Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB

Maria Laura Santoni, Elena Raponi, Renato de Leone, Carola Doerr

Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB

MARIA LAURA SANTONI, Sorbonne Université, CNRS, LIP6, France
ELENA RAPONI, Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
RENATO DE LEONE, School of Science and Technology, University of Camerino, Italy
CAROLA DOERR, Sorbonne Université, CNRS, LIP6, France

Bayesian Optimization (BO) is a class of surrogate-based black-box optimization heuristics designed to efficiently locate high-quality solutions for problems that are expensive to evaluate, and therefore allow only small evaluation budgets. BO is particularly popular for solving numerical optimization problems in industry, where the evaluation of objective functions often relies on time-consuming simulations or physical experiments. However, many industrial problems depend on a large number of parameters. This poses a challenge for BO algorithms, whose performance is often reported to suffer when the dimension grows beyond 15 decision variables. Although many new algorithms have been proposed to address this, it remains unclear which one is best suited for a specific optimization problem.

In this work, we compare five state-of-the-art high-dimensional BO algorithms with vanilla BO, CMA-ES, and random search on the 24 BBOB functions of the COCO environment at increasing dimensionality, ranging from 10 to 60 variables. Our results confirm the superiority of BO over CMA-ES for limited evaluation budgets and suggest that the most promising approach to improve BO is the use of trust regions. However, we also observe significant performance differences for different function landscapes and budget exploitation phases, indicating improvement potential, e.g., through hybridization of algorithmic components.

CCS Concepts: • **Mathematics of computing** → **Continuous optimization**; • **Theory of computation** → **Bayesian analysis**; **Design and analysis of algorithms**.

Additional Key Words and Phrases: Black-box optimization, Bayesian Optimization, High-dimensional Bayesian Optimization, Benchmarking

1 INTRODUCTION

In countless areas of science, engineering, and beyond, researchers and developers deal with free parameters that can be tuned to achieve a specific goal. With the increase in computing power and simulation methods, optimization techniques have become always more popular in industry. They replace the manual trial-and-error method for parameter tuning, which is highly dependent on prior knowledge and expertise about the particular research area and subject, and which requires many trials before a satisfactory parameter configuration is found, without any guarantee of its optimality. In many application areas such as machine learning, neural network design, robotics, aerospace, and mechanical design, optimization deals with black-box problems, i.e., problems where the structure of the objective function, its derivatives, and/or the constraints is unknown, unexploitable, or non-existent. This is the case, for example, in design and prototyping processes, where engineers simulate the performance of new products with increasingly accurate numerical models and simulations. This makes evaluating a particular parameter setting extremely costly, reducing the affordable budget for simulations and making it impossible to fully explore the design space. When limited evaluation budgets are available in optimization, it is a common practice to rely on surrogate models, i.e., approximations that mimic the behavior of the expensive and unknown objective function to be optimized while being computationally cheaper to evaluate.

Authors' addresses: Maria Laura Santoni, Sorbonne Université, CNRS, LIP6, Paris, France, maria-laura.santoni@lip6.fr; Elena Raponi, Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands, e.raponi@liacs.leidenuniv.nl; Renato De Leone, School of Science and Technology, University of Camerino, Camerino, Italy, renato.deleone@unicam.it; Carola Doerr, Sorbonne Université, CNRS, LIP6, Paris, France, carola.doerr@lip6.fr.

One of the most commonly used surrogate-based optimization methods is Bayesian Optimization (BO) [17, 45]. Through careful modeling – based on Gaussian process regression (GPR) – and intelligent search for candidate solutions – through the optimization of an acquisition function – BO algorithms can deliver impressive optimization performance even with small evaluation budgets [17]. The potential of BO has been demonstrated in hundreds of studies across a wide range of domains, such as chemistry and material science [19, 30, 36], biology, engineering and design [37, 49, 50], robotics [5, 34, 44], algorithm configuration [32], hyperparameter tuning [35, 46, 56], and automated machine learning [33, 41]. However, when the dimensionality of the problem exceeds 15 variables, the performance of BO deteriorates due to the so-called *curse of dimensionality* [2]. Scaling BO to high-dimensional spaces is challenging, because its statistical and computational complexity increases with dimension: the number of points queried to satisfactorily cover the search space increases exponentially with the dimension, and optimizing the acquisition function requires more and more computational power, being a non-convex optimization problem on the same design space itself.

Related work: In recent years, characterized by increasingly complex systems and large and high-dimensional data, great efforts have been made to extend BO to higher dimensions. Various strategies have been proposed. According to [4], these are mainly based on (but not limited to) one of the following methods: Variable selection, additive models, linear and nonlinear embeddings, and trust regions. All of these strategies have advantages and disadvantages (see Figure 1), and it is not clear which one is best for which optimization scenario. In Section 3 we discuss these categories further and give some examples. However, for a more comprehensive overview, we refer the reader to available surveys on this topic [4, 42], even though they are purely informative and lack an experimental study comparing the methods. On the other hand, comparative studies have been conducted to present new algorithms such as SMAC [31], REMBO [60], TuRBO [14], SAASBO [13], etc., but they are either outdated or limited to showing the potential of the proposed algorithm rather than presenting a comprehensive and unbiased performance comparison. Our work aims at filling this gap.

Disclaimer: In line with [4] we refer to the setting with dimension 60 as *high-dimensional*, even if BO-approaches for problems with several thousands of variables have been studied [60].

Our contribution: In this article we present the results of a benchmarking study that follows the standardized, well-established guidelines for unbiased performance comparison in numerical black-box optimization. With the goal to obtain a first overview over which high-dimensional BO (HDBO) approaches to favor for which problem characteristics, we benchmark BO variants that are specifically designed for high-dimensional, low-budget optimization problems. We also include in our comparison three standard baselines: vanilla BO [28], the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [26], and random search [27]. To obtain interpretable results, we focus on the 24 (noiseless) functions of the Black-Box Optimization Benchmarking (BBOB) suite from the COCO environment [22] and compare algorithm performance, in terms of both convergence speed and CPU time, for what we consider small evaluation budgets; motivated by our target applications, we set the budget to $10D + 50$. It is important to note that our study aims to fairly compare diverse HDBO algorithms by reporting metrics like model fitting time and acquisition function optimization time, utilizing open-source benchmark functions. This provides a leveled ground for algorithm performance analysis and for understanding how it is influenced by the inner mechanisms of algorithm classes. Key findings from our experiments are that (1) Vanilla BO performs better than CMA-ES for small dimensions and low budget, (2) many of the algorithms that aim to scale BO to high-dimensional spaces outperform vanilla BO and CMA-ES, with the convergence gap widening as the dimension increases, and (3) among the compared algorithms, the BO variant using trust regions performs particularly well on a large number of function, dimension, and budget combinations.

Reproducibility: Our code for reproducing the experiments is available on GitHub, in the public repository `IOH-HDBO-Comparison`.[1] The project data is also available for interactive analysis and visualization on the IOHanalyzer platform [58] as 'HDBO' dataset.

Code and full data are also stored in Zenodo under the name of `Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB`.[2] The collected data also includes results for the **Heteroscedastic and Evolutionary Bayesian Optimisation (HEBO) solver [7]**. This algorithm does not easily fit into one of the five categories proposed by [4], as it does not rely on search space reduction techniques. Instead, it employs nonlinear input and output warping to enhance the quality of the GP surrogates and multi-objective acquisition functions. In our experiments on the 24 BBOB functions, HEBO demonstrates promising performance. Nevertheless, it is excluded from our comparison due to a notable number of unsuccessful runs in the highest dimensions. The data collection also comprises results for **Ensemble Bayesian Optimization (EBO)** [59], another HDBO method tested within the 'additive models' category. However, this algorithm performed worse than **Random Decomposition Upper-Confidence Bound (RDUCB)** [64]; its results are therefore not shown in this paper. Further details can be found in A.3.

Outline of the paper: Section 2 introduces BO. Approaches to make BO efficient in high dimensions are discussed in Section 3. Following this, Section 4 presents an overview of the experimental setup employed for our comparative study. Results are presented and discussed in Sections 5 and 6. Finally, Section 7 summarizes the conclusions drawn from our study and outlines the subsequent important steps for further exploration of the current challenges. Additional details regarding the individual algorithms and an extensive analysis of the obtained results are provided in the appendix.

2 BAYESIAN OPTIMIZATION

Bayesian Optimization (BO) [16, 17, 45] is one of the most widely used surrogate-based optimization approaches. It is a global optimization strategy based on Bayesian inference, tailored to solve optimization problems with a small number of function evaluations.

BO involves two main components: a method for statistical inference, usually a GPR model, and an acquisition function that determines the optimal locations for sampling new points.

The procedure starts with a prior probability distribution for the objective function f, usually a Gaussian process prior, and a budget T for the maximum number of function evaluations. Next, the objective function f is evaluated on n_0 points that are distributed within the design space according to a Design of Experiments (DoE) scheme [15]. This process generates an initial sample set $(X, y) = \{(x^1, y^1), (x^2, y^2), ..., (x^{n_0}, y^{n_0})\}$. As long as the total budget is not exhausted, a GPR is used to construct a posterior Bayesian probability distribution describing potential values for $y = f(x)$ at a candidate point x in the design space, together with the uncertainty in the prediction. We determine the next point to evaluate by maximizing an acquisition function (e.g., expected improvement, probability of improvement, upper confidence bound, entropy search [55]) that indicates how much the evaluation of the proposed point contributes to the optimization goal, i.e., how much potential this point has to improve on the best solution found up to this iteration.

Depending on the acquisition function definition, the search strategy can show a more exploitative or more explorative attitude, favoring the investigation of areas of the search domain with low predicted mean or high predicted uncertainty, respectively.

We then observe the objective function at that point, i.e., we evaluate $y^n = f(x^n)$, add the new information (x^n, y^n) to the sample set, and update the GPR prediction to obtain a new posterior distribution for the objective function. This

[1] https://github.com/MariaLauraSantoni/IOH-Profiler-HDBO-Comparison
[2] https://zenodo.org/doi/10.5281/zenodo.8099720

Algorithm 1 Basic pseudocode for BO

Require: Objective function $f : S \to \mathbb{R}$, total budget T, number of initial points n_0, acquisition function α.

Place a Gaussian process prior on f;
Create a data set of n_0 points, $X = \{x^1, x^2, ..., x^{n_0}\}$ according to a space-filling DoE scheme;
Evaluate f on the X, let $y^i = f(x^i)$, and let $y = \{y^1, y^2, ..., y^{n_0}\}$;
Set $n = n_0$;
while $n \leq T$ **do**
    Increment n by 1;
    Train the GPR model on the sample set (X, y) to get the posterior probability distribution on f;
    Find the next point x^n to query by maximizing the acquisition function α using the current posterior distribution: $x^n = \text{argmax}\{\alpha(x) | \, x \in S\}$;
    Evaluate the function on the new point and let $y^n = f(x^n)$;
    Increase the data set by updating $(X, y) = (X, y) \cup (x^n, y^n)$;
    Update the current-best solution $x^* = \text{argmin}\left\{f(x^i) | 1 \leq i \leq n\right\}$;
end while
Return x^*.

process is iterated until a stopping criterion (e.g., exhausted evaluation budget) is met. Algorithm 1 provides a high-level overview of BO.

3 HIGH-DIMENSIONAL BAYESIAN OPTIMIZATION

Several independent studies report that *vanilla BO*, i.e., the standard version of the algorithm, works well up to around 15 decision variables [4, 62, 63]. For larger dimensions, it becomes inefficient compared to other black-box optimization solvers.

The problem of high dimensionality in BO has been addressed using several strategies that, according to [4], can be grouped into five broad categories:

- variable selection [13, 43, 53],
- additive models [8, 12, 59, 64],
- linear embeddings [6, 38, 39, 51],
- nonlinear embeddings [1, 18, 61],
- trust regions [10, 14, 52].

Researchers do not agree on which approach is best. There are pros and cons to each of them, and their performance is often influenced by the available evaluation budget and the problem structure. For example, the first two categories assume an underlying structure on the objective function, e.g., intrinsic lower dimensionality or additive structures (a decomposition of the objective function into a sum of lower-dimensional components).

In this article, we compare algorithms representing the five different approaches mentioned above. In the selection of these algorithms, preference was given to the most recent ones and those that provide a Python implementation.

For the variable selection approach, we focus on **Sparse Axis Aligned Subspace Bayesian Optimization (SAASBO)** [13]. This approach introduces a new surrogate model for high-dimensional BO based on the assumption that the coordinates of x in the design space S have a hierarchy of relevance. According to [13] this approach has several key advantages. First, it preserves the structure of the input space and can therefore exploit it. Second, it is adaptive and shows low sensitivity to its hyperparameters. Third, it can naturally accommodate both input and output constraints, unlike methods based on random projections for which input constraints are a particular challenge.

Among the algorithms that use additive models, we analyze **Random Decomposition Upper-Confidence Bound (RDUCB)** [64]. The main idea of RDUCB is to learn the decomposition of the black-box function by using a data-independent decomposition rule, a randomized tree sampling strategy. An additive acquisition function is used together with an upper confidence bound adjustment in conjunction with a message-passing optimizer. This optimizer is designed to better exploit additive acquisition structures. The algorithm successfully addresses the problem of data-driven strategies and prevents them from being easily misled into a local decomposition that does not apply globally to the entire search space.

Among the linear and nonlinear embeddings approaches, we focus on **PCA-assisted Bayesian Optimization (PCA-BO)** [51] and **Kernel PCA-assisted Bayesian Optimization (KPCA-BO)** [1], respectively.

KPCA-BO is an extended version of PCA-BO, which uses kernel methods to first map points to a reproducing kernel Hilbert space (RKHS) [3] using an implicit nonlinear feature mapping. Then, PCA in the RKHS is used to learn a linear transformation from all evaluated points during the run and select dimensions in the transformed space according to the variability of the evaluated points. The main advantage of KPCA-BO over PCA-BO is the nonlinearity of the submanifold in the search space, which makes it more likely to catch multiple basins of attraction simultaneously.

For trust-region-based approaches, we consider **Trust Region Bayesian Optimization (TuRBO)** [14]. TuRBO can be defined as a local strategy for BO. It introduces the use of trust regions in BO, making it a technique for global optimization that uses a collection of simultaneous local optimization runs with independent probabilistic models. Thompson sampling [29] is used to find new points to evaluate within a single trust region and across the set of trust regions simultaneously. The main advantage of this approach is that each local surrogate model has the typical advantages of Bayesian modeling, i.e., robustness to noisy observations and uncertainty estimates. Moreover, the local surrogates allow heterogeneous modeling of the objective function and do not suffer from overexploration. This local approach is complemented by a global bandit strategy that distributes samples across these confidence regions, implicitly balancing exploration and exploitation.

Each of the five HDBO classes has advantages and disadvantages, and the same applies to the algorithms that belong to them. Figure 1 summarizes the idea behind each approach and its pros and cons. To date, there is no consensus on which method is preferable under which circumstances (problem landscape, evaluation budget, and dimensionality).

4 EXPERIMENTAL SETUP

We compare the algorithms on the 24 noiseless functions of COCO's BBOB suite [22]. We access these functions via IOHprofiler [11]. After initial exploration of the data with IOHanalyzer, we use Python post-processing libraries to generate the plots visualizing our results in this paper.

The BBOB functions are divided into five groups: separable functions (f1-f5), functions with low or moderate conditioning (f6-f9), functions with high conditioning and unimodal structure (f10-f14), multimodal functions with appropriate global structure (f15-f19), and multimodal functions with weak global structure (f20-f24) [23]. In our experience, the problems belonging to the last two groups are most representative of real-world problems [40], as they have a more complex landscape characterized by high nonlinearity and roughness, with multiple peaks or valleys. For each function, we consider dimensions 10, 20, 40, and 60. For SAASBO we did not run complete experiments for dimensions 20, 40, and 60 due to time and memory constraints. In this case, we preferred to perform experiments on functions belonging to the last two groups (f15-f24).

Thus, we conducted 10 independent runs of each algorithm for three different instances (instance ID 0-2) of the 24 BBOB functions, with the only exception of SAASBO. For this algorithm, we have:

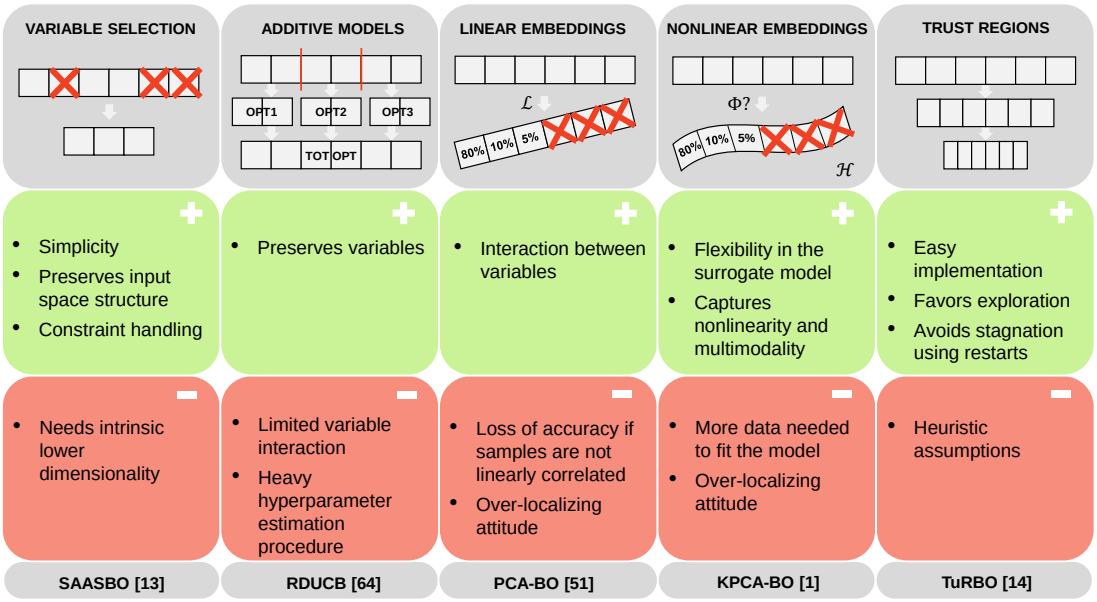


Fig. 1. Illustration of the five different categories of BO-based algorithms for high-dimensional problems. The advantages and disadvantages of each category are listed, along with the algorithm chosen for our comparison.

- data for functions f1-f24 at dimension 10,
- data for functions f15-f24 at dimensions 20 and 40,
- no data for dimension 60 (infeasible amount of computational time and memory required).

We compare the HDBO algorithms with a vanilla BO from [28], a default CMA-ES [21], and random search [27].

For each run, the total evaluation budget is set to $10D + 50$ function evaluations. For BO-based algorithms, the initial DoE size is set to D. By default for the BBOB suite, the domain is set to $[-5, 5]^D$. For each algorithm described in Section 3, we use default settings for their hyperparameters. We evaluate algorithm performance in terms of (1) loss (defined as the target precision, i.e., the absolute difference between the best-so-far value and the value of the global minimum) and (2) CPU time. We run our experiments on the LRZ Linux cluster,[3] with a 64-bit x86 CPU architecture. The cluster is equipped with Intel Xeon CPU E5-2690 v3 processors running at 2.60 GHz. It consists of 812 nodes, with each node having 28 cores and 64 GB of memory. We use source code as provided by the authors of the respective algorithms: we did not optimize code for CPU efficiency or other criteria. Although we are aware that GPU acceleration is available for certain Python package implementations, we deliberately opted against utilizing it to ensure a fair algorithm runtime comparison. This decision was also driven by our primary focus of investigating BO-based algorithms under constrained evaluation budgets – a circumstance frequently encountered in real-world applications, where high computational time is usually required for each function evaluation. Consequently, enhancing the computational efficiency of the algorithm, achievable through CUDA and GPU acceleration, was deemed secondary.

[3] https://doku.lrz.de/coolmuc-2-11484376.html

The implementation of vanilla BO is taken from the Python module `scikit-optimize`,[4] choosing Expected Improvement (EI) as the acquisition function and a noise level equal to 0.01. The implementation for CMA-ES [21] is the one in the pycma package, available from the GitHub repository `pycma`.[5] The implementation for the random search uses the Python module `numpy`[6] and in particular the method *random.uniform* that draws samples from a uniform distribution over the half-open interval that includes the lower bound, but excludes upper bound of the search space. The code for SAASBO is taken from the GitHub repository `saasbo`.[7]

For RDUCB, we use the implementation available at the GitHub repository `RDUCB`;[8] it is based on the code for HEBO [7], another HDBO algorithm mentioned in the introduction.

The code for PCA-BO and KPCA-BO is taken from the GitHub repository `Bayesian-Optimization`.[9] The TuRBO code is taken from the GitHub repository `TuRBO`[10]. In our experiments, two different implementations of TuRBO are compared: TuRBO1 and TuRBOm. They differ in the number of trust regions used by the algorithm: $tr = 1$ and $tr = \lfloor D/5 \rfloor$, respectively, where tr denotes the number of trust regions and $\lfloor \cdot \rfloor$ denotes the floor function. For more details on the individual algorithms and the values used for the hyperparameters, please refer to the appendix. The code to run the experiments is a modular framework compatible with IOHprofiler. It is available on GitHub in the repository `IOH-HDBO-Comparison`.[11]

We use Python post-processing libraries to present our results and the statistical Wilcoxon signed-rank test to confirm what can be seen through direct inspection of the plots.

To better assess the performance of our optimization algorithms, we also present empirical cumulative distribution function (ECDF) curves in Appendix D.

5 RESULTS

5.1 Dimension D = 10

5.1.1 Solution Quality. Figure 2 compares the convergence behavior of all algorithms on all BBOB functions from f1 to f24 in dimension 10.

In general, random search performed poorly compared to the other algorithms, with only a few exceptions: PCA-BO and KPCA-BO are often comparable or even worse than random search (f1, f3, f4, f6-f8, f10-f12), the same is true for RDUCB (f21, f22) and a few times for vanilla BO, especially for large budgets (f2, f4, f12, f16, f17). This becomes less and less visible with increasing dimensionality, since the purpose of these algorithms is to perform well in high-dimensions. For a small evaluation budget, vanilla BO always performs better than CMA-ES, as we can see in particular on f2-f4, f12, and f19. However, at the end of the budget, CMA-ES outperforms or is comparable to vanilla BO in many cases. Overall, we see a good performance of vanilla BO, which is due to the still low dimensionality. BO is always among the best or at least comparable to the other algorithms, with a few exceptions. It reaches a particularly good performance on f5, f6, f22, and f24.

In Figure 2, algorithm performance is not stable across functions. However, SAASBO and TuRBO tend to predominate. Specifically, TuRBO finds excellent loss values on f2, f3, f13, f14, f16, f18, f20, and f22 (here together with SAASBO).

[4] https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html
[5] https://github.com/CMA-ES/pycma
[6] https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html
[7] https://github.com/martinjankowiak/saasbo
[8] https://github.com/huawei-noah/HEBO/tree/master/RDUCB
[9] https://github.com/wangronin/Bayesian-Optimization/tree/KPCA-BO
[10] https://github.com/uber-research/TuRBO
[11] https://github.com/MariaLauraSantoni/IOH-Profiler-HDBO-Comparison

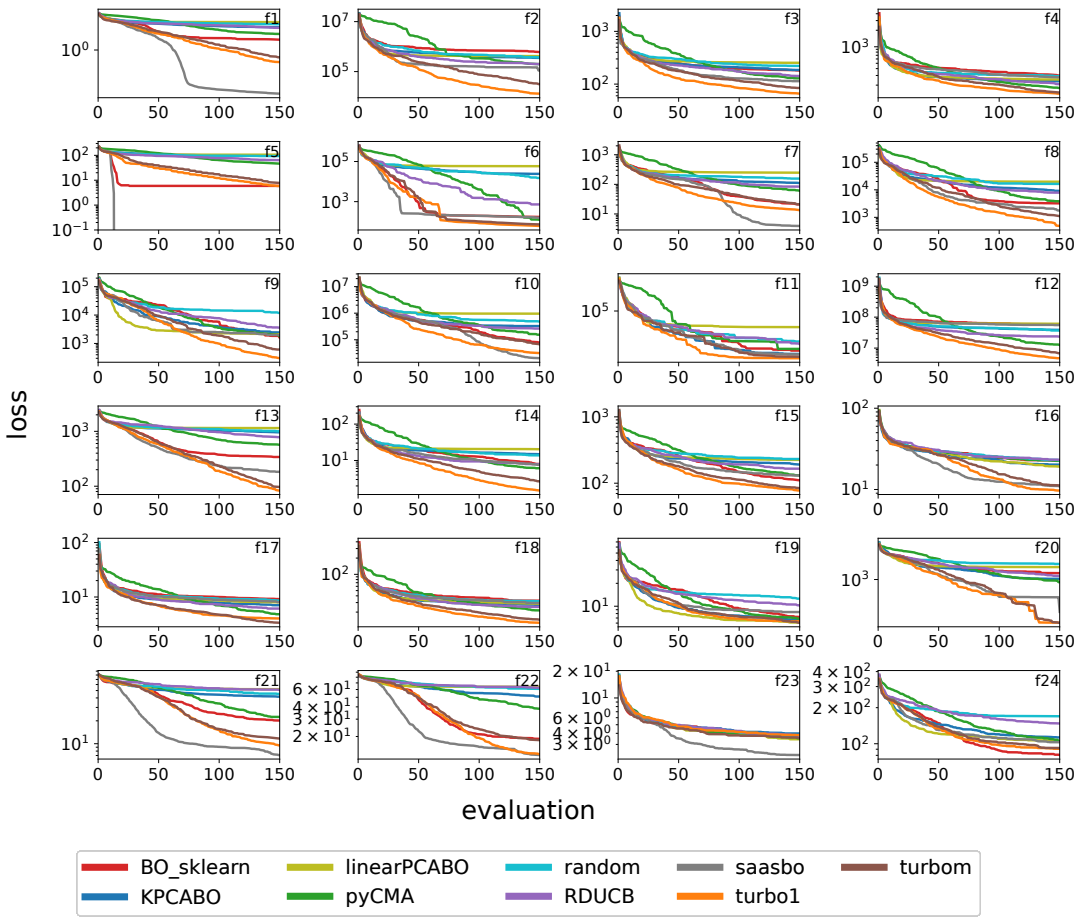


Fig. 2. The best-so-far target gap for dimension 10.

Stagnation at a very low budget is a common behavior of PCA-BO, KPCA-BO (f1, f5, f6, and f20-f22) and RDUCB (f1, f13, f16, f21, f22). If we focus on f5 (linear slope), we can observe the extremely good performance of BO and SAASBO, which are able to immediately find the global optimum due to the unimodal and monotonic landscape of this function (this observation holds for all tested dimensions). Finally, we note that sometimes random search outperforms some of the HDBO algorithms (PCA-BO, KPCA-BO, and RDUCB). However, this behavior is not consistent across problems and vanishes in higher dimensionalities, for which HDBO algorithms are specifically designed. The above observations are confirmed by the ECDF curves in the appendix.

5.1.2 CPU time. In Figure 3, we compare the CPU time in seconds (logarithmic scale) to complete the entire run and the CPU time taken by the different algorithms to fit the GPR model and optimize the acquisition function. We also show the bootstrap confidence interval by the black line in each bar [9]. We can clearly see that SAASBO is the most expensive strategy in terms of total CPU time with an average of 5 284.51 seconds. TuRBO1 is the fastest, followed by RDUCB, TuRBOm, and PCA-BO (close to vanilla BO). According to the right side of the plot in Figure 3, SAASBO is

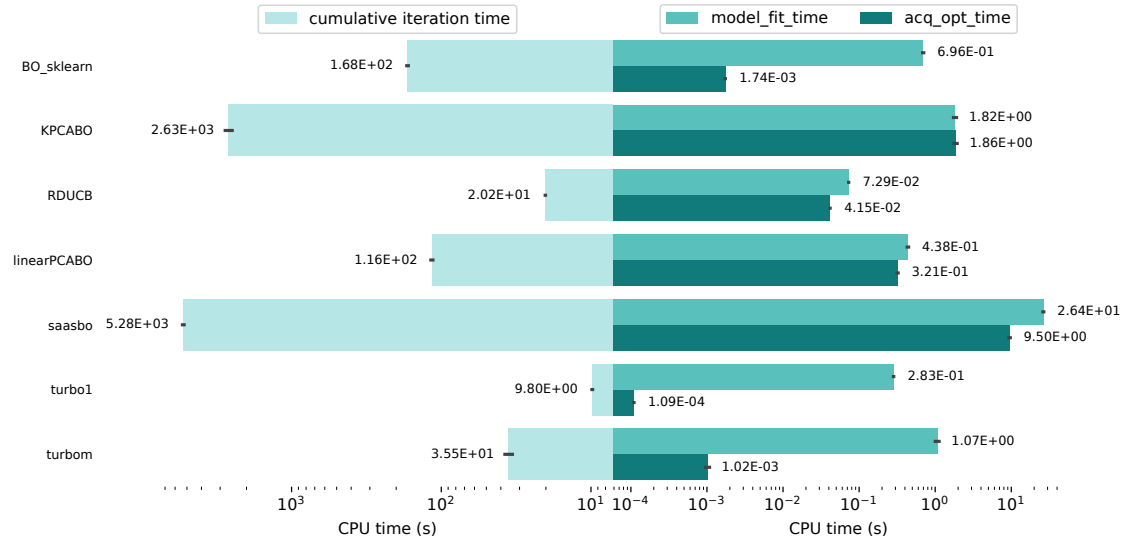


Fig. 3. CPU time in seconds (logarithmic scale) for the entire run (left) and model fitting and acquisition function optimization (right) in dimension 10. Values for the total CPU time are averaged across all 24 BBOB functions. Model fitting time and acquisition function optimization time are first averaged over all iterations of one run, and then across the 24 BBOB functions. The black line in each bar represents the bootstrap confidence interval.

also the one that takes more time to complete both steps, fitting the model and optimizing the acquisition function. For almost all algorithms, the CPU time for optimizing the acquisition function is shorter than that for building the model. For PCA-BO and KPCA-BO, they are comparable. From the analyses of convergence and CPU time at 10D, TuRBO1 and TuRBOm seem to perform best.

5.2 Dimension D = 20

5.2.1 Solution Quality. Figure 4 compares the convergence behavior of the algorithms on the 20-dimensional problems. We recall that for SAASBO we only have results for f15-f24, due to its high running time. With few exceptions, random search performs worse than all tested algorithms. Only for f21 and f22 its performance is comparable to RDUCB, the latter showing stagnant behavior. The overall behavior of the BO algorithms and CMA-ES is very similar to that observed for dimension 10. In particular, we see that vanilla BO always outperforms CMA-ES for small budgets, but CMA-ES catches up for a larger budget of 100 evaluations in many cases (f2-f4, f9, f11, f12, and f14). At a high level, vanilla BO is either better or at least comparable to CMA-ES for the entire budget for all multimodal BBOB functions (f15-f24). We can attribute this to the CMA-ES overlocalizing search attitude. It is also interesting to note that at dimension 20, vanilla BO still shows very good performance for some of the functions compared to the BO variants that are specifically designed for high-dimensional problems. In particular, it has one of the best average performances among all algorithms on function f8 and the best one on functions f5 and f24. SAASBO and TuRBO1 are the two HDBO variants whose average performance is best among all tested algorithms for the largest number of functions. Their competitive advantage over the other algorithms is clearly visible in the convergence plots in Figure 4, in particular, for f1, f2, f13, and f20-f23. From this dimension on, we can observe a jump in the convergence behavior of RDUCB that we could not observe for dimension 10 due to the small budget. This jump always occurs around budget 200, often

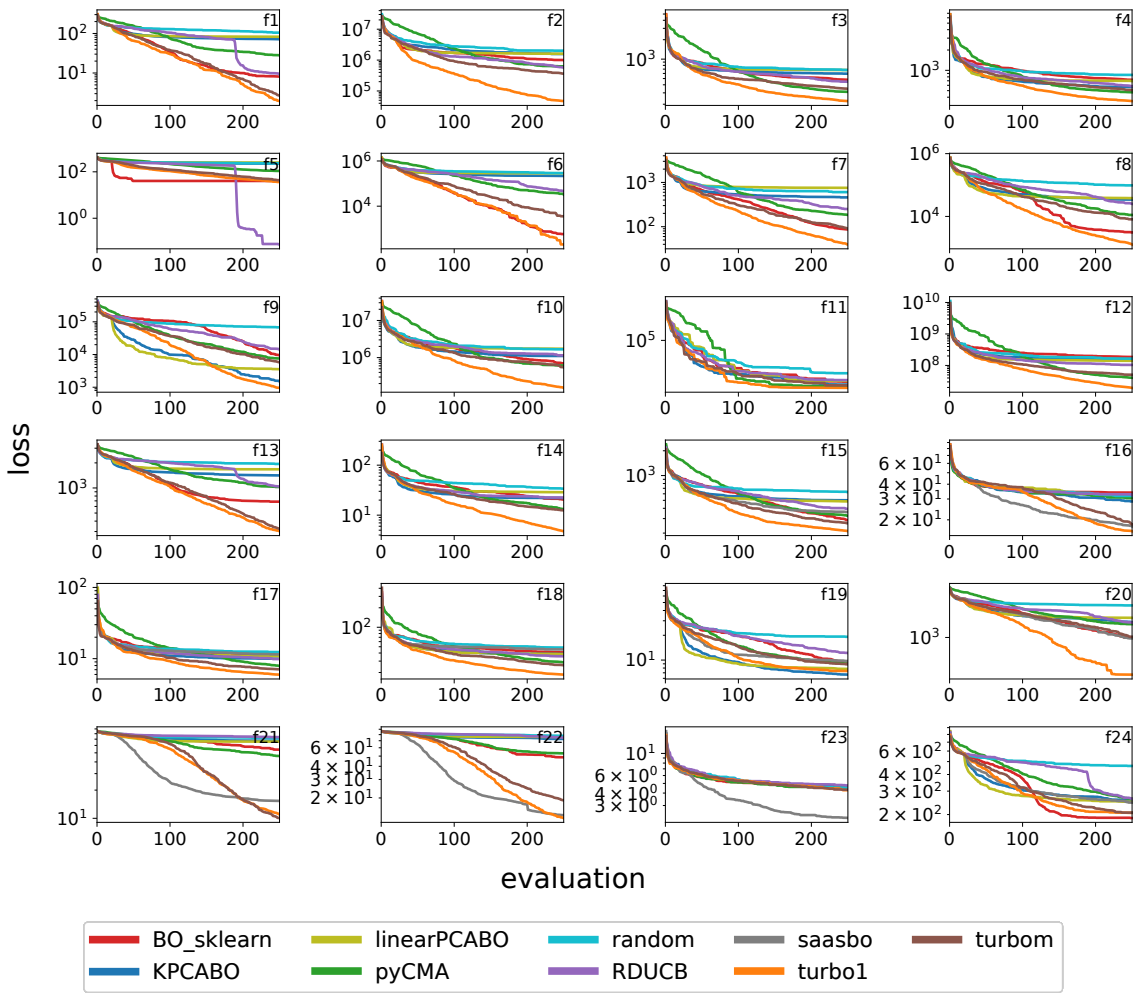


Fig. 4. The best-so-far target gap for dimension 20.

after several evaluations showing a plateau, and it is particularly evident for some functions (f1, f5, f13, f24). For this reason, RDUCB is the best solution for f5 in this dimension. This behavior can also be observed for the plots shown in the original work [64].[12] Nevertheless, RDUCB still shows stagnation on some problems (f16, f21, f22). The above observations are confirmed by the ECDF curves in the appendix also for dimension 20. Of particular interest is the still good performance of vanilla BO, which continues to show good results, as the dimension is not yet too high for the curse of dimensionality to take effect.

[12]In recent, private communication, the authors attribute this behavior to the *GPy* package used in the original implementation of the algorithm. This package has been replaced by *GPyTorch* in an alternative implementation available at https://github.com/huawei-noah/HEBO/tree/master/MCBO. However, we have not tested this new implementation and cannot confirm whether it resolves the jumps.

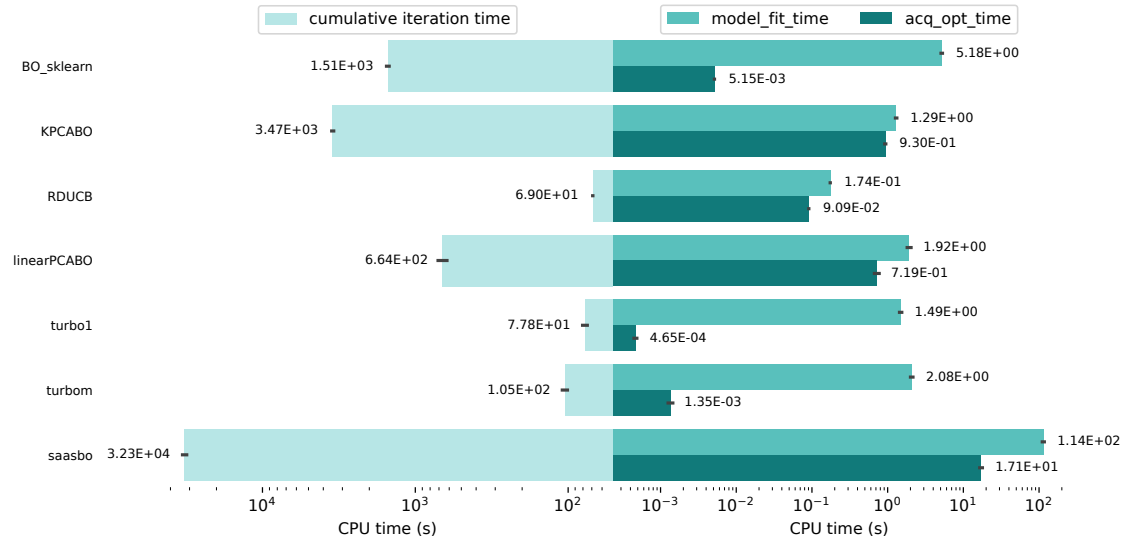


Fig. 5. CPU time in seconds (logarithmic scale) for the entire run (left) and model fitting and acquisition function optimization (right) in dimension 20. Values for the total CPU time are averaged across all 24 BBOB functions. Model fitting time and acquisition function optimization time are first averaged over all iterations of one run, and then across the 24 BBOB functions. The black line in each bar represents the bootstrap confidence interval.

5.2.2 CPU time. Figure 5 compares the CPU times for the entire run, for the modeling phase, and for the optimization of the acquisition function for each algorithm tested. As the figure shows, the time increases significantly compared to dimension 10. For this dimension, the fastest algorithm in terms of cumulative iteration CPU time is RDUCB, followed by TuRBO1 and TuRBOm. Considering the two different analyses, the experimental results suggest that the best algorithm in dimension 20 is TuRBO1. It seems particularly well suited for f20-f22, which belongs to the category of multimodal functions with weak global structure.

5.3 Dimension D = 40

5.3.1 Solution Quality. Figure 6 shows the convergence plots for dimension 40, where the difference in performance between random search and the other algorithms becomes more visible. We start seeing that vanilla BO scales badly with increasing dimensionality. For a low budget, of $5D$, i.e., 200 function evaluations, BO is never competitive, with the only exception of f5. The best algorithms remain SAASBO and TuRBO1 for most functions. However, we can also notice that on f9 and f19, the KPCA-BO algorithm, followed by PCA-BO, performs significantly better than all the other methods for the entire evaluation budget. We attribute this to the very good global structure of the function landscapes. In this case, PCA-BO and KPCA-BO are more capable of properly capturing the isocontour of the objective function and it is more likely that basins of attraction are detected. The ECDF curves in the appendix confirm all previous observations and in particular the effect of the curse of dimensionality for vanilla BO.

5.3.2 CPU time. In Figure 7, the three different CPU times of the algorithms confirm a prohibitive CPU time for SAASBO, while the computational efficiency of TuRBO1 compared to the other algorithms is even more evident than it

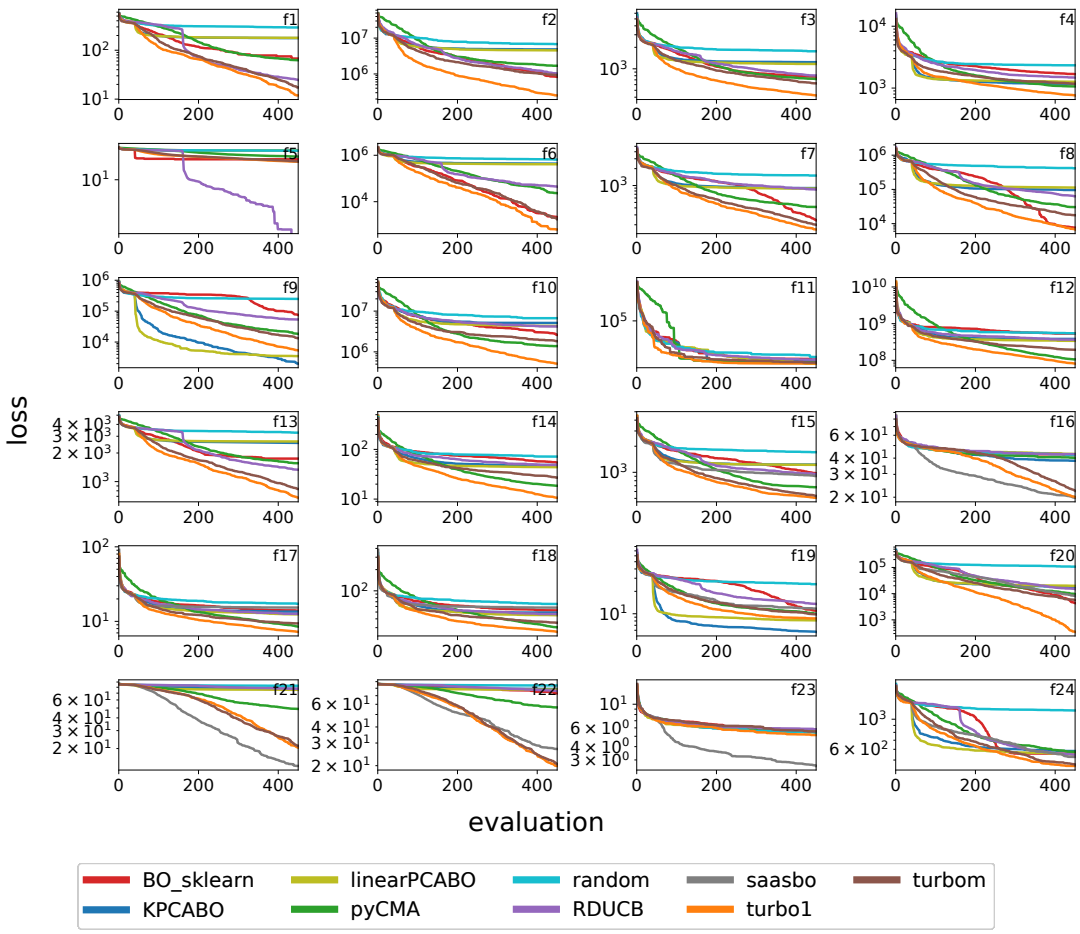


Fig. 6. The best-so-far target gap for dimension 40.

was for lower dimension. TuRBO is efficient in terms of CPU time because the Thompson Sampling approach substitutes the sub-optimization of a traditional acquisition function. RDUCB still maintains a good CPU performance, following directly TuRBO1, and outperforming TuRBOm.

Considering the two different analyses, for dimension 40 we can firmly support the predominance of TuRBO1 over the other algorithms.

5.4 Dimension D = 60

5.4.1 Solution Quality. Figure 8 compares the convergence behavior of the algorithms for the 60-dimensional BBOB problems. Due to computational constraints, SAASBO is completely missing, as explained in Section 4. In dimension 60, the inefficiency of random search is obvious, leaving a few exceptions where vanilla BO stagnates and is comparable to it (f9, f12) or where all algorithms are comparable (f11, f16, f21-f23), except for some outliers, such as TuRBO1 and TuRBOm on f16, or also CMA-ES on f21 and f22. The figure demonstrates how BO suffers from a lower convergence

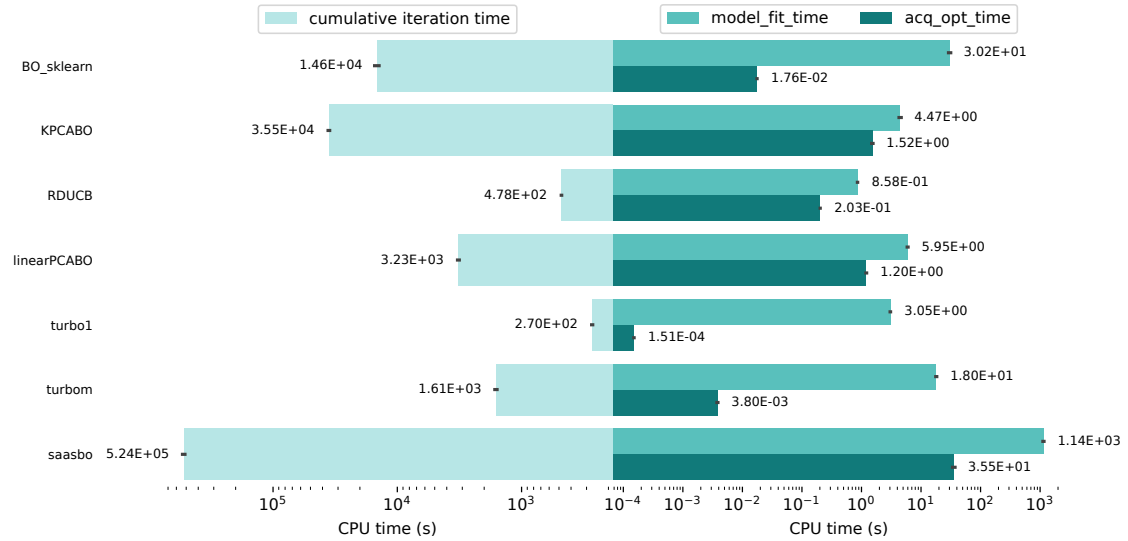


Fig. 7. CPU time in seconds (logarithmic scale) for the entire run (left) and model fitting and acquisition function optimization (right) in dimension 40. Values for the total CPU time are averaged across all 24 BBOB functions. Model fitting time and acquisition function optimization time are first averaged over all iterations of one run, and then across the 24 BBOB functions. The black line in each bar represents the bootstrap confidence interval.

rate at high dimensionality. In all cases except for f5, better convergence capabilities of CMA-ES are evident. BO suffers from premature stagnation, which is due to its higher computational complexity in dimension 60.

We can observe the same behavior for some HDBO methods, such as RDUCB and, in some cases, linear and kernel PCA-BO. We attribute this to the choice of their hyperparameters as the default ones, which might not be ideal for the function landscapes addressed in this study. The performance of BO decreases even on f24, where it was the best solver at lower dimensions. Both versions of TuRBO show very good performance for f1, f13, f21, and f22, and there is a statistically significant difference between them and the other algorithms, which we confirmed by running a Wilcoxon signed-rank test. Moreover, it is worth noting that TuRBO is the only algorithm that continues to improve as the number of evaluations increases, while the other algorithms stagnate easily. Nonetheless, we can observe interesting performance of PCA-BO and KPCA-BO in some cases. They find excellent loss values on f9, f11, f18-f20, and f24, where they either rank first or show initial speedups that make them good candidates for high-dimension/low-budget optimization problems.

Also in dimension 60, the ECDF curves in the appendix confirm all previous observations. The results highlight the strong impact of high dimensionality on BO, the distinctive and advantageous characteristic of fast convergence for very small evaluation budgets observed for both KPCA-BO and PCA-BO, and the supremacy of TuRBO1 and TuRBOm, which are the only algorithms beating CMA-ES in a fixed-target analysis considering the full evaluation budget.

5.4.2 CPU time. In Figure 9, on the left, we can see that KPCA-BO and vanilla BO are the algorithms that require the highest CPU time for the run. Again, TuRBO1 is significantly better than the other algorithms and the difference in performance between TuRBO1 and TuRBOm becomes clearer, which suggests the use of just one trust region for high-dimensional problems. Once again, we observe RDUCB positioned between TuRBO1 and TuRBOm. On the

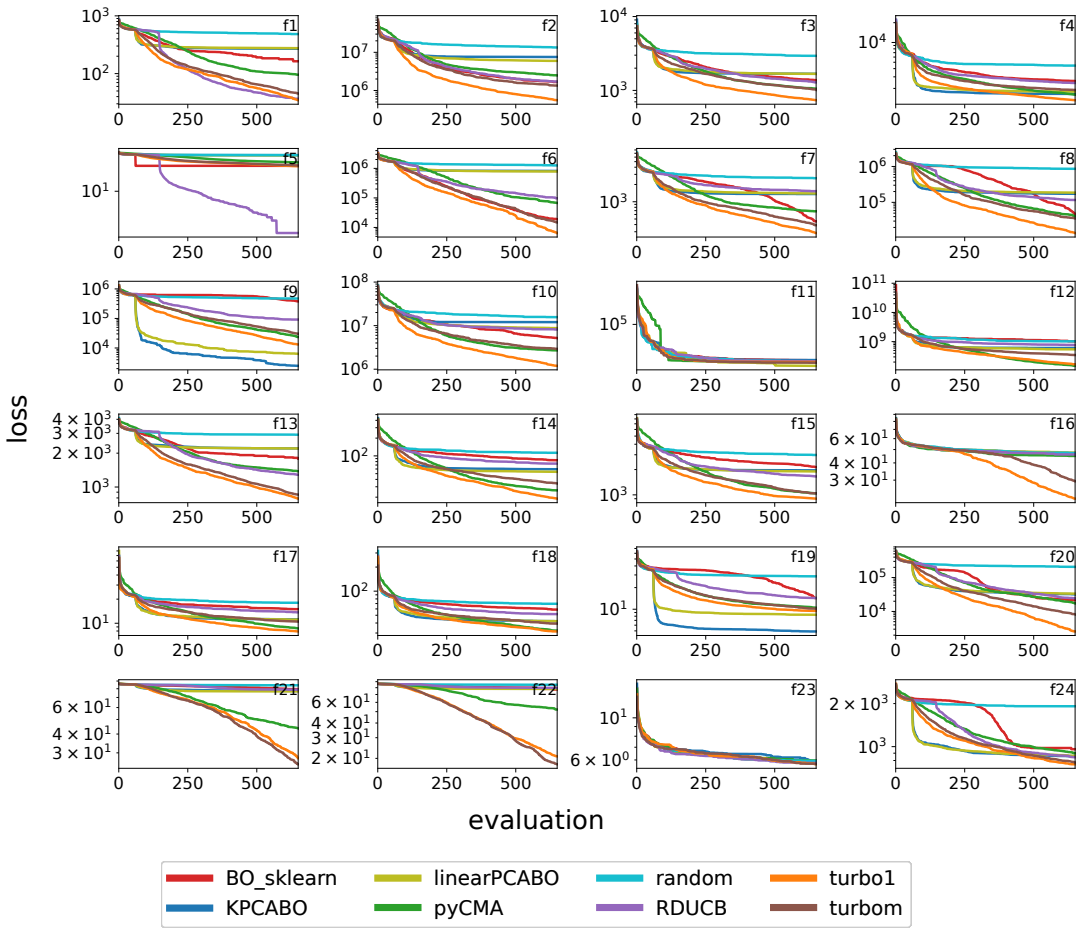


Fig. 8. Best-so-far target gap for dimension 60.

right-hand side, Figure 9 confirms that the CPU time for optimizing the acquisition function is generally much shorter than the time required for fitting the model, with almost comparable values for linear and kernel PCA-BO, given that both steps are performed in a mapped space with reduced dimensionality. Finally, we note that the time taken to optimize the acquisition function varies widely between algorithms. This shows that great efforts have been made to improve the efficiency of the infill criterion, which is a main differentiating trait between the algorithms studied.

6 FURTHER DISCUSSION

For an in-depth comparison, we also present in Figure 10 violin plots that compare the performance of three candidate algorithms for each BBOB function for dimension 40: vanilla BO, CMA-ES, and the best among the HDBO algorithms for a specific function at the end of the budget (see the appendix for similar violin plots for dimensions 10, 20, and 60). While BO is usually outperformed by CMA-ES at the end of the evaluation budget, the best among the HDBO algorithms always performs better than CMA-ES.

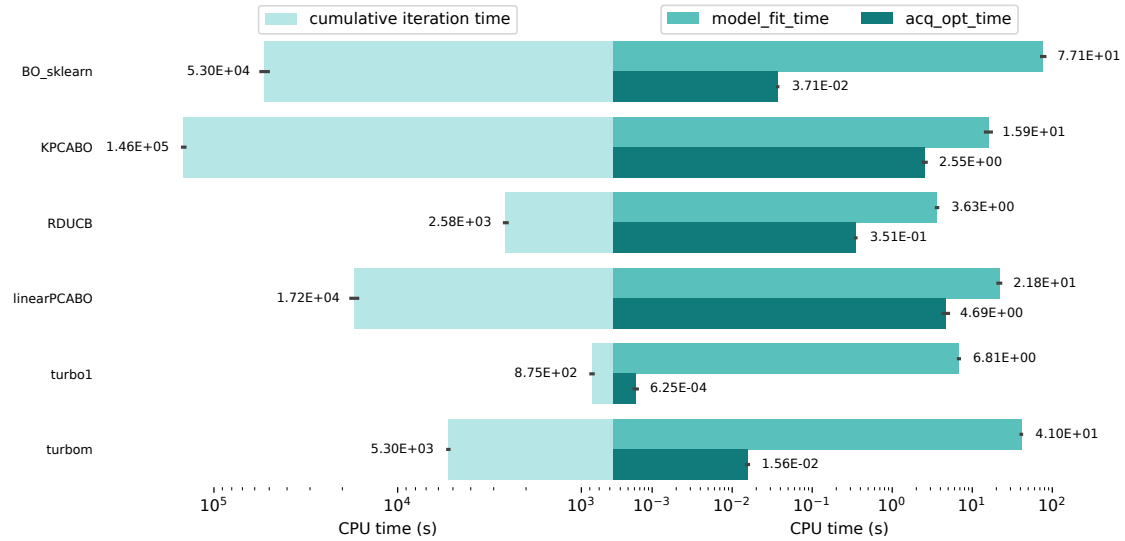


Fig. 9. CPU time in seconds (logarithmic scale) for the entire run (left) and model fitting and acquisition function optimization (right) in dimension 60. Values for the total CPU time are averaged across all 24 BBOB functions. Model fitting time and acquisition function optimization time are first averaged over all iterations of one run, and then across the 24 BBOB functions. The black line in each bar represents the bootstrap confidence interval.

In general, we can see that TuRBO1 performs the best in dimension 40. However, there are some exceptions: KPCA-BO on f9 and f19, PCA-BO on f11, RDUCB on f5, and SAASBO on f21 and f23. The alternative algorithms outperform TuRBO in these functions. This can be attributed to the intrinsic properties of the specific functions, whose landscapes can be better searched through other inner mechanisms than trust regions. For example, some functions have a hierarchical structure in the variables, or they provide a reliable subspace that is well suited for dimensionality reduction by embeddings. In addition, certain algorithms, such as RDUCB or SAASBO (Fig 2), prove to be more effective in dealing with boundary problems. This is particularly evident for f5, where the global optimum lies at the edge of the search space. Therefore, TuRBO demonstrates strong performance across a diverse set of problems in our comparison. However, when specific assumptions about function properties are met, alternative algorithms might prove to be more effective. This emphasizes the importance of choosing the most appropriate algorithm tailored to the specific characteristics of the optimization problem landscape.

Figure 11 shows the convergence evolution of the algorithms compared on f24 at dimension 60, by freezing it at three different budgets: 200, 400, and 600 evaluations. The figure gives an idea of the ranking of the algorithms in different phases of the optimization runs and clearly shows in which context one of the algorithms is preferable to the others. For a limited budget (Figure 11, budget 200), PCA-BO and KPCA-BO find the lowest loss values. We attribute this to their better ability to find good solutions in a lower-dimensional manifold of the original search space. However, this often drives the search towards local optima. Already at budget 400 (Figure 11, budget 400), PCA-BO and KPCA-BO become comparable to the other HDBO algorithms and they are significantly outperformed at the end of the run (Figure 11, budget 600). At this point, TuRBO1 and TuRBOm represent the best choice. In fact, TuRBO seems to offer a better balance between exploration and exploitation of the domain. This is due to the use of dynamic trust regions and,

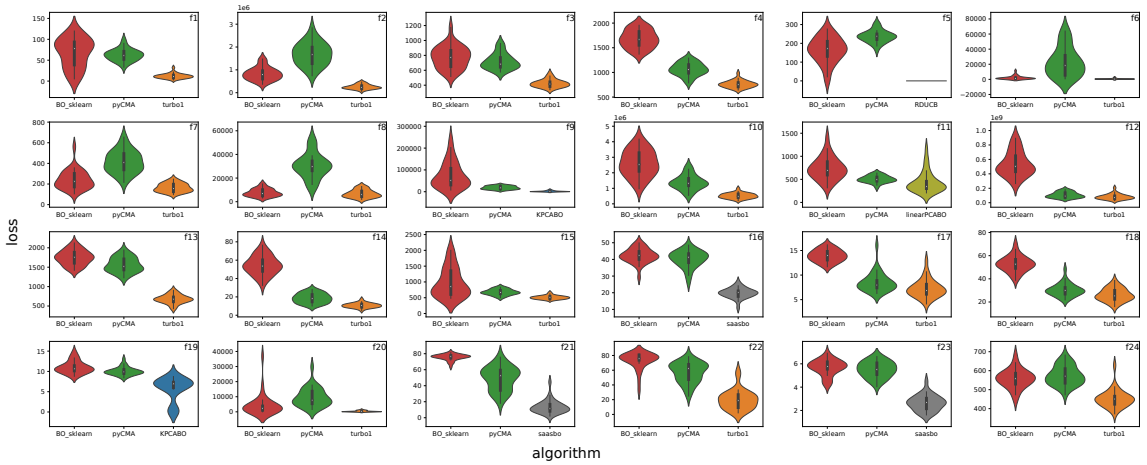


Fig. 10. The violin plots show, for dimension 40 and budget 450 (final budget), the distribution of the best-so-far loss values for all functions, obtained by vanilla BO, CMA-ES, and the best among the HDBO algorithms. The plots include a marker for the median of the data and a box indicating the interquartile range.

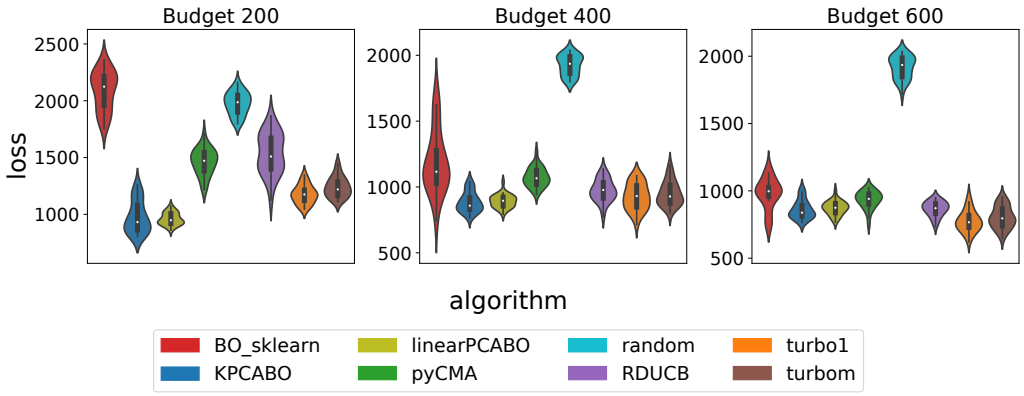


Fig. 11. Violin plots showing loss values at three different budgets (200, 400, and 600 function evaluations) for function 24 in dimension 60.

in particular, the strategic integration of multiple restarts. The use of random restarts in TuRBO proves particularly beneficial as it allows local optima to be effectively abandoned and exploration to begin anew, increasing its adaptability to a variety of function landscapes. Based on a Wilcoxon signed-rank test, these results are statistically significant. Given these observations, we believe that it would be interesting to explore the possibility of combining the concepts behind PCA-BO and TuRBO, by building local low-dimensional embeddings. In this way, one can benefit from both the flexibility of the trust regions and the lower complexity of the manifolds with reduced dimension. We leave this to our future research.

Finally, although there is no consistent behavior indicating that one algorithm outperforms other algorithms across different BBOB classes, we found that some inner mechanisms are better suited for certain landscape characteristics

than others. The main findings are the following: TuRBO performs remarkably well on unimodal landscapes, where its trust regions effectively evolve towards the global basin of attraction (e.g. f2, f3, f13, f14). This trend also extends to multimodal functions with a global unimodal structure and added noise (e.g., f15, f18, f20). However, in the case of multimodal functions with numerous basins of attractions distributed across the design space, a trust region approach, initially guided by a generated sample, runs the risk of neglecting a global perspective and overlooking significant basins of attraction. In such scenarios, SAASBO is a better choice and outperforms other algorithms (e.g. f16, f21-f23). Another interesting case is shown where PCA-BO and KPCA-BO, which use low-dimensional embeddings, show superiority. These algorithms usually perform well in landscapes where the dominant dimension of the basins of attraction easily aligns with the origin of the space (e.g., f9, f19, f24), yet they demonstrate limited generalization capabilities across various functions and instances.

7 CONCLUSION AND FUTURE PERSPECTIVES

In this work, we have conducted an experimental study comparing the performance of vanilla BO, CMA-ES, random search, and five BO-based algorithms designed to improve BO performance in high-dimensional search spaces. For our tests, we selected one representative for each of the main categories of algorithms for high-dimensional BO: SAASBO for variable selection, RDUCB for additive models, PCA-BO and KPCA-BO for linear and nonlinear embeddings, respectively, and TuRBO for trust regions. We compared the algorithms on the 24 functions from the open-source BBOB benchmark suite from the COCO benchmarking environment, by running 10 repetitions of each algorithm on 3 different instances of each function, with some exceptions due to time and memory constraints. With our convergence and CPU time analyses, we provided unbiased and reproducible results for both researchers in the field and practitioners. Researchers can gain insights for algorithm design, while practitioners can leverage our findings for informed algorithm selection in real-world problem-solving. For those looking for a quick overview of HDBO algorithms, our work is a valuable resource that provides a concise and informative introduction to the topic.

Our results confirm a good performance of BO at low dimension (10D), which gradually deteriorates as the dimensionality of the problem increases. Here, CMA-ES performs better, especially for larger budgets. However, the average observed performance of CMA-ES is worse than that of the HDBO algorithms. Although we observe different performances for different function landscapes and budget utilization phases, TuRBO seems to be the most promising algorithm, both in terms of convergence trend and CPU time. However, PCA-BO and KPCA-BO also show potential for small evaluation budgets, with very fast convergence towards a near-optimal solution. Therefore, further work is planned to develop a hybrid algorithm that combines PCA-BO and TuRBO. This algorithm could avoid the stagnation of PCA-BO by using restarts and trust regions and still benefit from a linear low-dimensional embedding, resulting in a very competitive algorithm for optimizing expensive black-box functions at high dimensionality. Indeed, the high potential resulting from the combination of linear embeddings and trust regions has already been demonstrated through the introduction of the Bayesian optimization with adaptively expanding subspaces (BAxUS) algorithm [48]. This algorithm will serve as a valuable baseline for comparison purposes.

We also found that the poor performance of some algorithms on some functions could be due to poor initialization of their hyperparameters. Hence, we plan to investigate how they could benefit from hyperparameter optimization and repeat our comparative analysis.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of the manuscript and the reproducibility reviewer for their constructive comments which helped us improve the choice of algorithms to discuss and the ease by which our experiments can be reproduced. We also thank Diederick Vermetten from the University of Leiden for making our data available through the IOHanalyzer website https://iohanalyzer.liacs.nl/.

This work was realized with the support of the Sorbonne Center for Artificial Intelligence (SCAI) of Sorbonne University (IDEX SUPER 11-IDEX-0004), of the ANR T-ERC project *VARIATION* (ANR-22-ERCS-0003-01), and of CNRS INS2I project *IOHprofiler*. The work leading to this publication was also supported by the PRIME programme of the German Academic Exchange Service (DAAD) with funds from the German Federal Ministry of Education and Research (BMBF). The contribution of Renato De Leone has been funded by the European Union - NextGenerationEU under the Italian Ministry of University and Research (MUR) National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP J13C22000430001. This study was also carried out under the project INdAM – GNCS Project 2023, codice CUP_E53C22001930001. The manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

[1] Kirill Antonov, Elena Raponi, Hao Wang, and Carola Doerr. 2022. High Dimensional Bayesian Optimization with Kernel Principal Component Analysis. In *Proc. of Parallel Problem Solving from Nature (PPSN)*, Vol. 13398. Springer, 118–131. https://doi.org/10.1007/978-3-031-14714-2_9

[2] R. Bellman. 1966. Dynamic Programming. *Science (New York, N.Y.)* 153, 3731 (July 1966), 34–37. https://doi.org/10.1126/science.153.3731.34

[3] Alain Berlinet and Christine Thomas-Agnan. 2011. *Reproducing kernel Hilbert spaces in probability and statistics.* Springer Science & Business Media.

[4] Mickaël Binois and Nathan Wycoff. 2022. A Survey on High-dimensional Gaussian Process Modeling with Application to Bayesian Optimization. *ACM Trans. Evol. Learn. Optim.* 2, 2 (2022), 8:1–8:26. https://doi.org/10.1145/3545611

[5] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. 2016. Bayesian Optimization for Learning Gaits under Uncertainty. *Annals of Mathematics and Artificial Intelligence* 76, 1 (Feb. 2016), 5–23. https://doi.org/10.1007/s10472-015-9463-9

[6] Jingfan Chen, Guanghui Zhu, Rong Gu, Chunfeng Yuan, and Yihua Huang. 2020. Semi-supervised Embedding Learning for High-dimensional Bayesian Optimization. *CoRR* abs/2005.14601 (2020). arXiv:2005.14601 https://arxiv.org/abs/2005.14601

[7] Alexander I. Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Hao Jianye, Jun Wang, Jan Peters, and Haitham Bou Ammar. 2022. HEBO Pushing The Limits of Sample-Efficient Hyperparameter Optimisation. arXiv:2012.03826 [cs.LG]

[8] Ian Delbridge, David Bindel, and Andrew Gordon Wilson. 2020. Randomly projected additive Gaussian processes for regression. In *International Conference on Machine Learning*. PMLR, 2453–2463.

[9] Thomas J DiCiccio and Bradley Efron. 1996. Bootstrap confidence intervals. *Statistical science* 11, 3 (1996), 189–228.

[10] Youssef Diouane, Victor Picheny, Rodolophe Le Riche, and Alexandre Scotto Di Perrotolo. 2023. TREGO: a trust-region framework for efficient global optimization. *Journal of Global Optimization* 86, 1 (2023), 1–23.

[11] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2018. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv preprint arXiv:1810.05281* (2018). https://iohprofiler.github.io/.

[12] Nicolas Durrande, David Ginsbourger, and Olivier Roustant. 2011. Additive kernels for Gaussian process modeling. *arXiv preprint arXiv:1103.4023* (2011).

[13] David Eriksson and Martin Jankowiak. 2021. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Uncertainty in Artificial Intelligence*. PMLR, 493–503.

[14] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. 2019. Scalable global optimization via local Bayesian Optimization. *Advances in Neural Information Processing Systems* 32 (2019).

[15] Alexander I. J. Forrester, András Sóbester, and Andy J. Keane. 2008. *Engineering Design via Surrogate Modelling - A Practical Guide.* John Wiley & Sons Ltd.

[16] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[17] Roman Garnett. 2023. *Bayesian Optimization.* Cambridge University Press.

[18] David Gaudrie, Rodolphe Le Riche, Victor Picheny, Benoit Enaux, and Vincent Herbert. 2020. Modeling and optimization with Gaussian processes in reduced eigenbases. *Structural and Multidisciplinary Optimization* 61, 6 (2020), 2343–2361.

[19] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. 2020. Constrained Bayesian Optimization for Automatic Chemical Design Using Variational Autoencoders. *Chemical Science* 11, 2 (Jan. 2020), 577–586. https://doi.org/10.1039/C9SC04026A

[20] Nikolaus Hansen. 2006. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation* (2006), 75–102.

[21] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. CMA-ES/pycma on Github. Zenodo, DOI: 10.5281/zenodo. 2559634.(Feb. 2019).

[22] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. 2021. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* 36 (2021), 114–144. Issue 1. https://doi.org/10.1080/10556788.2020.1808977

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

[24] Nikolaus Hansen and Stefan Kern. 2004. Evaluating the CMA evolution strategy on multimodal test functions. In *International conference on parallel problem solving from nature*. Springer, 282–291.

[25] Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11, 1 (2003), 1–18.

[26] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[27] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[28] Tim Head, M Kumar, H Nahrstaedt, G Louppe, and I Shcherbatyi. 2017. scikit-optimize: Sequential model-based optimization in Python.

[29] José Miguel Hernández-Lobato, Edward Pyzer-Knapp, Alan Aspuru-Guzik, and Ryan P Adams. 2016. Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space. In *NIPS Workshop on Bayesian Optimization*.

[30] José Miguel Hernández-Lobato, James Requeima, Edward O. Pyzer-Knapp, and Alán Aspuru-Guzik. 2017. Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 1470–1479.

[31] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2013. An Evaluation of Sequential Model-Based Optimization for Expensive Blackbox Functions. In *Proc. GECCO (Companion)*. ACM, 1209–1216. https://doi.org/10.1145/2464576.2501592

[32] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'05)*. Springer-Verlag, Berlin, Heidelberg, 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[33] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. https://doi.org/10.48550/arXiv.1806.10282 arXiv:1806.10282 [cs, stat]

[34] Kai Junge, Josie Hughes, Thomas George Thuruthel, and Fumiya Iida. 2020. Improving Robotic Cooking Using Batch Bayesian Optimization. *IEEE Robotics and Automation Letters* 5, 2 (April 2020), 760–765. https://doi.org/10.1109/LRA.2020.2965418

[35] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. https://doi.org/10.48550/arXiv.1605.07079 arXiv:1605.07079 [cs, stat]

[36] Lars Kotthoff, Hud Wahab, and Patrick Johnson. 2021. Bayesian Optimization in Materials Science: A Survey. *arXiv:2108.00002 [cond-mat, physics:physics]* (July 2021). arXiv:2108.00002 [cond-mat, physics:physics]

[37] Rémi Lam, Matthias Poloczek, Peter Frazier, and Karen E. Willcox. 2018. Advances in Bayesian Optimization with Applications in Aerospace Engineering. In *2018 AIAA Non-Deterministic Approaches Conference*. American Institute of Aeronautics and Astronautics, Kissimmee, Florida. https://doi.org/10.2514/6.2018-1656

[38] Ben Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. 2020. Re-examining linear embeddings for high-dimensional bayesian optimization. *Advances in neural information processing systems* 33 (2020), 1546–1558.

[39] Benjamin Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. 2020. Re-Examining Linear Embeddings for High-Dimensional Bayesian Optimization. arXiv:2001.11659 [stat.ML]

[40] Fu Xing Long, Bas van Stein, Moritz Frenzel, Peter Krause, Markus Gitterle, and Thomas Bäck. 2022. Learning the Characteristics of Engineering Optimization Problems with Applications in Automotive Crash. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 1227–1236. https://doi.org/10.1145/3512290.3528712

[41] Gustavo Malkomes, Charles Schaff, and Roman Garnett. 2016. Bayesian Optimization for Automated Model Selection. In *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc.

[42] Mohit Malu, Gautam Dasarathy, and Andreas Spanias. 2021. Bayesian Optimization in High-Dimensional Spaces: A Brief Survey. In *International Conference on Information, Intelligence, Systems & Applications (IISA)*. IEEE, 1–8. https://doi.org/10.1109/IISA52424.2021.9555522

[43] Amandine Marrel, Bertrand Iooss, François Van Dorpe, and Elena Volkova. 2008. An efficient methodology for modeling complex computer codes with Gaussian processes. *Computational Statistics & Data Analysis* 52, 10 (2008), 4731–4744.

[44] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José Castellanos, and Arnaud Doucet. 2009. A Bayesian Exploration-Exploitation Approach for Optimal Online Sensing and Planning with a Visually Guided Mobile Robot. *Autonomous Robots* 27, 2 (Aug. 2009), 93–103. https://doi.org/10.1007/s10514-009-9130-2

[45] Jonas Mockus. 2012. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media.

[46] Vu Nguyen. 2019. Bayesian Optimization for Accelerating Hyper-Parameter Tuning. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. 302–305. https://doi.org/10.1109/AIKE.2019.00060

[47] Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.

[48] Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. 2022. Increasing the scope as you learn: Adaptive Bayesian optimization in nested subspaces. *Advances in Neural Information Processing Systems* 35 (2022), 11586–11601.

[49] Elena Raponi, Mariusz Bujny, Markus Olhofer, Nikola Aulig, Simonetta Boria, and Fabian Duddeck. 2019. Kriging-Assisted Topology Optimization of Crash Structures. *Computer Methods in Applied Mechanics and Engineering* 348 (May 2019), 730–752. https://doi.org/10.1016/j.cma.2019.02.002

[50] Elena Raponi, Dario Fiumarella, Simonetta Boria, Alessandro Scattina, and Giovanni Belingardi. 2021. Methodology for Parameter Identification on a Thermoplastic Composite Crash Absorber by the Sequential Response Surface Method and Efficient Global Optimization. *Composite Structures* (Sept. 2021), 114646. https://doi.org/10.1016/j.compstruct.2021.114646

[51] Elena Raponi, Hao Wang, Mariusz Bujny, Simonetta Boria, and Carola Doerr. 2020. High Dimensional Bayesian Optimization Assisted by Principal Component Analysis. In *Proc. of Parallel Problem Solving from Nature (PPSN) (LNCS, Vol. 12269)*. Springer, 169–183. https://doi.org/10.1007/978-3-030-58112-1_12

[52] Rommel G Regis. 2016. Trust regions in Kriging-based optimization with expected improvement. *Engineering optimization* 48, 6 (2016), 1037–1059.

[53] Malek Ben Salem, François Bachoc, Olivier Roustant, Fabrice Gamboa, and Lionel Tomaso. 2019. Sequential dimension reduction for learning features of expensive black-box functions. (2019).

[54] Petru-Aurelian Simionescu, Gerry Vernon Dozier, and Roger L Wainwright. 2006. A two-population evolutionary algorithm for constrained optimization problems. In *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 1647–1653.

[55] András Sobester, Alexander Forrester, and Andy Keane. 2008. *Engineering Design via Surrogate Modelling: A Practical Guide*. John Wiley & Sons.

[56] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. 2021. Bayesian Optimization Is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. https://doi.org/10.48550/arXiv.2104.10201 arXiv:2104.10201 [cs, stat]

[57] Pradnya A Vikhar. 2016. Evolutionary algorithms: A critical review and its future prospects. In *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*. IEEE, 261–265.

[58] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. 2022. IOHanalyzer: Detailed Performance Analyses for Iterative Optimization Heuristics. *ACM Trans. Evol. Learn. Optim.* 2, 1 (2022), 3:1–3:29. https://doi.org/10.1145/3510426

[59] Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. 2018. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 745–754.

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

[61] Nathan Wycoff, Mickael Binois, and Stefan M Wild. 2019. Sequential learning of active subspaces. *arXiv preprint arXiv:1907.11572* (2019).

[62] Anatoly Zhigljavsky and Antanas Žilinskas. 2021. *Bayesian and high-dimensional global optimization*. Springer.

[63] Anatoly A Zhigljavsky. 2012. *Theory of global random search*. Vol. 65. Springer Science & Business Media.

[64] Juliusz Krzysztof Ziomek and Haitham Bou Ammar. 2023. Are Random Decompositions all we need in High Dimensional Bayesian Optimisation?. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 43347–43368. https://proceedings.mlr.press/v202/ziomek23a.html

APPENDIX

A ALGORITHMS

Here we provide the reader with a more detailed description of the algorithms that were compared in our study and the chosen hyperparameter settings.

A.1 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES)) [20, 24, 25] belongs to the family of evolutionary algorithms [54, 57] and is considered to be the state-of-the-art in this category, being the most commonly used for continuous optimization in many research laboratories and industrial environments. CMA-ES is mostly used to solve difficult nonlinear nonconvex black-box optimization problems, unconstrained or constrained, in continuous domains. It efficiently addresses search spaces of dimension between three and one hundred. CMA-ES is based on an idea similar to the Quasi-Newton method [47], that is, it is a second-order estimator that estimates a positive definite matrix, the covariance matrix, in an iterative way. Unlike the Quasi-Newton method, CMA-ES does not use or approximate gradients and does not even assume their existence. For this reason, it can be used for non-smooth and even non-continuous problems, as well as for multimodal and/or noisy problems. Furthermore, CMA-ES does not require expensive hyperparameter tuning, since the choice of hyperparameters is not left to the user (apart from population size). The only hyperparameters that the user needs to set for the application of CMA-ES are an initial parent design, an initial step-size, and a termination criterion.

In this study, the initial solution was taken as a random sample array in the design space and the initial step-size in each coordinate is equal to 1. We considered a default population size of $4 + 3 \log D$, where $\log D$ is the natural logarithm of the dimension of the design domain.

A.2 Sparse Axis-Aligned Subspaces BO (SAASBO)

One of the most difficult problems with high-dimensional BO is defining an appropriate class of surrogate models. On the one hand, a class of models that is too flexible may lead to overfitting. On the other hand, a class that is too rigid would not be able to capture the important properties of the objective function landscape.

Sparse Axis Aligned Subspace Bayesian Optimization (SAASBO) [13] introduces a new surrogate model for high-dimensional BO based on the assumption that the coordinates of x in S have a relevance hierarchy. According to [13] this approach has several key advantages. First, it preserves the structure of the input space and therefore it can exploit it. Second, it is adaptive and shows low sensitivity to its hyperparameters. Third, it can naturally accommodate both input and output constraints, unlike methods based on random projections for which input constraints are particularly challenging. In SAASBO, the usual Gaussian process is used, but with the help of some new components.

The innovations are:

- sparsity-inducing SAAS function prior;
- combination of the surrogate model with the No-Turn-U-Sampler (NUTS), an adaptive form of Hamiltonian Monte Carlo (HMC) sampling that the algorithm must perform to do inference on that model. It allows the surrogate model to quickly identify the most important low-dimensional subspace, resulting in a sample-efficient BO.

In our experiments, we set the following values for the main hyperparameters:

- α = 0.1, a positive float that controls the level of shrinkage/sparsity of the GP model. Smaller alpha for more sparsity, and so most dimensions "turned off";
- num_warmup = 256, the number of warmup samples to use in the NUTS. During the warm-up, the NUTS algorithm adjusts the HMC algorithm parameters metric and step-size in order to sample efficiently. After the warm-up, the fixed metric and step-size are used to produce a set of draws;
- num_samples = 256, the number of post-warmup samples to use in HMC inference;
- thinning = 32, a positive integer that controls the fraction of posterior hyperparameter samples that are used to compute the expected improvement;
- kernel = rbf. By default, SAASBO uses radial basis function kernels in the GP model definition.

A.3 Random Decomposition Upper-Confidence Bound (RDUCB)

Random Decomposition Upper-Confidence Bound (RDUCB) [64] introduces a decomposition method that assumes additively structured black boxes based on data-independent decomposition rules. The reason behind this is that learning the decomposition of the function based on the data can be misleading if the dataset is not fixed, but rather data is dynamically added during the iterations. Indeed, data-driven approaches can easily fail to apply globally to the entire search space and lead to erroneous local decompositions.

RDUCB is an easy-to-implement approach that leads to empirical gains while supporting rigorous theoretical results. RDUCB uses an additive approach with a tree structure. This method efficiently achieves accurate approximations by randomly sampling trees, ensuring each edge has an equal chance of being selected. The acquisition function to be maximized is an additive version of the upper-confidence bound function. They achieve this through the utilization of message-passing optimizers, which can more effectively leverage the knowledge of the dependency graph. This approach results in improved empirical performance by transmitting information from children to parent nodes of the tree.

In our experiments we set the following values for the main hyperparameters:

- eps = -1, minimum distance between two consecutive x-values to keep running the model;
- exploration_weight = 'lambda t: 0.5 np.log(2t)', a configurable hyperparameter of the acquisition function that controls the exploration behavior, where t denotes the timestep;
- graphSamplingNumIter = 100, number of iterations to build the best graph;
- learnDependencyStructureRate = 1, hyperparameter that controls when to learn a new GP model;
- lengthscaleNumIter = 2, number of samples taken for the parameters during the optimization of the graph;
- max_eval = -4, hyperparameter of the optimizer of the acquisition function that decides how many times to divide the search space into subdomains to find a better solution;
- noise_var = 0.1, observation error of the GP model;
- size_of_random_graph = 0.2, to decide the dimension of the random graph;
- grid_size = 150, which controls the density of the discretization, given that the domain is a discretized representation of the search space.

A.4 Linear PCA-assisted Bayesian Optimization (PCA-BO)

Another method for scaling BO for high-dimensional data is based on the use of Principal Component Analysis (PCA) to generate a new BO algorithm called **PCA-assisted Bayesian optimization (PCA-BO)** [51]. Thanks to PCA,

the algorithm learns a linear transformation based on the points evaluated so far and selects the dimensions in the transformed space considering the variability of these points. Both the fitting of the GPR model and the optimization of the acquisition function are carried out in the space with reduced dimensionality. The primary benefits are the reduction of CPU time for high-dimensional problems and the ability to maintain a good convergence rate for problems with an adequate global structure.

PCA-BO starts with a DoE and adaptively learns a linear map, updated at each iteration, to reduce the dimensionality through a weighted PCA procedure. Weights are used to account for objective values.

The main steps of the PCA-BO algorithm can be summarized as follows:

(1) Generate an initial DoE in the original space composed of a set of evenly distributed points;
(2) Design a weighted scheme to take into account the information from the objective function into the DoE points: smaller weights are assigned to the points with worse function values.
(3) Apply the PCA technique to create a linear map from the original search space to a lower-dimensional space, using the weighted DoE points;
(4) Train a GPR model and maximize an infill criterion to find the new infill point, both in the lower-dimensional space;
(5) Map back the infill point to the original space and evaluate its objective function value;
(6) Append the new infill point and its objective value to the data set and then proceed to Step 2 for a new iteration.

We set the following values for the main hyperparameters:

- n_point = 1, the number of infill points selected during the optimization of the acquisition function;
- n_components = 0.90, the amount of variance that needs to be explained by the principal components kept after dimensionality reduction by PCA;
- acquisition_optimization = BFGS, optimizer used to maximize the acquisition function.

A.5 Kernel PCA-assisted BO (KPCA-BO)

The **kernel PCA-assisted algorithm BO (KPCA-BO)** [1] is an extended version of PCA-BO, which uses kernel methods to first map points to a reproducing kernel Hilbert space (RKHS) [3] using an implicit nonlinear feature mapping. Then, PCA is used in the RKHS to learn a linear transformation from all evaluated points during the run and select dimensions in the transformed space according to the variability of the evaluated points. The main advantage of KPCA-BO over PCA-BO is the nonlinearity of the submanifold in the search space, which makes it more likely that multiple basins of attraction will be discovered simultaneously. Besides learning a forward map from the original space to a lower-dimensional submanifold, it constructs a backward map that converts an infill point determined in the reduced space to the original space, so that it can be evaluated. In this way, the two most extensive procedures of BO, training the GP model and optimizing the acquisition function, are performed in a low-dimensional space, reducing CPU time.

We set the following values for the main hyperparameters:

- n_point = 1, the number of infill points selected during the optimization of the acquisition function;
- max_information_loss=0.1, the decimal value of the maximum information of variability data points that can be lost during the PCA procedure in the Hilbert space;
- kernel_fit_strategy = KernelFitStrategy.AUTO. The AUTO setting uses an RBF kernel.

A.6 Trust Region BO (TuRBO)

Two key issues often lead to poor performance of classical BO in high-dimensional settings: the homogeneity of global probabilistic models and overemphasized exploration due to global coverage. To overcome these problems, the global perspective can be discarded, and a local approach can be used.**Trust regions BO (TuRBO)** [14] is a technique for global optimization that uses a collection of simultaneous local optimization runs with independent probabilistic models. To combine all the local parts and return to a global vision, an implicit multi-armed bandit strategy is used at each iteration to distribute the samples across different local domains and hence decide which local optimization runs are preferred. In each global iteration, which means considering all the trust regions, the algorithm selects a batch of q candidates drawn from the union of all the trust regions, and updates all the local models for which candidates were drawn. A Thompson sampling [29] is used to select infill points within a single trust region and in all trust regions simultaneously.

The main advantages of this approach are that (1) each local surrogate model is robust to noisy observations and uncertainty estimates, (2) the local surrogates allow heterogeneous modeling of the objective function and do not suffer from over-exploitation, and (3) it provides local search trajectories to quickly discover excellent target values.

We set the hyperparameters as follows:

- batch_size = 5, number of infill points found in each iteration,
- max_cholesky_size = 2000, after how many iterations the algorithm switches from Cholesky to Lanczos to train the GP;
- n_training_steps = 50, number of steps of ADAM to learn the hyperparameters of the GP;
- n_cand = min(100 * D, 5000), number of vectors of dimension D generated by a Sobolev sequence where to evaluate the batch_size GP samples;
- failtol = ceil(max([4.0/ batch_size, D/batch_size])) for TuRBO1 and failtol = max(5, D) for TuRBOm, where ceil of the scalar x is the smallest integer i such that $i \leq x$ and it is the threshold for failures after which trust regions halves;
- succtol = 3, the threshold for successes after which trust regions doubles;
- length_min = 0.5^7, the minimum threshold for the base side length of the trust regions before restart;
- length_max = 1.6, the maximum threshold for the base side length of the trust regions;
- length_init = 0.8, value to initialize the base side length of the trust regions.

A.7 Ensemble BO (EBO)

In our data on Zenodo and IOHanalyzer, we also included results for **Ensemble Bayesian Optimization (EBO)** [59]. This algorithm belongs to the additive models category. When dealing with high-dimensional problems, the inefficiency of BO occurs not only in the creation of the surrogate model but also in the optimization of the acquisition function, which is sometimes very expensive to evaluate. Moreover, reliable search and estimation for complex functions in very high-dimensional spaces may require a large number of observations. EBO attempts to answer precisely these three challenges:

(1) large-scale observations,
(2) high-dimensional input spaces,
(3) selection of batches of query points that balance quality and diversity.

To solve the three challenges, the authors propose to improve the GP models by using a hierarchical additive approach based on tile coding, also known as random binning or Mondrian Forest features. Then, thanks to a Gibbs sampling, the posterior distribution is learned over the kernel width and the additive structure to prevent overfitting. The third challenge is to improve the sampler which depends on the likelihood of the observations. This is accelerated by an efficient randomized block approximator of the Gram matrix based on a Mondrian process. Thus, EBO relies on two main ideas implemented at different levels: 1) using efficient partition-based function approximators to simplify and speed up the model-building and the optimization procedure and 2) improving the expressive power of these approximators by using ensembles and a stochastic approach that relies on the Mondrian process. Moreover, they use an ensemble of Tile Gaussian Processes (TileGPs) for each part, a new GP model based on tile coding and additive structure. Their method can be defined as a stochastic method over a randomized and adaptive sample of partitions of the input data space.

The EBO code is taken from the GitHub repository `Ensemble-Bayesian-Optimization`,[13] but we redefine the acquisition function as the EI, because the default implementation uses a global minimum value that is assumed to be known, while we assume that it works in a complete black-box scenario. In our experiments, two different implementations of the EBO algorithm are utilized: EBO and EBO_B. They differ for the value of the hyperparameter B, which represents the number of query points selected at each iteration. We use $B = 1$ and $B = 10$, respectively. To use the same total budget, EBO_B runs for budget/10 iterations.

We set the following values for the main hyperparameters:

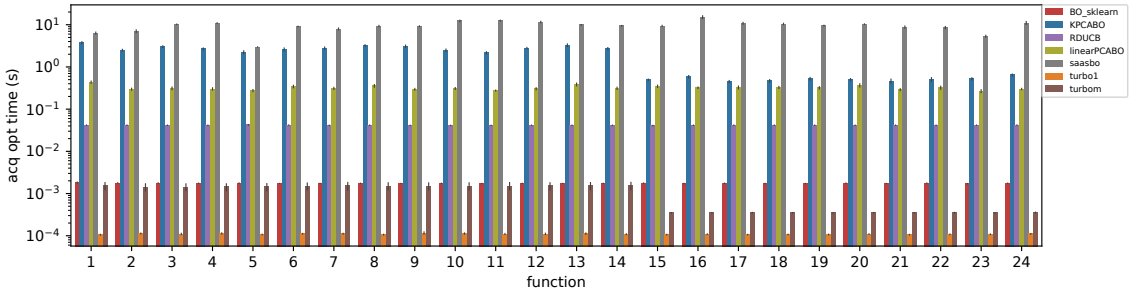
- z = sample_z(D). This parameter controls the additive decomposition of the input feature space. It is an array of dimension D and it can assume only discrete values. Here it is selected randomly through the method `sample_z`;
- k = array([10] * D), the number of cuts in each dimension. It is an array of dimension D and it can assume only discrete values;
- $\alpha = 1$, hyperparameter of the Gibbs sampling subroutine;
- β = array([5., 2.]), hyperparameter of the Gibbs sampling subroutine;
- opt_n = 1000 points randomly sampled to initiate the continuous optimization of the acquisition function;
- gibbs_iter = 10 number of iterations for the Gibbs sampling subroutine;
- nlayers = 100, number of the layers of tiles;
- gp_sigma = 0.1, noise standard deviation;
- n_top = 10, how many points to look ahead when selecting the new infill point.

The advantage we can observe is a low CPU time for optimizing the acquisition function, but this advantage cannot outweigh the poor convergence performance. In fact, stagnation at low budget is a very common behavior of this algorithm in the problems we treat. We attribute this to the choice of its default hyperparameters, which may not have been designed for our function landscapes. For these reasons, EBO was not further considered in great detail in our work, and we decided to use RDUCB [64] as representative for the additive model approaches.

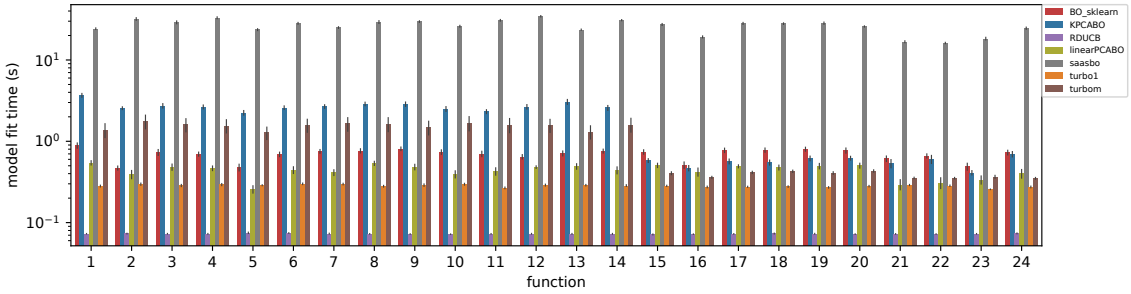
B EXTENSIVE CPU TIME ANALYSIS

Figures 12, 13, 14, and 15 present additional bar plots for the CPU time required to fit the model, optimize the acquisition function, and perform the complete run, for each method and on each BBOB function.

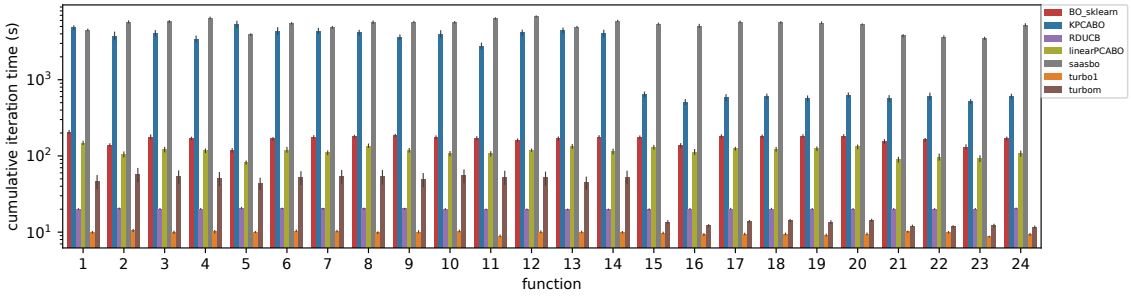
[13] https://github.com/zi-w/Ensemble-Bayesian-Optimization



(a) Bar plot showing the CPU time in seconds for the acquisition function optimization phase function by function. Time is averaged over all iterations of one run.

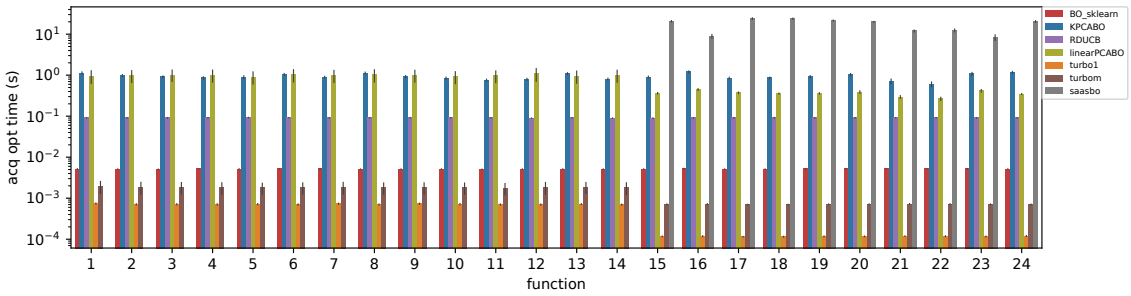


(b) Bar plot showing CPU time in seconds to fit the model, function by function. Time is averaged over all iterations of one run.

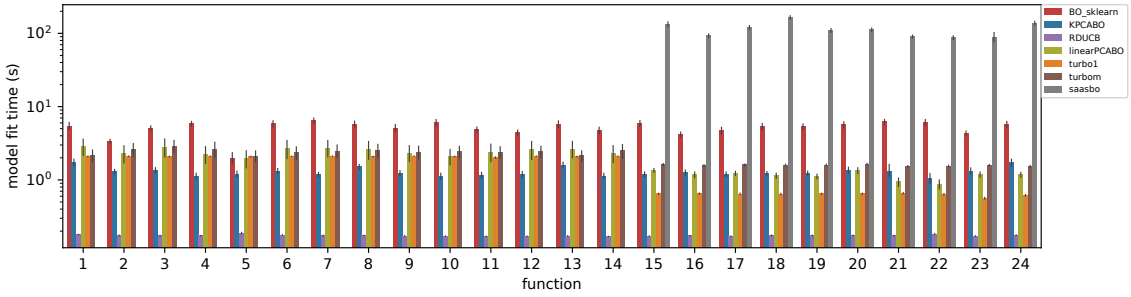


(c) Bar plot showing CPU time in seconds for a whole run, function by function.

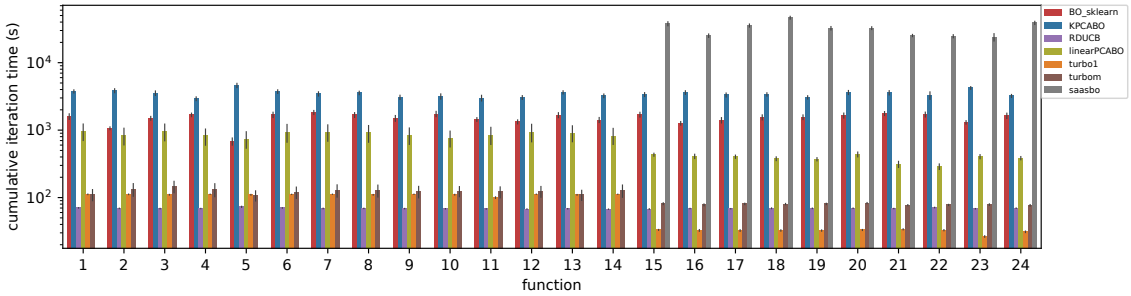
Fig. 12. CPU time bar plots for 10D, function by function.



(a) Bar plot showing the CPU time in seconds for the acquisition function optimization phase function by function. Time is averaged over all iterations of one run.

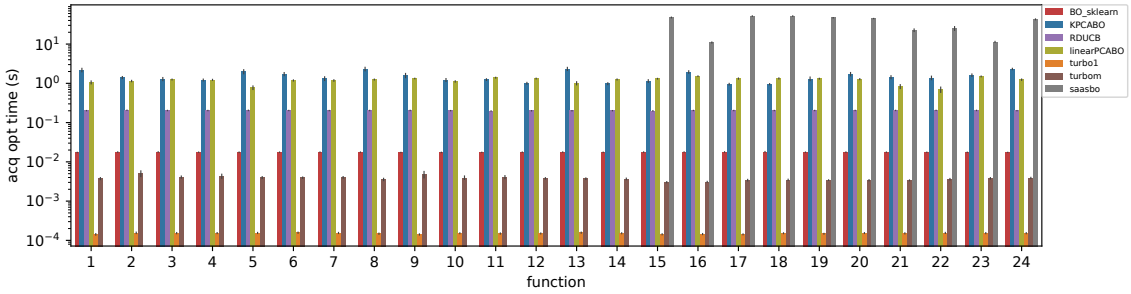


(b) Bar plot showing CPU time in seconds to fit model phase, function by function. Time is averaged over all iterations of one run.

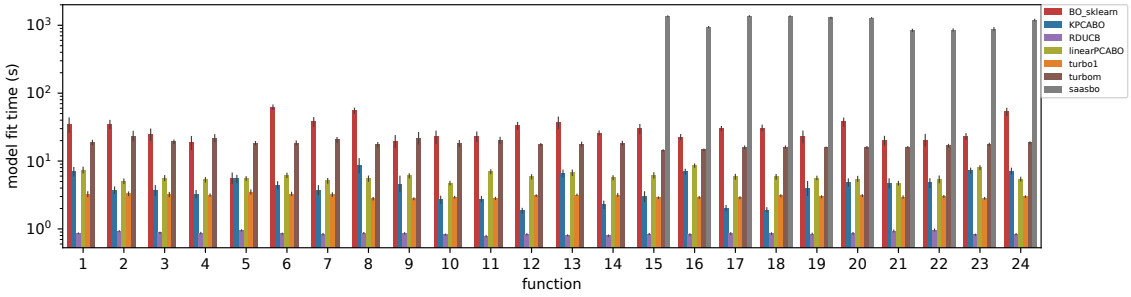


(c) Bar plot showing CPU time in seconds for a whole run, function by function.

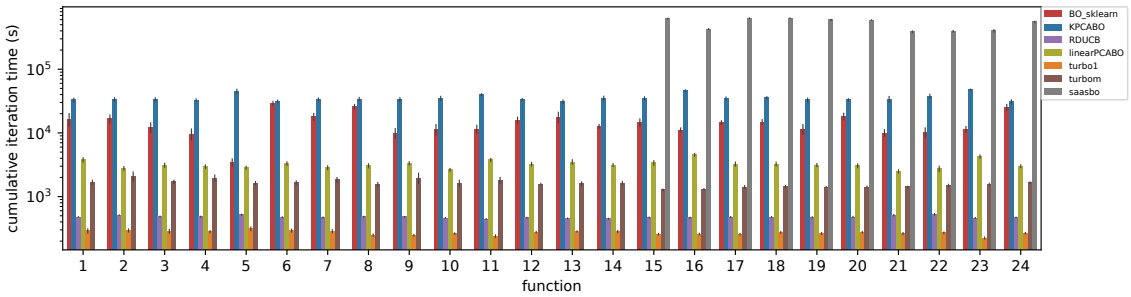
Fig. 13. CPU time bar plots for 20D, function by function.



(a) Bar plot showing the CPU time in seconds for the acquisition function optimization phase function by function. Time is averaged over all iterations of one run.

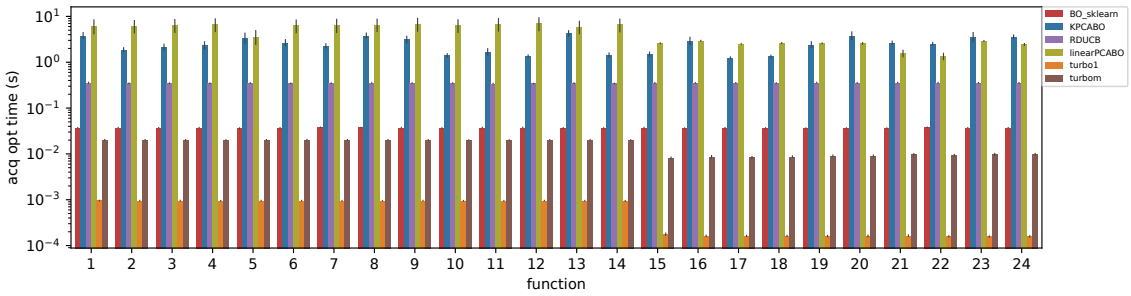


(b) Bar plot showing the CPU time in seconds for the fit of the model phase, function by function. Time is averaged over all iterations of one run.

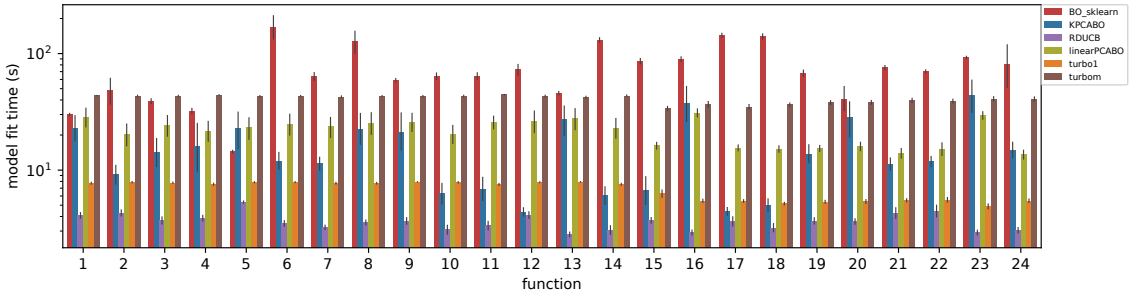


(c) Bar plot showing the CPU time in seconds for a whole run, function by function.

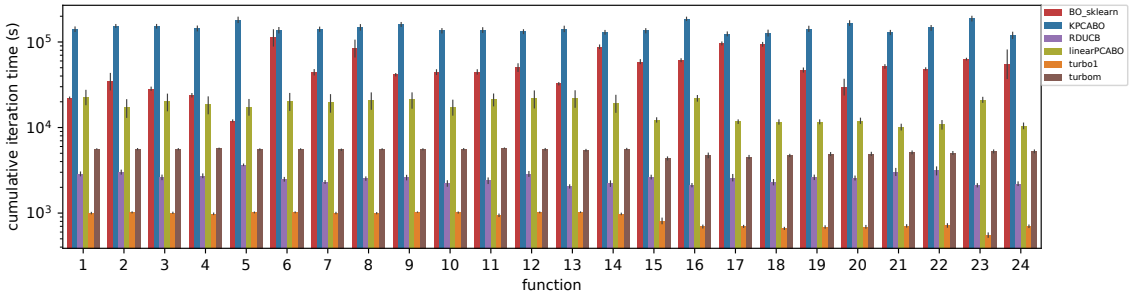
Fig. 14. CPU time bar plots for 40D, function by function.



(a) Bar plot showing the CPU time in seconds for the acquisition function optimization phase function by function. Time is averaged over all iterations of one run.



(b) Bar plot showing CPU time in seconds to fit model phase, function by function. Time is averaged over all iterations of one run.



(c) Bar plot showing CPU time in seconds for the entire process, function by function.

Fig. 15. CPU time bar plots for 60D, function by function.

C VIOLIN PLOTS

Figures 16, 17, and 18 provide a comparison through violin plots for the performance of standard BO, CMA-ES, and the best among the HDBO algorithms for each setting (combination of function and dimension) at the end of the budget. Here we show results for dimensions 10, 20, and 60, while dimension 40 was already discussed in Section 6.

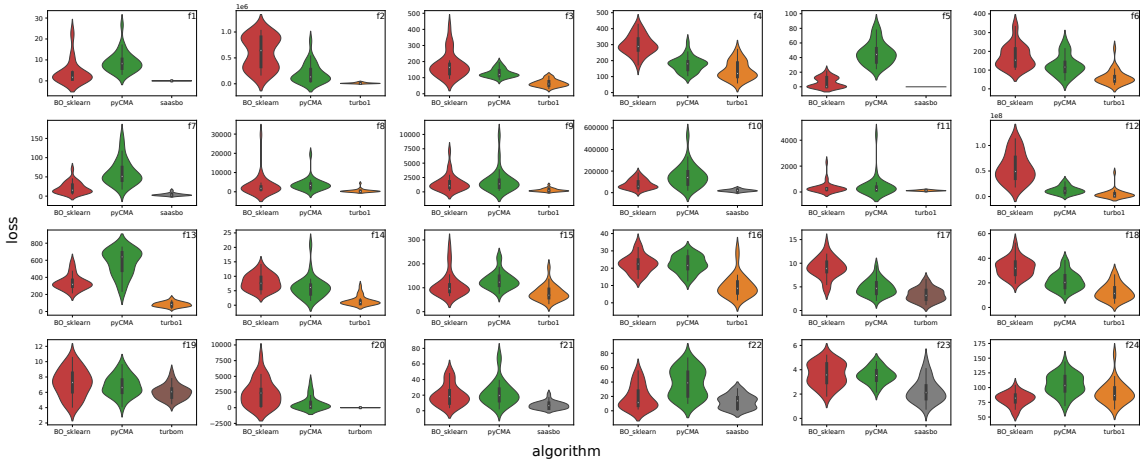


Fig. 16. The violin plots show, for dimension 10 and budget 150 (final budget), the distribution of the best-so-far function values for all functions, obtained by vanilla BO, CMA-ES, and the best among the HDBO algorithms. The plots include a marker for the median of the data and a box indicating the interquartile range.

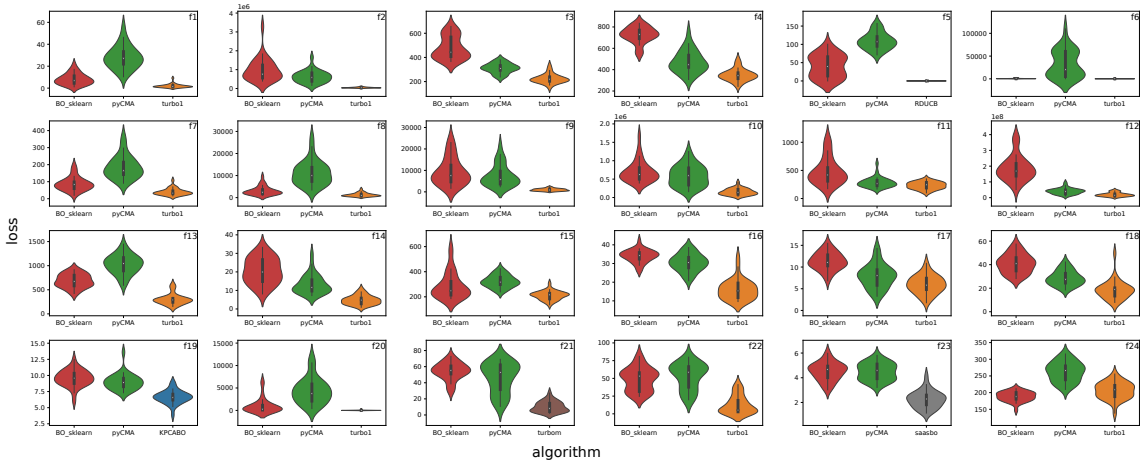


Fig. 17. The violin plots show, for dimension 20 and budget 250 (final budget), the distribution of the best-so-far function values for all functions, obtained by vanilla BO, CMA-ES, and the best among the HDBO algorithms. The plots include a marker for the median of the data and a box indicating the interquartile range.

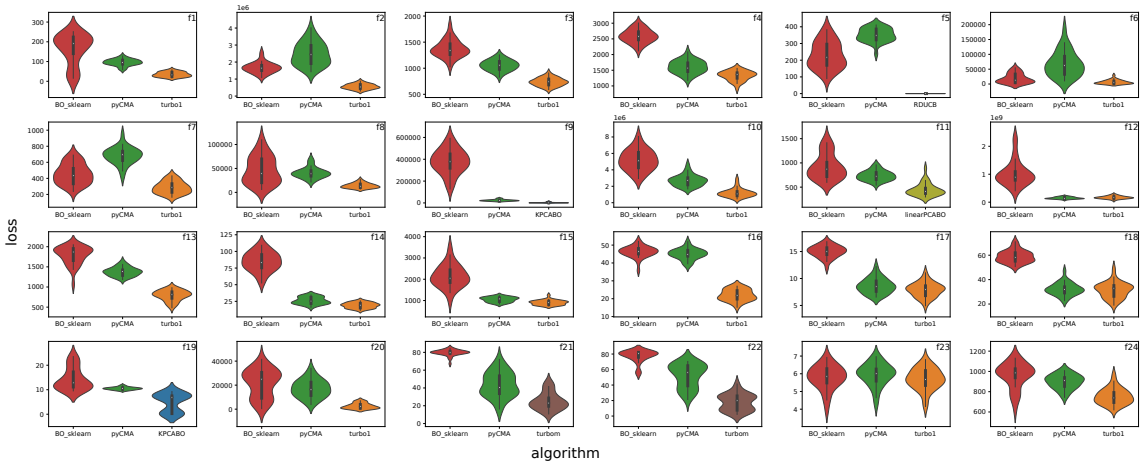


Fig. 18. The violin plots show, for dimension 60 and budget 650 (final budget), the distribution of the best-so-far function values for all functions, obtained by vanilla BO, CMA-ES, and the best among the HDBO algorithms. The plots include a marker for the median of the data and a box indicating the interquartile range.

D ECDF ANALYSIS

For a more comprehensive analysis of the results, we also present cumulative distribution function (ECDF) plots. These plots were generated in the 'Fixed-Target Results: Cumulative Distribution' section of IOHanalyzer [58]. They illustrate the aggregated ECDF curves for selected target values across all 24 BBOB functions. The term 'aggregated' refers to a comprehensive overview resulting from the collective performance across the entire range of functions. For each function, we select 10 target values logarithmically distributed within the range defined by the minimum and maximum values attained by all the algorithms. For a given number of random targets and each method, our plots show the percentage (shown on the y-axis) of runs that achieved that target within a given budget (shown on the x-axis), with this percentage then averaged across all the different fixed targets.

D.1 Dimension D = 10

Figure 19 confirms the assertions made in Section 5.1. Random search proves to be the least effective method. Initially, CMA-ES exhibits poor performance, but it shows a slight improvement over time. Vanilla BO demonstrates strong performance and greater robustness across evaluations when compared to other algorithms like PCA-BO, KPCA-BO, RDUCB, and CMA-ES. SAASBO achieves the highest percentage of target values nearly throughout the entire budget range. Towards the end of the run, TuRBO1 continues to enhance the percentage without stagnation, overcoming SAASBO at approximately 110 evaluations.

D.2 Dimension D = 20

Figure 20 validates the claims stated in Section 5.2 and distinctly illustrates that all the examined algorithms outperform random search. Even in dimension 20, vanilla BO clearly outperforms CMA-ES. SAASBO and TuRBO1 exhibit comparable performance within a small evaluation budget. As the number of evaluations increases, TuRBO1 demonstrates more pronounced distinctions compared to other algorithms than in dimension 10, surpassing SAASBO. It is important

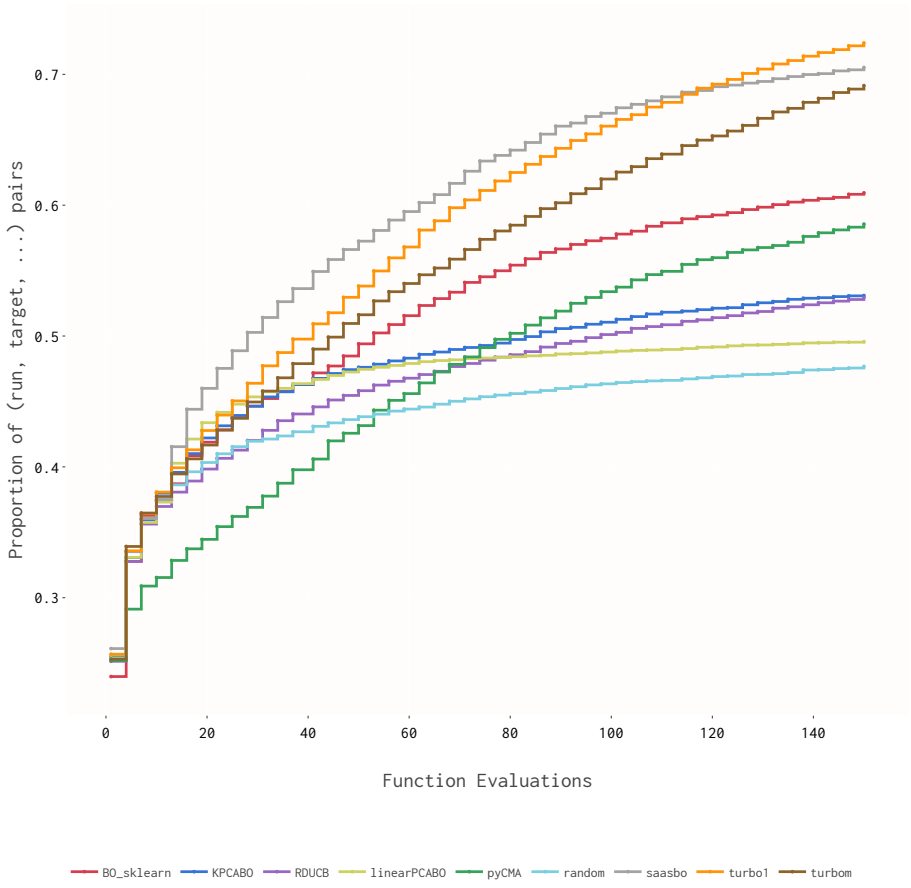


Fig. 19. 'Fixed-Target Results: Cumulative Distribution': Aggregated ECDF curves for selected target values for dimension 10.

to note that, for this dimension, we have complete data only for f15-f24. Hence, aggregation is based only on these functions. The sudden performance boost of RDUCB explained in Section 5.2, is also evident: around budget 200, all runs show an improvement, increasing the probability of reaching the target value.

D.3 Dimension D = 40

Figure 21 is in line with the analysis presented in Section 5.3. It is worth noting that while vanilla BO performs strongly up to dimension 20, here it stands out as one of the less effective strategies. It is also noteworthy that PCA-BO shows a slightly higher probability of reaching the target values than other algorithms for small budgets. As the budget increases, the superiority of TuRBO1 over all other algorithms is evident. SAASBO is no longer comparable with TuRBO1, even if the aggregation is based only on the functions f15-f24, as in dimension 20.

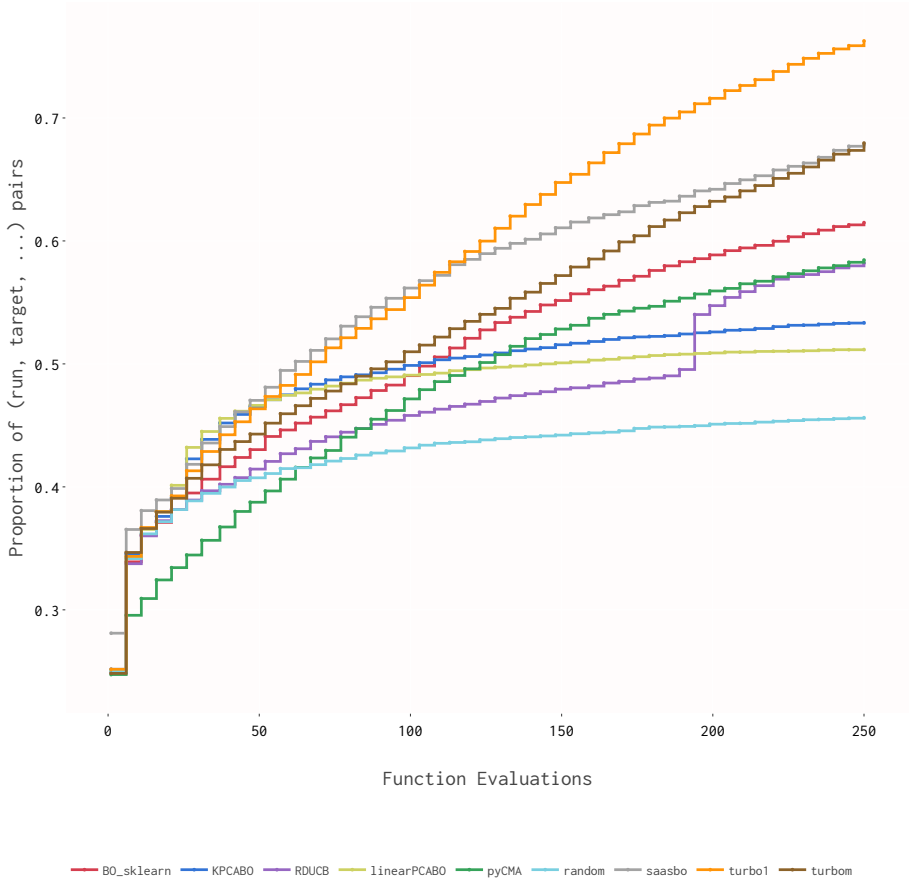


Fig. 20. 'Fixed-Target Results: Cumulative Distribution': Aggregated ECDF curves for selected target values for dimension 20.

D.4 Dimension D = 60

The ECDF curves in Figure 22 validate the observations presented in Section 5.4. The ECDF curve for SAASBO is unavailable for dimension 60, as experiments for this algorithm were not completed due to the prohibitive computational costs of the runs. BO clearly suffers from the high dimension. The potential of PCA-BO and KPCA-BO for small evaluation budgets becomes even more evident than in dimension 40. They stand out as the most likely to reach the target values, which highlights the effectiveness of KPCA-BO and PCA-BO in the early stages of optimization. However, they tend to plateau around evaluation 200. On the other hand, TuRBO demonstrates an enhancement in the percentage with an increasing budget, positioning it as one of the most effective methods. Considering the full evaluation budget, it is followed by CMA-ES and RDUCB.

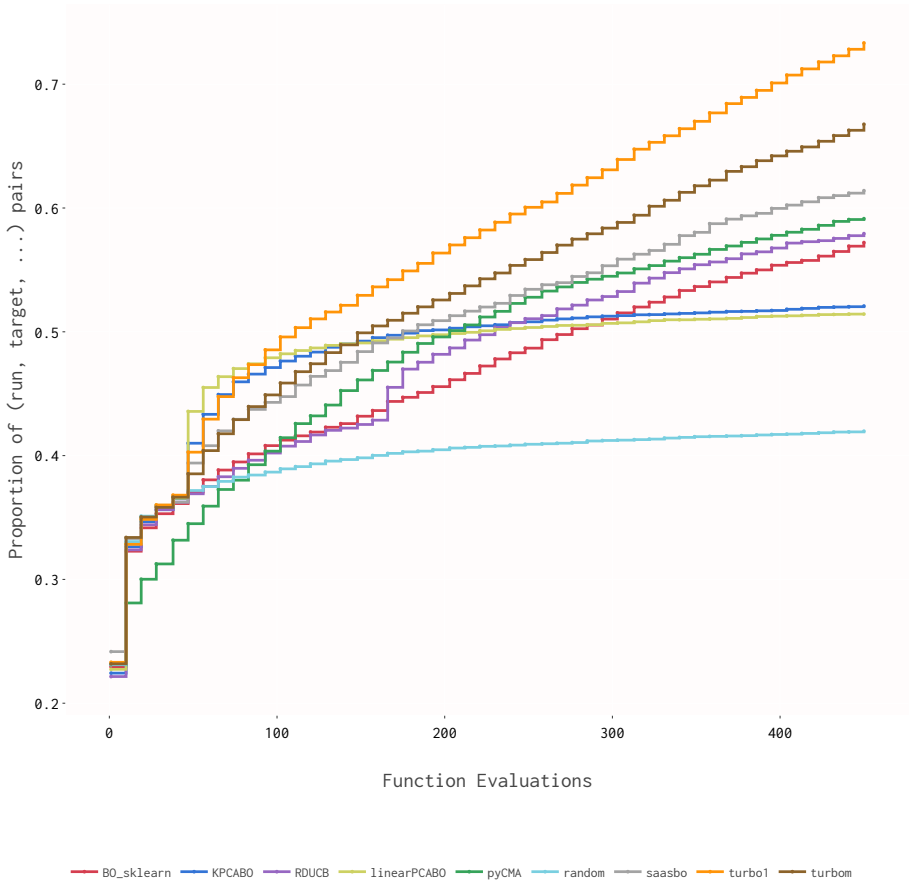


Fig. 21. 'Fixed-Target Results: Cumulative Distribution': Aggregated ECDF curves for selected target values for dimension 40.

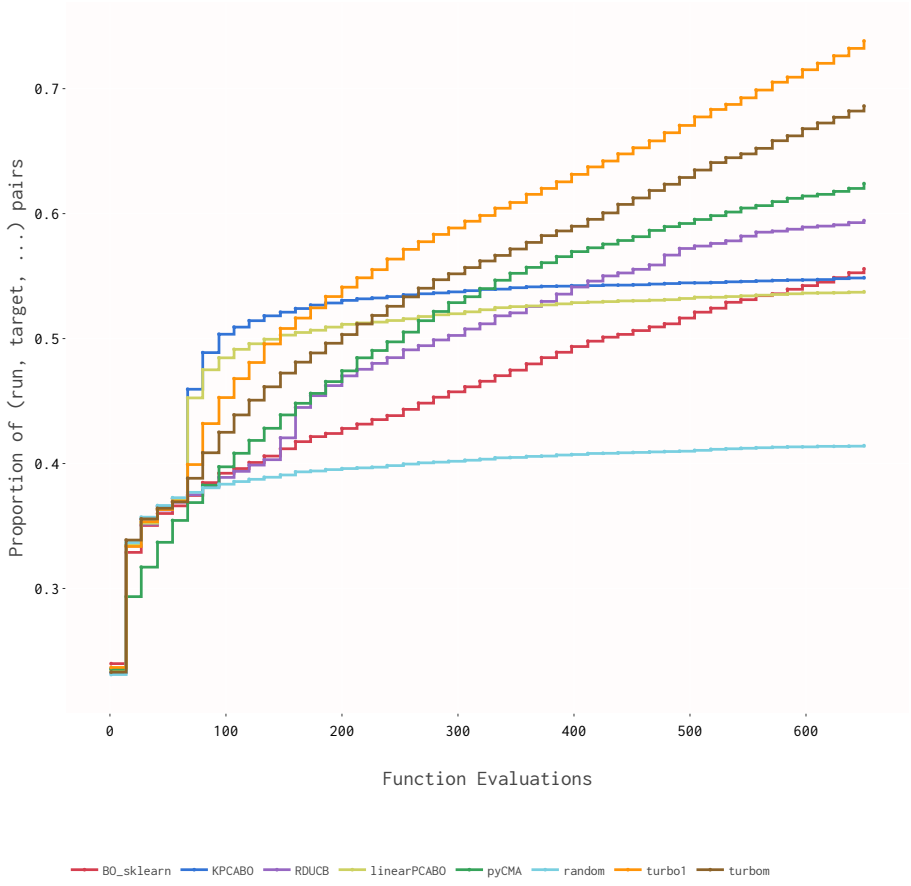


Fig. 22. 'Fixed-Target Results: Cumulative Distribution': Aggregated ECDF curves for selected target values for dimension 60.