BENCHMARKING ALGORITHMS FROM MACHINE LEARNING FOR LOW-BUDGET BLACK-BOX OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning has invaded various domains of computer science, including black-box optimization. Recent research is particularly concerned with Bayesian Optimization and with Monte Carlo Tree Search. However, comparative experiments are usually performed on rather small benchmarks and there are visible issues in the experimental setup, such as poor initialization of baselines, overfitting by specifying hyperparameters specifically for each test function, and low statistical significance. In addition, the interface is sometimes very problem-specific and has more impact on the results than the algorithm itself.
We compare several black-box optimization tools from the machine learning world and benchmark them on the classical BBOB benchmark suite, well known in the black-box optimization field, and on Direct Policy Search for OpenAI Gym, a classical Machine Learning benchmark.
The benchmarks in this work include randomization of the optimum. For BBOB, we consider 15 random instances per test function and dimension, resulting in a total of 24 functions $\times$ 6 dimensionalities $\times$ 15 random instances $=$ 2160 instances. For OpenAI Gym, we consider tiny and larger neural networks, on a total number of 13 problems $\times$ 8 budgets $\times$ 10 repetitions $=$ 1040 and 18 problems $\times$ 8 budgets $\times$ 10 repetitions $=$ 1440 instances, respectively.

1 INTRODUCTION

Black-Box Optimization (BBO) is an affirmed and rapidly growing field of optimization and a topic of critical importance in many application areas including complex systems engineering, energy and the environment, materials design, drug discovery, chemical process synthesis, and computational biology (Bajaj et al., 2021). As in other classical optimization contexts, BBO assumes that we are facing an objective function f for which we aim to provide a solution x with $f(x)$ as good as possible with as few calls to f as possible. The key distinguishing property of BBO is that we assume that we no not have any information about f other than the quality of the solution candidates that have already been evaluated. In particular, we do not have direct access to gradients. A typical BBO scenario is the optimization of a problem that requires simulations or physical experiments to assess and to compare the quality of the solution candidates.

Because of its high practical relevance, people with many different backgrounds are drawn into BBO, leading to a multitude of approaches in the area, ranging from heuristics to local/global modeling approaches. To understand strengths and weaknesses of these different methods, fair performance comparisons are needed. Several platforms and benchmark suites (i.e., collections of benchmark problems) address this empirical comparison, among them the Black-Box Optimization Benchmarking (BBOB) collection (Hansen et al., 2009b; Varelas et al., 2020; Hansen et al., 2021), Large-Scale Global Optimization (LSGO) (Tang et al., 2010), Nevergrad (Rapin & Teytaud, 2018), Pseudo-Boolean Optimization (PBO) (Doerr et al., 2018), and Machine Learning and Data Analysis (MLDA) (Marcus Gallagher, 2018; Rapin et al., 2019).[1] The proposed benchmarks are reproducible and, depending on the platform, are rooted in the real-world, have randomized optima, and may involve large-scale problems.

[1]Non-surprisingly, many other benchmarking environments for evaluating and comparing performances for specific BBO tasks, such as hyperparameter optimization (Bischl et al., 2016; Hutter et al., 2014), neural

Without knowing the structure of the objective function, the optimization problem can be addressed through learning (Wang et al., 2020b). Based on a training set, i.e., a set of samples whose objective function value is known, a surrogate regressor or model $\hat{f}$ is learned and optimized through an iterative procedure. In fact, surrogate models allow for the construction of a computationally cheap-to-evaluate approximation of the considered expensive objective function and replace the direct optimization of the real objective with the model. Specifically, if the true nature of the high-fidelity model is represented by a vector of variables $\boldsymbol{x}$ and an output y such that $y = f(\boldsymbol{x}),\ f : X \subset \mathbb{R}^D \rightarrow \mathbb{R}$, then the surrogate model is an approximation of the form:

$$\hat{y} = \hat{f}(\boldsymbol{x}, \boldsymbol{x}^{(i)}, y^{(i)}), \qquad i = 1, \ldots, n, \tag{1}$$

where n is the number of training points such that $y^{(i)} = f(\boldsymbol{x}^{(i)})$. The set of training points $\{\boldsymbol{x}^{(i)} \mapsto y^{(i)} = f(\boldsymbol{x}^{(i)}) \mid i = 1, \ldots, n\}$ represents the only insight one can gain into the true model function f and, since it is expensive to obtain, it must be intelligently chosen.

Recently, Machine Learning (ML) has gone down this road and proposed several tools for BBO along these lines. In particular, a large field of research is Bayesian Optimization (BO), which is based on the pioneering Efficient Global Optimization (EGO) algorithm (Jones et al., 1998). Despite its success, BO is stated to be limited to less than 15 parameters (Nayebi et al., 2019; Wang et al., 2016) and a few thousand evaluations (Wang et al., 2018) according to the literature. To overcome this issue, recent research started to explore space partitioning and local modeling. In fact, learning a classifier that locates the samples on a promising subregion of the domain with high probability might be more effective that learning a regressor on the whole domain. Among other partitioning strategies, LA-MCTS (Wang et al., 2020b) recursively learns space partition in a hierarchical manner using Monte Carlo Tree Search (MCTS) (Coulom, 2007). Therefore, in addition to BO, MCTS has also been adapted from control and games to BBO (Munos, 2011; Wang et al., 2020a). However, both the BO and MCTS tools for BBO have rarely been compared to existing BBO methods in a systematic and satisfactory manner. For example, the comparisons in the LA-MCTS paper (Wang et al., 2020b) heavily depend on poor initialization of competitors, while the comparisons in (Turner et al., 2021) – despite mentioning a reference Nevergrad – consider only the simple $(1+1)$ sampling method and not the BBO methods provided by the Nevergrad platform. However, we note some efforts to make neutral and comprehensive comparisons. Extensive comparisons in Cotton (2020) and Rapin & Teytaud (2018) tend to favor more classical methods such as tools from mathematical programming like Cobyla and others (Powell, 1994; Cartis et al., 2018; Powell, 1964) or evolution strategies (Beyer, 2001) like CMA (Hansen & Ostermeier, 2003), possibly equipped with surrogate models (Auger et al., 2005; Khowaja et al., 2021; Luksic et al., 2019). Hutter et al. (2013) ran a well-known BO framework on the most popular benchmark suite, namely BBOB, and got positive results for BO for limited settings only.

In this work, we extend the comparison made in (Hutter et al., 2013) by adding several state-of-the-art solvers and by comparing not only on the BBOB benchmark, but also – as it is closer to ML – on direct policy search for OpenAI Gym problems. We focus on moderate budgets, which are supposed to be the killer application of BO.

2 State of the art: well known tools in BBO, and recent new methods from ML

We compare tools from the ML world to classical BBO tools on classical benchamrks. As baselines classically used in BBO, we consider **CMA**, which stands for CMA-ES, a classical evolution strategy (Hansen & Ostermeier, 2003), **Cobyla**, a tool from mathematical programming (Powell, 1994), and **NGOpt** from Nevergrad (Rapin & Teytaud, 2018). NGOpt is a wizard (Liu et al., 2020; Meunier et al., 2021) that combines many classical algorithms in various ways (Meunier et al., 2021). In the use-cases considered in this work (sequential, low-dimensional, noise-free problems), NGOpt mainly uses CMA, Cobyla, and (1+1)-type sampling equipped with metamodels.

Extensive comparisons have already been proposed in Nevergrad (Meunier et al., 2021), focusing on reproducibility, real-world, and different problem sizes. We propose here additional experiments

architecture search (Zela et al., 2020), expensive global optimization (Bliek et al., 2021) exist, but have a much more limited scope than the above-mentioned packages.

in the well-known BBOB framework (Hansen et al., 2009a), chosen for its simplicity/canonicity. Indeed, the tested framework only needs to optimize dozens of objective functions within given bounds, and the interface typically consists of one or two lines of code (see Appendix B). We also tested several optimizers on OpenAI Gym with Direct Policy Search. Compared to Nevergrad's experiments (Rapin & Teytaud, 2018; 2020), we have a **focus on bounded domains** in BBOB, which makes the comparison easier: most BBO frameworks have a bounded setting mode, while BBOB randomizes the position of the optimum to avoid overfitting, which makes the results more reliable. In addition, BBOB was developed independently of the ML tools, Cobyla and NGOpt, which we are comparing. This should lead to a more neutral comparison. For readers familiar with NGOpt, we note that NGOpt is based in part on the results obtained on YABBOB, which is similar to BBOB but assumes unbounded domains. We nevertheless carefully check that our conclusions hold even without considering NGOpt. Also, we include DefaultCMA (Hansen & Ostermeier, 2003), a version of CMA without the BBOB-specific initialization used in BBOB specifically for the experiments of CMA on BBOB (Hansen et al., 2009a): results are less good, but without big impact on the conclusion.

Compared to other benchmarks, the original BBOB is based on low dimension (≤ 40), average budget ($1000D$), continuous optimization, and a fully sequential setting (Hansen et al., 2021). BO is supposed to be well suited for small dimensions and moderate budgets. Therefore, we adjust the BBOB setting for lower budgets, such as $10D$ and $100D$, as suggested in (Hutter et al., 2013). We note that running BO for large budgets is difficult due to the computational cost. Moreover, BO is usually not competitive for larger budgets. In our OpenAI Gym experiments, we maintain the low-dimensional focus by using tiny neural networks. Our goal is not top performance, but neutral comparison between BBO methods. We chose OpenAI Gym because it has become a standard in the reinforcement learning environment (Thoma, a;b; Manukyan et al., 2019; Lewis-II et al., 2019; Henry & Ernst, 2021; Zubow et al., 2021; Green et al., 2018; Sinha et al., 2020; Rezazadeh et al., 2021).

In the BBOB benchmark with our specific setting using a low budget, we observe that there are elements that have a significant impact on the overall result:

- The LinearSlope function, which has optima in the corners, is difficult for methods that assume that the optimum is supposed to be clearly inside. We have adapted Nevergrad's NGOpt so that it can also search the boundary.
- The precision parameter, $1e-8$ by default, has a significant impact on results. As we focus on a lower budget than BBOB's classical setting, we use $1e-5$.

However, we verified that removing LinearSlope or changing the precision parameter has an impact on the results, but not on the overall picture of comparing new optimization methods from the ML literature with known established methods from the rest of the literature.

3 BACKGROUND

3.1 BAYESIAN OPTIMIZATION

In this section, we now provide a brief description of BO (Jones et al., 1998; Mockus, 2012). BO is a sequential design strategy targeting global optimization of black-box functions that does not assume any functional forms. It is particularly advantageous for problems where the objective function is difficult to evaluate, is a black box with some known structure, relies upon less than 15/20 dimensions, and where no sensitivity and derivative information is available. Since the objective function does not have an explicit mathematical formulation, BO treats it as a random function and places a prior over it. A Kriging model, also known as Gaussian Process Regression (GPR), can be used as a prior probability distribution over functions in BO, also known as Efficient Global Optimization (EGO) (Jones et al., 1998).

BO starts with sampling an initial Design of Experiments (DoE) of size n_0: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_0}]^\top \subseteq \mathrm{D}^{n_0}$ (Santner et al., 2003; Forrester et al., 2008). The corresponding objective function values are denoted as $\mathbf{y} = (f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_{n_0}))^\top$. Conventionally, a centered Gaussian process prior is assumed on the objective function: $f \sim gp(0, K(\cdot,\cdot))$, where $K \colon \mathrm{D} \times \mathrm{D} \to \mathbb{R}$ is a positive definite function – also known as *kernel function* – which computes the autocovariance

of the process. Often, a Gaussian likelihood is taken, leading to a conjugate posterior process (Rasmussen & Williams, 2006), i.e., $f \mid \mathbf{y} \sim gp(\hat{f}(\cdot), K'(\cdot,\cdot))$, where $\hat{f}$ and K' are the posterior mean and covariance function, respectively.

On an unknown point $\mathbf{x}$, $\hat{f}(\mathbf{x})$ yields the maximum a posteriori estimate of $f(\mathbf{x})$ whereas $\hat{s}^2(\mathbf{x}) := K'(\mathbf{x},\mathbf{x})$ quantifies the uncertainty of this estimation. The posterior process is, again, a GPR. Based on the posterior process, promising points are identified via the so-called *infill-criterion*, i.e., by optimizing an *acquisition function* that typically balances $\hat{f}$ with $\hat{s}^2$ (exploitation vs. exploration). A variety of infill-criteria has been proposed in the literature, e.g., Probability of Improvement (Forrester et al., 2008; Mockus, 2012), Expected Improvement (Forrester et al., 2008), and the Upper Confidence Bound (Lai & Robbins, 1985; Srinivas et al., 2012). When a new candidate point is selected by the infill-criterion, it is evaluated and added to the BO data set, which is used to update the GPR posterior. This process is repeated until a stopping condition is met, i.e., a good enough result is located or resources are exhausted.

3.2 Monte Carlo Tree Search

While BO is widely used in ML, other tools have also migrated from ML to BBO. For example, Monte Carlo Tree Search (MCTS) (Coulom, 2007) migrated from trees of bandits for games and control, including alpha-zero (Silver et al., 2016; 2017), to applications in BBO (Munos, 2014; Wang et al., 2020a). In particular, LA-MCTS (Wang et al., 2020a) progressively learns and generalizes promising regions in the problem space by recursively partitioning so that solvers like BO can access these regions to improve their performance.

At any iteration t of the main algorithm, we have a training dataset $D_t = (\mathbf{X}, \mathbf{Y})$ of all the points evaluated so far. A tree node A represents a subregion of the search space Ω_A. Therefore, $D_t \cap \Omega_A$ is the set of all the samples falling in the subregion Ω_A. Let us consider Ω_B and Ω_C as a high and a low performing disjoint subregion of Ω_A. MCTS uses the Monte Carlo simulation to accumulate value estimates that lead to highly rewarding trajectories in the search tree. In other words, MCTS pays more attention to promising nodes (i.e., subregions of the search space), in order to avoid the need to brute force all possibilities. This is done by using a Upper Confidence Bound (UCB) policy. More specifically, each node has an associated UCB value and, during selection, we always chose the child node, associated with a suitable subregion of Ω_A, with the highest UCB value. The statistics used to compute the UCB are (1) n_A, which is the number of samples in $D_t \cap \Omega_A$, and (2) the node value $v_A := \frac{1}{n_A} \sum_{\boldsymbol{x}_i \in D_t \cap \Omega_A} f(\boldsymbol{x}_i)$.

Therefore, in LA-MCTS, which is the MCTS-based optimizer analyzed in this study and compared to the other optimizers, promising regions are found by recursively partitioning the search space using latent actions. In one iteration, LA-MCTS starts building the tree by splitting and then selects a region based on UCB. Finally, sampling is performed in the selected region using BO. In this way, BO avoids over-exploration of the search space and its performance is supposed to be improved, especially for high-dimensional problems.

4 Experimental results

We now discuss our key findings for the two benchmark suites, BBOB and OpenAI Gym. In particular, our results allow for drawing considerations about different BO-based methods on the considered test beds.

4.1 Experimental Setup

Algorithms: From the large collection of existing solvers from the ML world, we have selected the following ones, which we compare to the standard BBO approaches mentioned in Section 2:

- BO: the Bayesian Optimization algorithm (Snoek et al., 2012) implemented in Nevergrad. The python class is a wrapper over the bayes_opt package (Nogueira, 2014).
- Turbo: trust-region inspired algorithm proposed at NeurIPS 2019 (Eriksson et al., 2019). Turbo20 denotes the multi-trust-regions counterpart of Turbo.

- AX: a ML system to help iteratively exploring parameter spaces in order to identify optimal configurations in a resource-efficient manner, proposed in (FacebookResearch, 2020).
- SMAC (Hutter et al., 2011): sequential model-based algorithm for the hyperparameter optimization of ML algorithms, specifically suitable for high dimensions and discrete input dimensions. SMAC2 refers to SMAC-HPO, i.e., SMAC with Hyperparameter Optimization.
- HyperOpt (Bergstra et al., 2015): library for serial and parallel hyperparameter optimization, designed to accommodate Bayesian optimization algorithms based on Gaussian processes and regression trees.
- Optuna (Akiba et al., 2019): automatic hyperparameter optimization software framework, particularly designed for ML. Thanks to its high modularity, the user can dynamically construct the search spaces for the hyperparameters.
- LA-MCTS (Wang et al., 2020a): MCTS-based derivative-free meta-solver that recursively learns space partition in a hierarchical manner. Sampling is then performed in the selected region using BO.

It should be noted that, to estimate the variability of results, we ran SMAC2 in 3 versions, resulting in SMAC2b and SMAC2c, which are exactly equivalent to SMAC2. We only report results from codes that are available and, therefore, reproducible. All results were independently computed by us, i.e., we did not make use of existing tables of results.

Benchmark Problems: For the standard single-objective BBOB benchmark suite, results are averaged over 24 noiseless, scalable test functions (f1 – f24). Although all functions are defined and can be evaluated over $\mathbb{R}^D$, the actual search domain is $[5, 5]^D$. We consider six different dimensions (2, 3, 5, 10, 20, and 40) and 15 random instances per test function and dimension.

Afterward, we work on a multi-deterministic Open AI Gym with tiny neural nets. Here, a random seed is randomly drawn for each run, to avoid overfitting. Within the gym library, we select a few environments to compare algorithms based on simple regret, i.e., the error between the algorithm's recommendation and the optimal solution: MountainCarContinuous-v0 ($D = 8$), NChain-v0 ($D = 40$), Acrobot-v1 ($D = 60$), LunarLanderContinuous-v2 ($D = 88$), GuessingGame-v0 ($D = 24$), MountainCar-v0 ($D = 12$), CartPole-v1 ($D = 28$), CartPole-v0 ($D = 28$). We selected those problems as being, in our context of tiny nets, sufficiently challenging and not too hard, i.e., not all algorithms performing equally. Here we use a neural factor of 1, i.e., the scaling coefficient used by Nevergrad to choose the size of neural networks is set to 1. The dimension is a consequence of the number of neurons, which in turn is based on the scaling factor, and the number of inputs and outputs. Aggregate plots based on average winning rates are also presented, where we include problems with dimensions $D < 50$ for a neural factor of 1 and $D \leq 264$ for larger networks defined by setting the neural factor to 3 inside Nevergrad. Both regret and winning percentage are evaluated at fixed budgets of 25, 50, 100, 200, 400, and 800 function evaluations.

4.2 Results on BBOB

As suggested in (Hutter et al., 2013), we focus on a lower budget ($10D$ or $100D$) compared to classical experiments with BBOB. In case of crash of a method, we rerun it with the remaining budget. Figs. 1 and 2 present results with budget equal to $10D$ and $100D$, respectively, in dimension D. The x-axis is the budget divided by the dimension, in logarithmic scale, while the y-axis shows the frequency of problem solving, i.e., the higher, the better. We use the plots built by COCO/BBOB, which include a horizontal line at the end (the rightmost points are not actual data, but are just for readability). The last x-tick corresponds to the allowed budget: the framework sometimes plots values that are slightly above the budget, when the code under consideration exceeds the limit, but the correct x-axis value is used.

Our experiments for a budget of $10D$ reproduce the results in (Hutter et al., 2013), where SMAC outperforms CMA. Hutter et al. (2013) mentioned that BO, specifically SMAC, can compete with CMA on BBOB for low budgets such as 10D. However, CMA is poorly suited for such a context. Although we tested fewer optimizers for the 20D and 40D cases due to the high computational cost, it is interesting to note that Cobyla, which is simply based on linear interpolation, often performs better than all other solvers. Although there is no tuning for our present results from Cobyla on

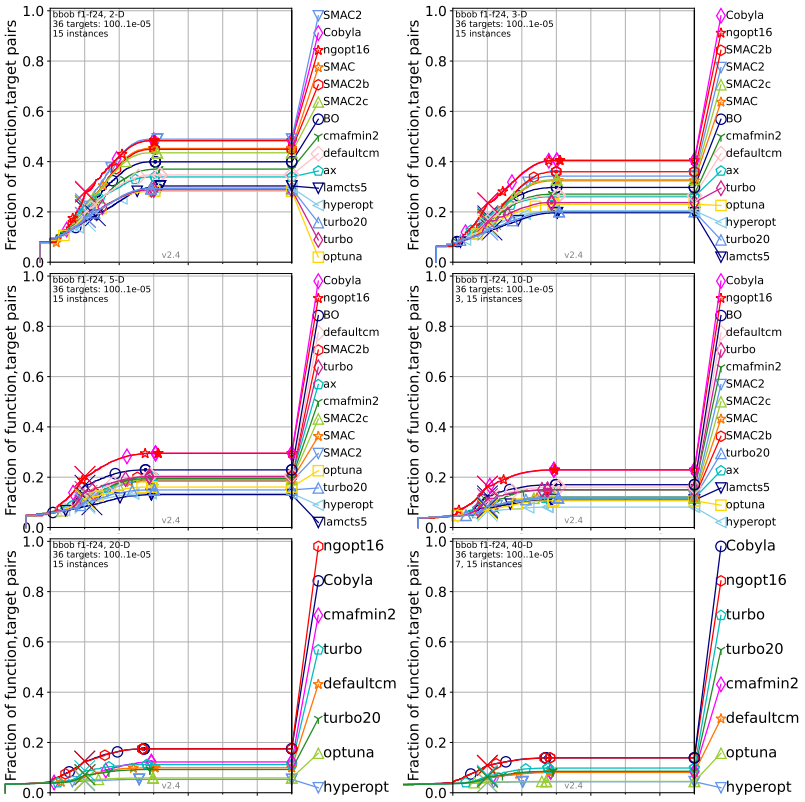


Figure 1: BBOB with precision $1e-5$ and budget $10D$. X-axis: budget/dimension in log-scale. Y-axis: frequency of solving at the requested precision. The right-most point after each horizontal line is added for readability of lines by the BBOB framework. Plots for dimensions 2, 3, 5, 10, 20, and 40 are shown.

BBOB, it outperformed all BO-based algorithms. Cobyla is the algorithm chosen by a wizard like NGOpt for the test cases considered. This is the reason for the similar performance and also a proof of the good tuning of the Nevergrad wizard NGOpt.

4.3 Direct Policy Search on OpenAI Gym

LA-MCTS (Wang et al., 2020a) claims good results for some OpenAI Gym problems, but the plots provided in the paper clearly show the influence of bad initialization of its competitors. We therefore provide an independent comparison. To this end, we consider Ng-Full-Gym, which is Nevergrad's direct policy search applied to OpenAI Gym. This is optimizing a neural network as a controller for OpenAI Gym problems. However, we consider a small version (small number of neurons, low budget) for matching the low-budget context of the present work. The values of the objective function are noisy.

We consider the multi-deterministic case, i.e., we randomly draw a seed and keep it for each optimization run. This is somewhat analogous to random perturbation of the optimum in classical BBO benchmarks in the sense that the objective function is deterministic but drawn randomly. DE, PSO, TwoPointsDE, CMandAS2, CMA, DiagonalCMA and other solvers can all be found in (Rapin & Teytaud, 2018) and are classical BBO tools from the black-box optimization community. QO-RandomSearch (Rahnamayan et al., 2007), MetaTuneRecentering (Meunier et al., 2020), MetaRecentering (Cauwet et al., 2019) are variants of random search that are fully parallel and reduce redundancies compared to random search.

On the OpenAI Gym task, we obtain good results for some BO methods for low budgets. We plot simple regrets in Fig. 3. Here, the lower the curve, the better. It can be noted that Cobyla does not perform as well as for BBOB, while PSO performs impressively well. As shown in the aggregated

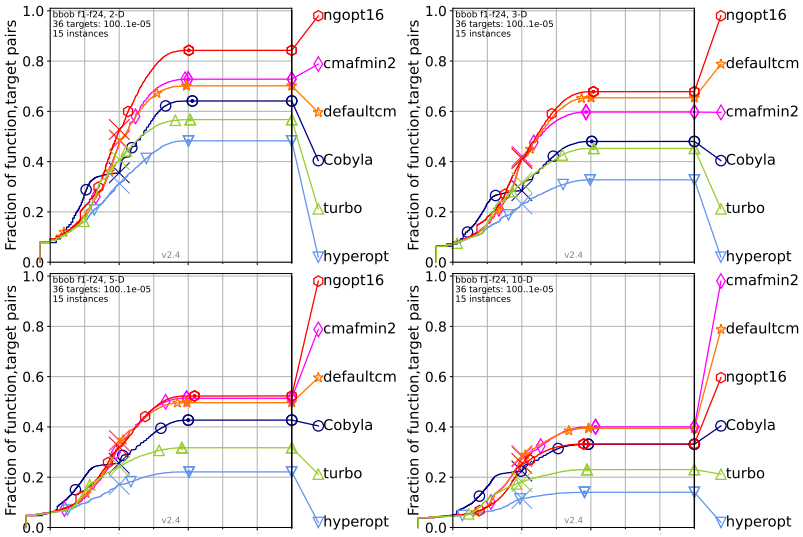


Figure 2: As Fig. 1 but with a $100d$ budget. Turbo's speed makes its for this higher budget. Results are shown for dimensions 2, 3, 5, and 10.

comparison in Fig. 4, it is the only algorithm that reliably outperforms all other methods by a significant margin. This is actually good for fast low-precision approximations on difficult problems. After PSO, many other methods perform more or less equivalently, with some BO-based solvers also showing potential. At budget 400, SMAC and SMAC2 actually perform better than PSO, which has the upper hand for all the other evaluation budgets. For higher dimensions, Fig. 5 shows that BO becomes less competitive, and for high budgets, Meunier et al. (2021) (or Appendix F) shows that (possibly Diagonal) CMA or wizards perform best. Nevertheless, BO was not too far off on the smallest OpenAI Gym problems, and would have been successful on the lowest dimensions if PSO had not been included: many of the reasonably good methods were some form of BO in Fig. 4 (though not for Fig. 5, at larger dimension, nor in (Meunier et al., 2021), for larger budget).

5 Conclusion

Both BBOB and Nevergrad (see Appendix A), with their randomized optima, many repetitions, and large number of test functions, combined with a framework that prevents users from optimizing hyperparameters for each objective function separately, are more reliable than ad hoc benchmarks designed specifically for a new method.

Our results provide insight into the comparison between different BO methods. SMAC performed best among the tested BO methods for both BBOB and OpenAI Gym. We note that it was also part of the best solution for the BBO challenge (Turner et al., 2021; Awad et al., 2020). Turner et al. (2021) highlights Turbo as a strong BO, but SMAC was consistently better in our experiments. However, Turbo and HyperOpt have the advantage of being computationally cheaper. HyperOpt did not perform poorly on low-budget neurocontrol for OpenAI-Gym. We note that this benchmark is very sensitive to variable scaling. While BBOB focuses on uniform translations of optima, a good method here should be able to check the center of the domain and quickly move closer to it if needed.

More generally, despite efforts to consider the most favorable settings for ML-based methods (in particular for OpenAI Gym, where we restricted the setting until the dimension and budget match the capabilities of BO), and despite the fact that tools from ML are not penalized in the present paper for their internal computational costs, there is no tool from ML that outperforms classical algorithms in our experiments, neither in BBOB nor in the direct policy search for OpenAI Gym.

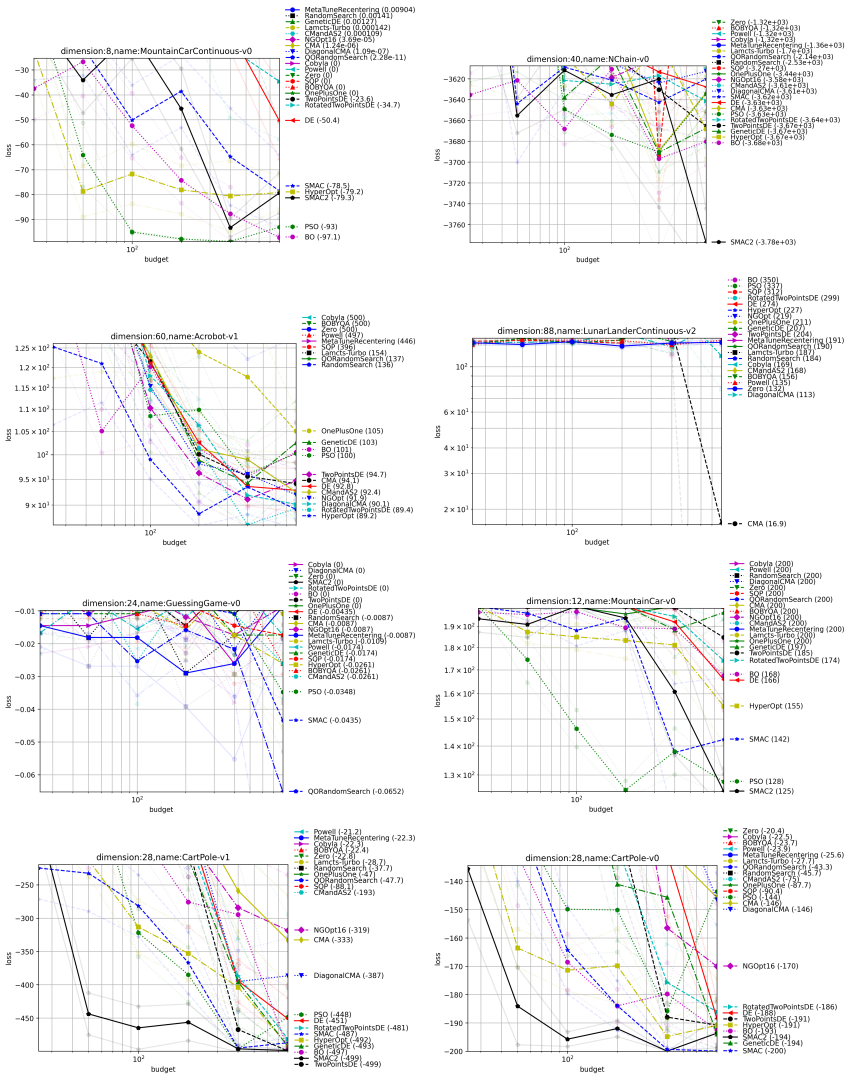


Figure 3: Multi-deterministic Open AI Gym with tiny neural net: a random seed is randomly drawn for each optimization run, so that overfitting is more difficult. See Fig. 4 for an aggregated view.

FURTHER WORK & LIMITATIONS

In future work, we plan to compare BO and other methods in a discrete setting as well. We plan to look for reproducible, error-free benchmarks where BO performs better than the classical methods. At the moment, there is only anecdotal evidence that BO performs well in BBO, but our low-dimensional low-budget control tasks in Fig. 4 suggest that BO may perform well in some cases, although it suffers from competition with PSO. These neurocontrol tasks are quite sensitive to the scaling of the variables. There is room for further investigation, PSO might be good at finding the right scale around the center of its domain. In this context, we note the impressive performance of quasi-opposite sampling despite its extreme simplicity: we plan to investigate algorithms that are robust to scaling, e.g., using Cauchy sampling or bet-and-run over multiple scalings, or bandit algorithms for choosing between different scalings. Our benchmarks cover only a limited range of problems: we refer to (Rapin & Teytaud, 2018; Gould et al., 2015; Gallagher & Saleem, 2018; Li et al., 2013) and the many variants of BBOB (Hansen et al., 2009a) for more. Another possibility for which the tools from BO might be well suited are benchmarks that focus on conditional variables.

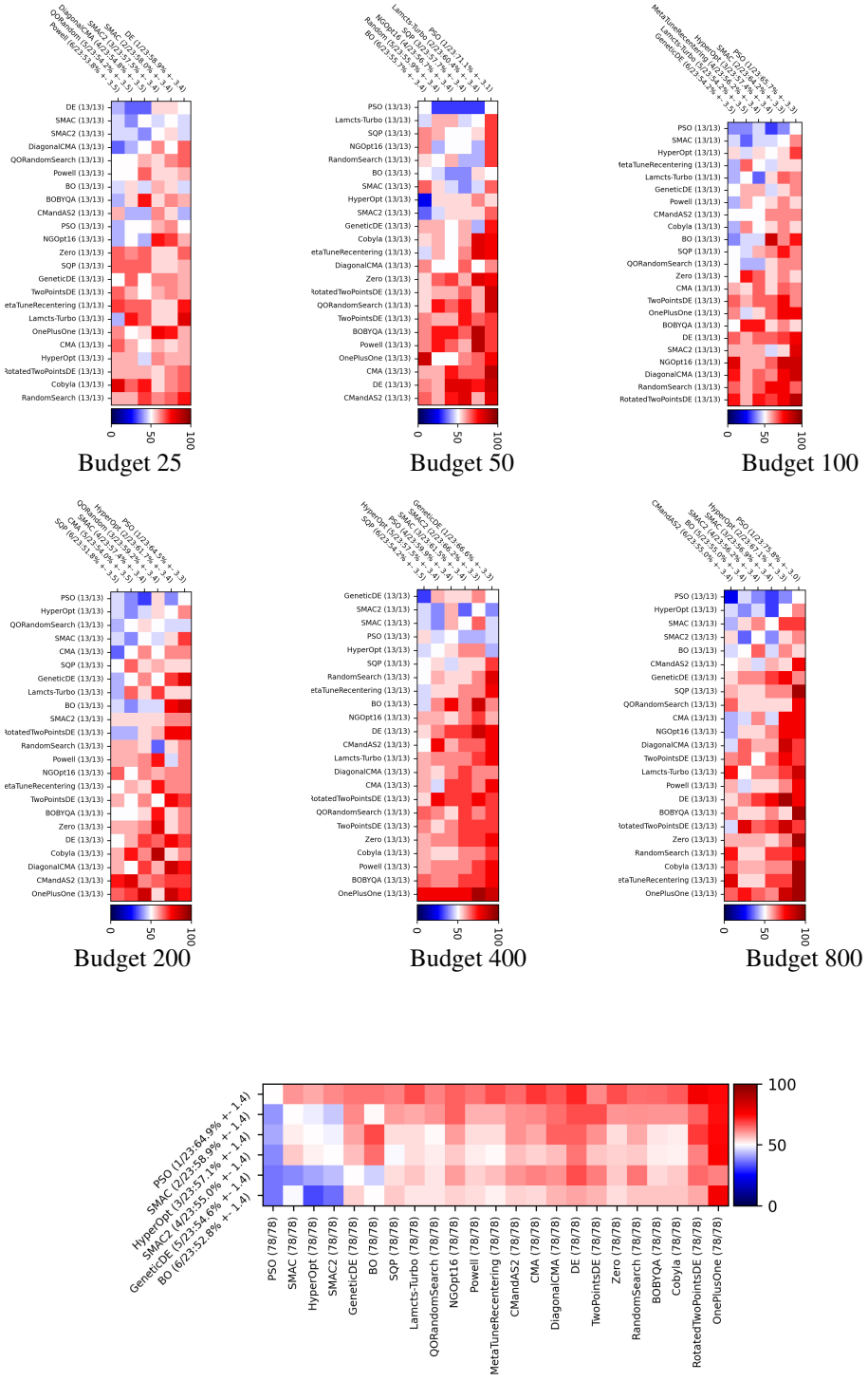


Figure 4: Same problem as Fig. 3, but aggregated comparison as provided by Nevergrad, best methods first: row A col B shows the frequency at which method A outperformed method B for the given budget. 13 distinct problems per budget. We include only problems for which dimension is $D < 50$. Methods are ranked per average winning rate. Note that winning rates are all very close to each other: only PSO is significantly better. Fig. 5 presents similar experiments but with bigger neural nets. Fig. 6 extends the present results to budgets 1600 and 3200.

ETHICS STATEMENT

The authors declare they read the ICLR Code of Ethics and adhere to the general ethical principles it provides. Moreover, the authors declare that they have no conflicts of interests.

REPRODUCIBILITY STATEMENT

The results in this paper are easily reproducible. Appendix B contains the essential lines of code to run each of the considered ML-based black-box algorithms on the COCO/BBOB benchmark suite. This can be used by cloning a repository and following the steps provided in the COCO/BBOB documentation, as described in Appendix C. The OpenAI Gym benchmark is already included in Nevergrad. Therefore, we apply the standard Nevergrad's direct policy search to it. The steps to perform optimization studies using Nevergrad are available at `https://facebookresearch.github.io/nevergrad/` and developed in Appendix D of the present paper.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv:1907.10902 [cs, stat]*, July 2019.

Anne Auger, Marc Schoenauer, and Olivier Teytaud. Local and global order 3/2 convergence of a surrogate evolutionary algorithm. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pp. 857–864, New York, NY, USA, June 2005. Association for Computing Machinery. ISBN 978-1-59593-010-1. doi: 10.1145/1068009.1068154.

Noor Awad, Gresa Shala, Difan Deng, Neeratyoy Mallik, Matthias Feurer, Katharina Eggensperger, Andre' Biedenkapp, Diederick Vermetten, Hao Wang, Carola Doerr, Marius Lindauer, and Frank Hutter. Squirrel: A switching hyperparameter optimizer, 2020.

Ishan Bajaj, Akhil Arora, and M. M. Faruque Hasan. Black-Box Optimization: Methods and Applications. In Panos M. Pardalos, Varvara Rasskazova, and Michael N. Vrahatis (eds.), *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, Springer Optimization and Its Applications, pp. 35–65. Springer International Publishing, Cham, 2021. ISBN 978-3-030-66515-9. doi: 10.1007/978-3-030-66515-9_2.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

Hans-Georg Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Heidelberg, 2001.

Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artif. Intell.*, 237:41–58, 2016. doi: 10.1016/j.artint.2016.04.003. URL `https://doi.org/10.1016/j.artint.2016.04.003`.

Laurens Bliek, Arthur Guijt, Rickard Karlsson, Sicco Verwer, and Mathijs de Weerdt. Expobench: Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *CoRR*, abs/2106.04618, 2021. URL `https://arxiv.org/abs/2106.04618`.

Coralia Cartis, Jan Fiala, Benjamin Marteau, and Lindon Roberts. Improving the flexibility and robustness of model-based derivative-free optimization solvers, 2018.

Marie-Liesse Cauwet, Camille Couprie, Julien Dehos, Pauline Luc, Jérémy Rapin, Morgane Riviere, Fabien Teytaud, and Olivier Teytaud. Fully parallel hyperparameter search: Reshaped space-filling. *arXiv preprint arXiv:1910.08406. To appear in Proc. of ICML 2020*, 2019.

Peter Cotton. An Introduction to Z-Streams (and Collective Microprediction) — LinkedIn. https://www.linkedin.com/pulse/short-introduction-z-streams-peter-cotton-phd/, 2020.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pp. 72–83. Springer-Verlag, 2007.

Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281*, October 2018. URL https://arxiv.org/abs/1810.05281.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/6c990b7aca7bc7058f5e98ea909e924b-Paper.pdf.

FacebookResearch. Ax - adaptive experimentation. ax.dev, 2020.

Alexander I. J. Forrester, András Sóbester, and Andy J. Keane. *Engineering Design via Surrogate Modelling - A Practical Guide*. John Wiley & Sons Ltd., 2008. ISBN 978-0-470-06068-1.

Marcus Gallagher and Sobia Saleem. Exploratory landscape analysis of the mlda problem set. In *PPSN'18 workshop*, 2018.

Nicholas Gould, Dominique Orban, and Philippe Toint. Cutest: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

Sam Green, Craig M. Vineyard, and Çetin Kaya Koç. Impacts of Mathematical Optimizations on Reinforcement Learning Policy Performance. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2018. doi: 10.1109/IJCNN.2018.8489519.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 11(1), 2003.

Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, France, 2009a.

Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noisy Functions Definitions. January 2009b.

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, January 2021. ISSN 1055-6788. doi: 10.1080/10556788.2020.1808977.

Robin Henry and Damien Ernst. Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems, 2021.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello (ed.), *LION*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer, 2011. ISBN 978-3-642-25565-6. URL http://dblp.uni-trier.de/db/conf/lion/lion2011.html#HutterHL11.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pp. 1209–1216, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319645. doi: 10.1145/2464576.2501592. URL https://doi.org/10.1145/2464576.2501592.

Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Aclib: A benchmark library for algorithm configuration. In *Proc. of Learning and Intelligent Optimization (LION'14)*, volume 8426 of *LNCS*, pp. 36–40. Springer, 2014. doi: 10.1007/978-3-319-09584-4_4. URL https://doi.org/10.1007/978-3-319-09584-4_4.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, December 1998. ISSN 0925-5001, 1573-2916. doi: 10.1023/A:1008306431147.

Kainat Khowaja, Mykhaylo Shcherbatyy, and Wolfgang Karl Härdle. Surrogate models for optimization of dynamical systems, 2021.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

W. Cannon Lewis-II, Mark Moll, and Lydia E. Kavraki. How much do unstated problem constraints limit deep robotic reinforcement learning? *CoRR*, abs/1909.09282, 2019. URL http://arxiv.org/abs/1909.09282.

Xiaodong Li, Ke Tang, Mohammmad Nabi Omidvar, Zhenyu Yang, and Kai Qin. Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. 01 2013.

Jialin Liu, Antoine Moreau, Mike Preuss, Jeremy Rapin, Baptiste Roziere, Fabien Teytaud, and Olivier Teytaud. Versatile black-box optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pp. 620–628, 2020.

Ziga Luksic, Jovan Tanevski, Saso Dzeroski, and Ljupco Todorovski. Meta-model framework for surrogate-based parameter estimation in dynamical systems. *IEEE Access*, 7:181829–181841, 2019. doi: 10.1109/ACCESS.2019.2959846.

A. Manukyan, M. A. Olivares-Mendez, M. Geist, and H. Voos. Deep reinforcement learning-based continuous control for multicopter systems. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 1876–1881, 2019. doi: 10.1109/CoDIT.2019.8820368.

Pascal Kerschke Marcus Gallagher, Mike Preuss. Ppsn'18 machine learning and data analysis (mlda) problem set, v1.0. https://drive.google.com/file/d/1fc1sVwoLJ0LsQ5fzi4jo3rDJHQ6VGQ1h/view, 2018.

Laurent Meunier, Carola Doerr, Jérémy Rapin, and Olivier Teytaud. Variance reduction for better sampling in continuous domains. In *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pp. 154–168. Springer, 2020.

Laurent Meunier, Herilalaina Rakotoarison, Jeremy Rapin, Paco Wong, Baptiste Roziere, Olivier Teytaud, Antoine Moreau, and Carola Doerr. Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking, 2021. URL https://openreview.net/forum?id=4D4Rjrwaw3q.

Jonas Mockus. *Bayesian Approach to Global Optimization: Theory and Applications*. Springer Science & Business Media, December 2012. ISBN 978-94-009-0909-0.

Rémi Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pp. 783–791, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

Rémi Munos. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. Technical report, 2014.

Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. A Framework for Bayesian Optimization in Embedded Subspaces. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4752–4761. PMLR, May 2019.

Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014. URL https://github.com/fmfn/BayesianOptimization.

Michael J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

Michael J.D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pp. 51–67. Springer Netherlands, 1994. ISBN 978-94-015-8330-5.

S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Quasi-oppositional differential evolution. In *2007 IEEE Congress on Evolutionary Computation*, pp. 2229–2236, Sep. 2007.

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

J. Rapin and O. Teytaud. Dashboard of results for Nevergrad platform. `https://dl.fbaipublicfiles.com/nevergrad/allxps/list.html`, 2020.

Jeremy Rapin, Marcus Gallagher, Pascal Kerschke, Mike Preuss, and Olivier Teytaud. Exploring the MLDA benchmark on the nevergrad platform. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, pp. 1888–1896, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6748-6. doi: 10.1145/3319619.3326830.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass, 2006. ISBN 978-0-262-18253-9.

Farhad Rezazadeh, Hatim Chergui, Luis Alonso, and Christos Verikoukis. Continuous multi-objective zero-touch network slicing via twin delayed ddpg and openai gym, 2021.

Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer-Verlag, New York, 2003. ISBN 978-1-4419-2992-1. doi: 10.1007/978-1-4757-3799-8.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL `http://arxiv.org/abs/1712.01815`.

Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning, 2020.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2951–2959, 2012.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *IEEE Transactions on Information Theory*, 58(5):3250–3265, May 2012. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2011.2182033.

Ke Tang, Xiaodong Li, P. N. Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, University of Science and Technology of China, 2010.

Martin Thoma. RL-agents. `https://martin-thoma.com/rl-agents/`, a.

Martin Thoma. . `https://martin-thoma.com/q-learning/`, b.

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020, 2021.

Konstantinos Varelas, Ouassim Ait ElHara, Dimo Brockhoff, Nikolaus Hansen, Duc Manh Nguyen, Tea Tusar, and Anne Auger. Benchmarking large-scale continuous optimizers: The bbob-largescale testbed, a COCO software guide and beyond. *Appl. Soft Comput.*, 97(Part):106737, 2020. doi: 10.1016/j.asoc.2020.106737. URL https://doi.org/10.1016/j.asoc.2020.106737.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using Monte Carlo Tree Search. *arXiv:2007.00708. To appear in Proc. of Neurips 2020*, 2020a.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. *arXiv:2007.00708 [cs, math, stat]*, July 2020b.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched Large-scale Bayesian Optimization in High-dimensional Spaces. *arXiv:1706.01445 [cs, math, stat]*, May 2018.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. Bayesian Optimization in a Billion Dimensions via Random Embeddings. *arXiv:1301.1942 [cs, stat]*, January 2016.

Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proc. of 8th International Conference on Learning Representations (ICLR'20)*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJx9ngStPH.

Anatolij Zubow, Sascha Rösler, Piotr Gawłowicz, and Falko Dressler. GrGym: When GNU Radio Goes to (AI) Gym. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, HotMobile '21, pp. 8–14, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383233. doi: 10.1145/3446382.3448358. URL https://doi.org/10.1145/3446382.3448358.

A DISCUSSION: DIFFERENCES BETWEEN BBOB AND NEVERGRAD

There are differences between the main benchmarking suites.

A.1 DOMAINS

First, working in unbounded domains with a Gaussian distributed optimum (as in some benchmarks in Nevergrad) leads to differences compared to benchmarks on bounded domains such as BBOB, especially for functions with an optimum at the boundary. Both contexts look interesting, but some adaptation of NGOpt was needed to make it tackle optima on the boundary.

A.2 BUDGETS AND INDEPENDENCE

In BBOB, when specifying a maximum budget of $100D$, the results plotted for a lower budget of $10D$ are obtained as a truncation of the $100D$-budget runs. On the contrary, Nevergrad runs each budget separately, which is more expensive from a computational point of view, but gives a more complete picture for the different budgets. This can be remedied by launching distinct runs for different budgets for BBOB.

A.3 PARALLELIZATION

Nevergrad was more suitable for testing very slow algorithms like AX (FacebookResearch, 2020) because it is easy to massively parallelize it on a cluster.

B BBOB INTERFACES

A strength of BBOB is that the interfacing is quite small, making reproducibility feasible.

```
% Nevergrad's NGOpt.
% Also SMAC, SMAC2, AX, BO: using Nevergrad's API.
ng.optimizers.NGOpt(ng.p.Array(lower=lbounds, upper=ubounds, shape=[dim]), num_workers=1, budget=evals).minimize(f)

% HyperOpt.
fmin(fn=lambda x: f([x['w'+str(i)] for i in range(dim)]), space={'w'+str(i): hp.uniform('w' + str(i), -5, 5) for i in range(dim)}, algo=tpe.suggest, max_evals=evals)

% Optuna.
class OptunaObjective(object):
    def __init__(self, problem):
        self.problem = problem

    def __call__(self, trial):
        x = []
        for i in range(self.problem.dimension):
            x.append(trial.suggest_float("x{}".format(i), problem.lower_bounds[i], problem.upper_bounds[i]))
        return self.problem(x)

study = optuna.create_study(direction="minimize")
study.optimize(OptunaObjective(problem), n_trials=evalsleft())

% Turbo.
class turbo_function:
    def __init__(self, dim=len(lbounds)):
        self.dim = dim
        self.lb = lbounds
        self.ub = ubounds

    def __call__(self, x):
```

```
        assert len(x) == self.dim
        assert x.ndim == 1
        assert np.all(x <= self.ub) and np.all(x >= self.lb)
        return f(x)

my_turbo = Turbo1(f=turbo_function(), lb=lbounds, ub=ubounds, max_evals=
    evals, n_init=min(evals, 20))

my_turbo.optimize()

% LA-MCTS.
% We tested several successive variants of the code,
% without much impact: the version below is the last.
% We also tested several values
% of ninits, without much change.

agent = MCTS(lb = f.lb,
             ub = f.ub,
             dims = f.dims,
             ninits = 40,  # We tested variants without much change.
             func = f
             )
agent.search(iterations = evalsleft())
```

C How to interface your favorite optimization method with BBOB

First you have to git clone COCO/BBOB at `https://github.com/numbbo/coco`. There are 7 steps, but this is pretty simple, as one just needs to download the package and run experiments. There is a single file to modify, where we optimize a given function f between -5 and 5 in dimension D.

Hundreds of examples are available at `https://numbbo.github.io/data-archive/bbob/`.

D How to run the experiments on OpenAI Gym

OpenAI Gym was recently introduced in Nevergrad. Therefore, starting experiments is straightforward than interfacing with COCO. After cloning Nevergrad at `https://github.com/facebookresearch/nevergrad.git`, experiments are launched with the following command line:

```
python -m nevergrad.benchmark ng_full_gym --repetitions=10 --plot
```

However, in order to run a reduced experiment like the one presented in this paper, we changed the definition of the budget, dimension, and scaling (budget $25, 50, 100, 200, 400, 800$, scaling-factor 1, and add a limit 40 to the dimension) in the `ng_full_gym` experiment in `nevergrad/benchmarks/gymexperiments.py` (Line 80) for obtaining the setup as in Section 4.1.

E Bigger neural nets for OpenAI Gym

Fig. 5 is a more complete version of the Ng-Full-Gym problem, where the neural_factor parameter, which scales the size of the neural networks, is equal to 3. Dimensions up to 264 are considered. This benchmark is unbounded: the scale of algorithms (the standard deviation of the first samples) is critical, and makes comparisons difficult.

We note that PSO is still quite good. However, this remains small compared to traditional direct policy search. Quasi-Opposite random search is still strong, although it is a fully parallel, feedback-free method. This confirms that scaling properly and avoiding redundancy can be sufficient for such

moderate budgets. Quasi-Opposite (Rahnamayan et al., 2007) sampling is a random search that combines many scalings (by randomly drawing the scale) and reduces redundancies (by mirroring).

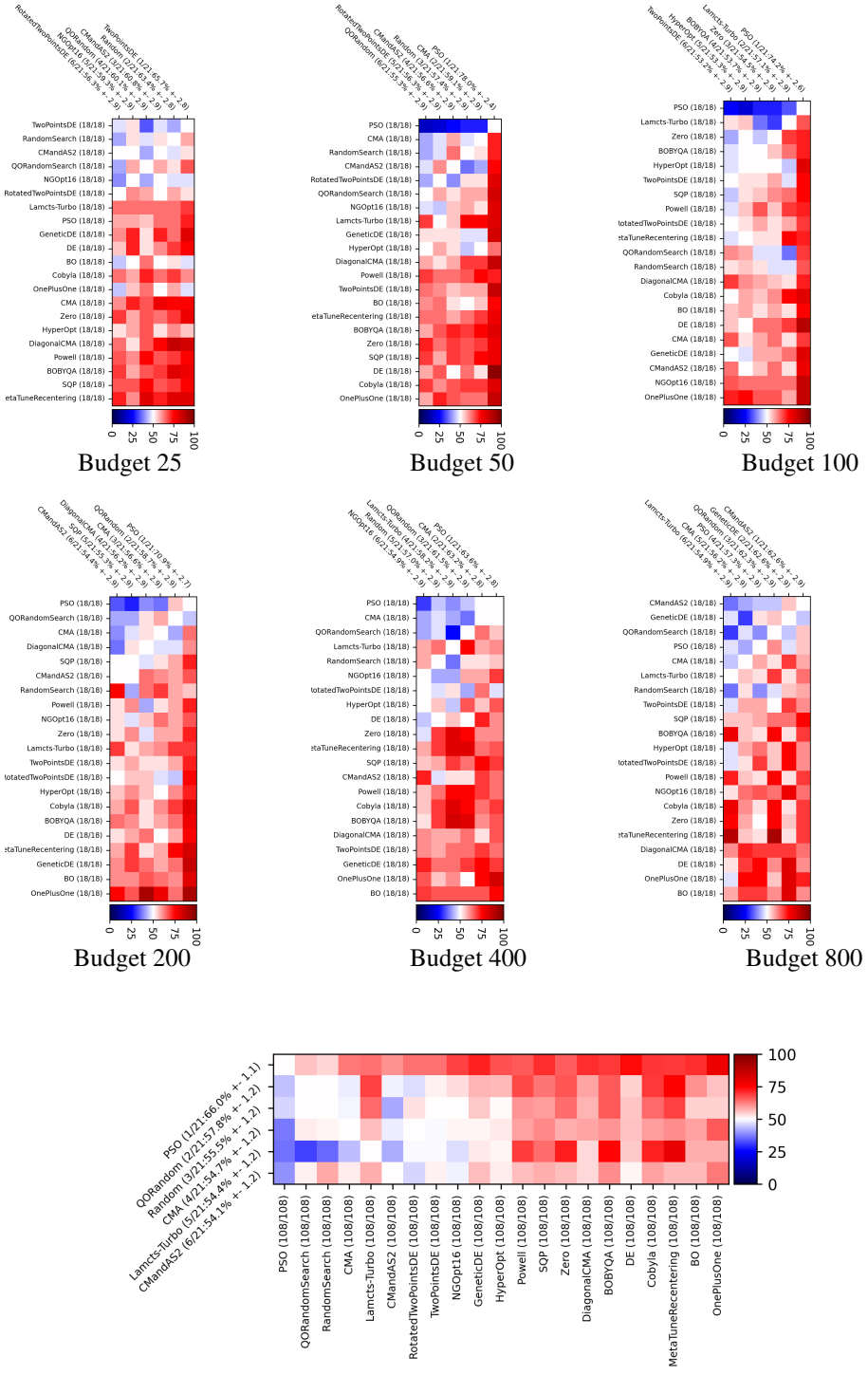
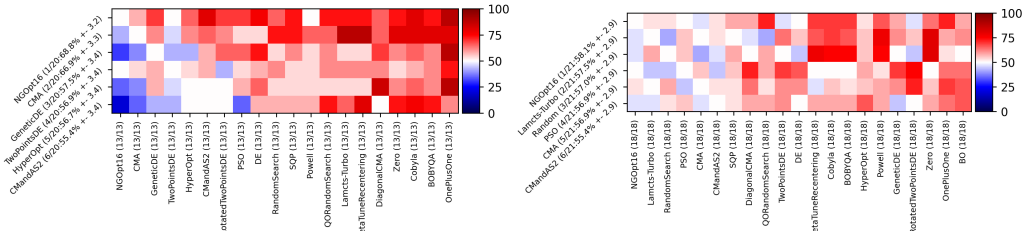


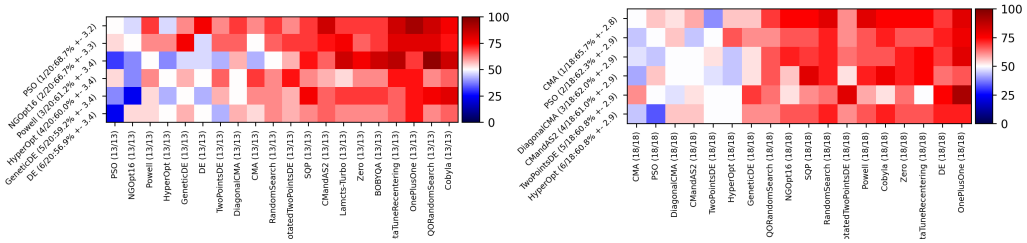
Figure 5: Same as Fig. 4, but with bigger nets (neural factor 3 in Nevergrad). 18 distinct problems per budget. We truncated at dimension ≤ 264. Dimension ranges from 24 to 264 instead of 8 to 40 in Fig. 4. We note that PSO still dominates by far. Due to the computational cost, it was not possible to finish the runs for SMAC. Fig. 6 extends the present results to budgets 1600 and 3200.

F BUDGETS 1600 AND 3200

While Figs. 4 and 5 present results restricted to budget ≤ 800, Fig. 6 presents results for a larger budget of 1600 evaluations applied to multi-deterministic Open AI Gym with both tiny and bigger neural nets. As in Meunier et al. (2021), CMA or NGOpt get better as the budget grows. We also find that, while most BO-based methods weaken, HyperOpt performs satisfactorily. The reader should refer to Meunier et al. (2021) for larger budgets.



Budget 1600



Budget 3200

Figure 6: Extension of Fig. 4 and 5, for budget 1600 (top row) and 3200 (bottom row) for algorithms that were sufficiently fast (faster than 3 days wall-clock time).