	622592.5	30%	-132%
17	5	2.51E-04	DE	268535.5	DE	PSO	21092.5	1218152	92%	-354%
18	5	1.00E+02	BFGS	Inf	MLSL	BFGS	Inf	Inf	n.a.	n.a.
19	5	6.31E-03	CMA-ES	1222586	CMA-ES	DE	589411.00	Inf	52%	n.a.
20	5	1.00E+00	PSO	172439.33	MLSL	PSO	168235.45	405181	2%	-135%
21	5	1.00E+00	MLSL	49843.62	BFGS	CMA-ES	2258.5	1184041.5	95%	-2276%
22	5	2.51E-05	BFGS	6348	BFGS	CMA-ES	1876.54	110732.79	70%	-1644%
23	5	1.58E-01	DE	127277.25	CMA-ES	DE	93415.64	144316.43	27%	-13%

24	5	1.00E+02	BFGS	Inf	DE	BFGS	Inf	Inf	n.a.	n.a.
1	10	1.00E+02	BFGS	35.76	CMA-ES	BFGS	22	47.52	38%	-33%
2	10	1.00E+02	BFGS	771.6	BFGS	BFGS	771.6	n.a. (self-switch)	0%	n.a.
3	10	6.31E+01	DE	1175326.5	MLSL	DE	1174760.9	n.a. (no improv.)	0%	n.a.
4	10	1.00E+02	BFGS	Inf	MLSL	DE	Inf	Inf	n.a.	n.a.
5	10	3.98E+01	CMA-ES	123.64	BFGS	CMA-ES	57.32	175.4	54%	-42%
6	10	1.00E+02	CMA-ES	4254.52	CMA-ES	CMA-ES	4254.52	n.a. (self-switch)	0%	n.a.
7	10	6.31E+00	CMA-ES	2461972	MLSL	CMA-ES	2461920.76	n.a. (no improv.)	0%	n.a.
8	10	1.00E+02	BFGS	1963.2	CMA-ES	BFGS	1771.28	1234.21	10%	**37%**
9	10	1.00E+02	BFGS	1645.09	CMA-ES	BFGS	1270.73	1080.77	23%	**34%**
10	10	2.51E-05	CMA-ES	6151.32	BFGS	CMA-ES	1267.4	1310	79%	**79%**
11	10	2.51E-04	CMA-ES	5797.32	BFGS	CMA-ES	1072.12	1226.2	82%	**79%**
12	10	2.51E-05	CMA-ES	11082.4	BFGS	CMA-ES	6270.12	6956.28	43%	**37%**
13	10	2.51E-03	CMA-ES	15987.36	BFGS	CMA-ES	11193.2	9209.24	30%	**42%**
14	10	3.98E-06	CMA-ES	6750.84	BFGS	CMA-ES	3414.2	2399.08	49%	**64%**
15	10	1.00E+02	BFGS	Inf	DE	DE	Inf	Inf	n.a.	n.a.
16	10	1.00E+02	BFGS	Inf	DE	BFGS	Inf	Inf	n.a.	n.a.
17	10	2.51E+01	DE	2442482	MLSL	DE	2442479.68	n.a. (no improv.)	0%	n.a.
18	10	1.00E+02	BFGS	Inf	MLSL	BFGS	Inf	Inf	n.a.	n.a.
19	10	1.00E+02	BFGS	Inf	MLSL	BFGS	Inf	Inf	n.a.	n.a.
20	10	1.58E+00	DE	2414838	MLSL	DE	2410938.92	n.a. (no improv.)	0%	n.a.
21	10	1.00E+00	MLSL	294066.29	BFGS	CMA-ES	9924.00	732071	97%	-149%
22	10	1.58E+00	CMA-ES	318490.5	BFGS	CMA-ES	12056.5	563100.88	96%	-77%
23	10	1.00E+02	BFGS	Inf	BFGS	BFGS	Inf	Inf	n.a.	n.a.
24	10	1.00E+02	BFGS	Inf	DE	DE	Inf	Inf	n.a.	n.a.
1	20	1.00E+02	BFGS	74.92	CMA-ES	BFGS	30.84	117.4	59%	-57%
2	20	3.98E-04	BFGS	2280.28	MLSL	BFGS	1651.88	37063.44	28%	-1525%
3	20	1.00E+02	BFGS	Inf	MLSL	DE	Inf	Inf	n.a.	n.a.
4	20	1.00E+02	BFGS	Inf	MLSL	DE	Inf	Inf	n.a.	n.a.
5	20	1.00E+02	BFGS	76.6	BFGS	BFGS	76.6	n.a. (self-switch)	0%	n.a.
6	20	1.00E+02	CMA-ES	10800	CMA-ES	CMA-ES	10800.00	n.a. (self-switch)	0%	n.a.
7	20	1.00E+02	MLSL	Inf	MLSL	MLSL	Inf	Inf	n.a.	n.a.
8	20	1.00E+02	BFGS	4746.16	CMA-ES	BFGS	4145.44	4282.82	13%	**10%**
9	20	6.31E+01	BFGS	4994.96	CMA-ES	BFGS	3745.76	3502.26	25%	**30%**
10	20	2.51E-05	CMA-ES	19825.48	BFGS	CMA-ES	3096.4	3341	84%	**83%**
11	20	1.58E-04	CMA-ES	15457	BFGS	CMA-ES	2540.52	2553.4	84%	**83%**
12	20	6.31E-05	CMA-ES	26711.84	BFGS	CMA-ES	16854.47	15785.28	37%	**41%**

13	20	2.51E-03	CMA-ES	76107.2	BFGS	CMA-ES	43279.43	37254.64	43%	**51%**
14	20	3.98E-06	CMA-ES	22488.8	BFGS	CMA-ES	12252.64	7291.48	46%	**68%**
15	20	1.00E+02	BFGS	Inf	DE	DE	Inf	Inf	n.a.	n.a.
16	20	1.00E+02	BFGS	Inf	DE	BFGS	Inf	Inf	n.a.	n.a.
17	20	1.00E+02	BFGS	Inf	DE	BFGS	Inf	Inf	n.a.	n.a.
18	20	1.00E+02	BFGS	Inf	MLSL	BFGS	Inf	Inf	n.a.	n.a.
19	20	1.00E+02	BFGS	Inf	BFGS	BFGS	Inf	Inf	n.a.	n.a.
20	20	1.00E+02	BFGS	Inf	BFGS	BFGS	Inf	Inf	n.a.	n.a.
21	20	1.58E-08	DE	654998.67	BFGS	MLSL	23500.00	n.a. (MLSL as A2)	96%	n.a.
22	20	1.00E+02	BFGS	Inf	BFGS	BFGS	Inf	Inf	n.a.	n.a.
23	20	1.00E+02	BFGS	Inf	BFGS	BFGS	Inf	Inf	n.a.	n.a.
24	20	1.00E+02	BFGS	Inf	DE	DE	Inf	Inf	n.a.	n.a.