EXPObench

Please check the document version of this publication:

EXPObench: Benchmarking Surrogate-based Optimisation Algorithms on Expensive Black-box Functions

Laurens Bliek, Arthur Guijt, Rickard Karlsson, Sicco Verwer, Mathijs de Weerdt

Abstract—**Surrogate algorithms such as Bayesian optimisation are especially designed for black-box optimisation problems with expensive objectives, such as hyperparameter tuning or simulation-based optimisation. In the literature, these algorithms are usually evaluated with synthetic benchmarks which are well established but have no expensive objective, and only on one or two real-life applications which vary wildly between papers. There is a clear lack of standardisation when it comes to benchmarking surrogate algorithms on real-life, expensive, black-box objective functions. This makes it very difficult to draw conclusions on the effect of algorithmic contributions. A new benchmark library, EXPObench, provides first steps towards such a standardisation. The library is used to provide an extensive comparison of six different surrogate algorithms on four expensive optimisation problems from different real-life applications. This has led to new insights regarding the relative importance of exploration, the evaluation time of the objective, and the used model. A further contribution is that we make the algorithms and benchmark problem instances publicly available, contributing to more uniform analysis of surrogate algorithms. Most importantly, we include the performance of the six algorithms on all evaluated problem instances. This results in a unique new dataset that lowers the bar for researching new methods as the number of expensive evaluations required for comparison is significantly reduced.**

I. Introduction

Unlike other black-box optimisation algorithms, surrogate-based optimisation algorithms such as Bayesian optimisation [1] are designed specifically to solve problems with expensive objective functions. Examples are music generation [2], materials science [3], temperature control [4], optics [5], and computer vision [6]. By making use of a surrogate model that approximates the objective function, these algorithms achieve good results even with a low number of function evaluations. However, the training and usage of the surrogate model is more computationally intensive than the use of typical black-box optimisation heuristics such as local search or population-based methods. This complicates thorough benchmarking of surrogate algorithms.

The current way of benchmarking surrogate algorithms does not give complete insight into the strengths and weaknesses of the different algorithms, for a variety of reasons. The most

L. Bliek is at the School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands, l.bliek@tue.nl

R. Karlsson is at the Department of Mathematical Sciences, Chalmers University of Technology, rickard.karlaxel@icloud.com

A. Guijt , S. Verwer and M. de Weerdt are at the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, the Netherlands, {A.Guijt-1,S.E.Verwer,M.M.deWeerdt}@tudelft.nl

important reason is the lack of a standard benchmark set of problems that come from real-life applications and that also have expensive objective functions. Another reason is the lack of insight in the computational efficiency of surrogate algorithms.

In this work, we compare several surrogate algorithms on the same set of expensive optimisation problems from real-life applications, resulting in a public benchmark library that can be easily extended both with new surrogate algorithms, as well as with new problems. Our other contributions are:

- the creation of a meta-algorithmic dataset of surrogate algorithm performance on real-life expensive problems,
- insight into the strengths and weaknesses of existing surrogate algorithms and verifying existing knowledge from literature,
- investigating how algorithm performance depends on the available computational resources and the cost of the expensive objective,
- separating the effects of the choice of surrogate model and the acquisition step of the different algorithms.

We furthermore show that continuous models can be used on discrete problems and vice versa. The main insights that we obtained are that the accuracy of a surrogate model and the choice of using a continuous or discrete model, are less important than the evaluation time of the objective and the way the surrogate algorithm explores the search space.

II. Background and related work

This section starts by giving a short explanation of surrogate-based optimisation algorithms, or surrogate algorithms for short. We then describe some of the shortcomings in the way surrogate algorithms are currently benchmarked: the lack of standardised benchmarks and the lack of insight in computational efficiency. Finally, we give an overview of related benchmark libraries and show how our library fills an important gap.

A. Surrogate-based optimisation algorithms

In surrogate optimisation, the goal is to minimise an *expensive* black-box objective function

$$\min_{x \in X} f(x), \tag{1}$$

where $X \subseteq \mathbb{R}^d$ is the d-dimensional search space with d the number of decision variables. The objective can be expensive

for various reasons, but in this work we assume f is expensive in terms of computational resources, as it involves running a simulator or algorithm. Optimising f using standard black-box optimisation algorithms such as local search methods or population-based techniques may require too many evaluations of the expensive objective.

Surrogate algorithms reduce the number of required objective evaluations by iterating over three steps at every iteration i:

1) (*Evaluation*) Evaluate $y_i = f(x_i)$ for a candidate solution x_i.
2) (*Training*) Update the surrogate model $g : X \to \mathbb{R}$ by fitting the new data point (x_i, y_i).
3) (*Acquisition*) Use g to determine a new candidate solution x_{i+1}.

Usually, in the first R iterations x_i is chosen randomly and therefore the acquisition step is skipped for these iterations. The training step consists of machine learning techniques such as Gaussian processes or random forests, where the goal is to approximate the objective f with a *surrogate model g*. For the acquisition step, an *acquisition function α* is used that indicates which region of the search space is the most promising by trading off exploration and exploitation:

$$x_{i+1} = \underset{x \in X}{\operatorname{argmax}} \, \alpha(g(x)). \qquad (2)$$

Example acquisition functions are Expected Improvement, Upper Confidence Bound, or Thompson sampling [1].

By far the most common surrogate algorithm is Bayesian optimisation [1], [7], which typically uses a Gaussian process surrogate model. Other common surrogate models are random forests, as used in the SMAC algorithm [8], and Parzen estimators, as used in HyperOpt [6]. Our own earlier work contains random Fourier features as surrogate models in the DONE algorithm [5] and piece-wise linear surrogate models in the IDONE and MVRSM algorithms [9], [10]. An overview of different methods and their surrogate models is given in Table I. Details about which methods are included in the comparison are given in Section III-B.

Since the training and acquisition steps above are relatively expensive even if f would not be expensive to evaluate, surrogate algorithms are usually not compared on large numbers of synthetic benchmarks, as is common for other black-box optimisation methods that require less computational resources per iteration. The most common way of benchmarking in surrogate-algorithm literature at this moment is to test a new or variant surrogate algorithm and compare it with similar algorithms on some synthetic functions, and then on one or two real-life applications, which vary wildly across papers. While this approach makes sense given the computational resources required to run surrogate algorithms, it does not give enough insight into which algorithm to use in practice. In this section we briefly describe two shortcomings in the current way of benchmarking surrogate algorithms.

B. Shortcoming 1: lack of standardised real-life benchmarks

As mentioned, surrogate models have been applied to expensive objective functions in many different areas. A ques-

tionnaire on real-life optimisation problems [14] confirms that this type of objective function often appears in practice: *"For example, we find that work specialising in handling expensive optimisation problems (such as e.g. surrogate-assisted optimisation) is highly relevant, as several responses to the questionnaire report objective/constraint evaluation times of more than one day."*

Since most surrogate algorithms are developed with the goal of being applicable to many different problems, these algorithms should be tested on multiple benchmark functions. Preferably, these benchmarks are *standardised*, meaning that they are publicly available, easy to test on, and used by a variety of researchers. For synthetic benchmarks, standardised benchmark libraries such as COCO [15] have been around for several years now, and these types of benchmark functions are often used for the testing of surrogate algorithms as well. However, benchmarks from real-life applications are much harder to find, even though they are common in practice. As noted in [16], *"unfortunately, despite its importance, studies to compare various optimisation algorithms on real-world problems are still limited, mainly because such problems are typically not publicly available. "*

Simply taking the benchmarking results on synthetic functions and applying them to expensive real-life applications, or adding a delay to the synthetic function, is not enough [16]–[20]. An example is the ESP benchmark discussed later in this paper. For this benchmark we have noticed that changing only one of the variables at a time leads to no change in the objective value at all, meaning that there are more 'plateaus' than in typical synthetic functions used in black-box optimisation. Another example is that of hyperparameter tuning, a problem that is known to sometimes contain properties that are not present in common synthetic benchmark functions [21]. In general, expensive objectives are often expensive because they are the result of some kind of complex simulation or algorithm, and the resulting fitness landscape is therefore much harder to analyse/model than that of a synthetic function. In contrast, synthetic functions can simply be described with a mathematical function.

It is clear that there is a need for a more standardised approach in benchmarking real-life optimisation problems, especially for surrogate algorithms. Even though this lack of expensive benchmarks already holds for the typical problems assumed in surrogate algorithms, namely that of continuous optimisation of expensive objectives, the need is particularly high for expensive discrete problems as noted in [22]: *"While benchmarking is still not resolved for continuous model-based optimisation, the situation is even less settled in the discrete domain. Of the few published, real-world, expensive, combinatorial problems, most are not openly accessible. Even in case of availability, the benchmark set would be rather small and the expense of computation would hinder broader experimental studies."*

C. Shortcoming 2: lack of insight in computational efficiency

In many works on surrogate algorithms, computation times of the algorithms are not taken into consideration, and are

TABLE I
SURROGATE-BASED APPROACHES IN THIS BENCHMARK ENVIRONMENT.

Name	Surrogate model	Continuous	Integer	Categorical	Conditional
SMAC [8], [11]	Random forest	✓	✓	✓	✓
HyperOpt [6]	Parzen estimator	✓	✓	✓	✓
Bayesian Opt. [1], [12]	Gaussian process (GP)	✓			
CoCaBO [13]	GP+multi-armed bandit	✓		✓	
DONE [5]	Random Fourier	✓			
IDONE [9]	Piece-wise linear		✓		
MVRSM [10]	Piece-wise linear	✓	✓		

often not even reported. This is because of the underlying assumption that the expensive objective is the bottleneck. However, completely disregarding the computation time of the surrogate algorithm leads to the development of algorithms that are too time-consuming to be used in practice. In some cases, the algorithms are even slower than the objective function of the real-life application, shifting the bottleneck from the expensive objective to the algorithm. Computation times should be reported, preferably for problems of different dimensions so that the scalability of the algorithms can be investigated. This also helps answering the open question posed in [18]: *"One central question to answer is at what point an optimisation problem is expensive "enough" to warrant the application of surrogate-assisted methods."* Since many surrogate algorithms have a computational complexity that increases with every new function evaluation [10], even more preferable is to report the computation time used by the surrogate algorithm *at every iteration* to gain more insight into the time it takes to run surrogate algorithms for different numbers of iterations.

Besides the computation time used by the algorithms, different real-life applications have different *budgets* available that put a limit on the number of function evaluations or total computation time. Taking this computational budget into account is a key issue when tackling real-world problems using surrogate models [16]. Yet for most surrogate algorithms, it is not clear how they would perform for different computational budgets.

D. Related benchmark environments

From the way surrogate algorithms are currently benchmarked and the shortcomings that come with it, we conclude that we do not sufficiently understand the performance regarding both quality and run-time on realistic expensive black-box optimisation problems. A *benchmark library* can help in gaining more insight as algorithms are compared on the same set of test functions. In the context of black-box optimisation, such a library consists of multiple objective functions and their details (such as the search space and problem dimension, whether variables are continuous or integer, etc.) and possibly of baseline algorithms that can be applied to the problems. For non-expensive problems, many such libraries exist [15], [23], [24], particularly with synthetic functions. Some of these libraries also contain real-life functions that are not expensive [25]–[27]. See Table II for an overview of related benchmark environments.

The real-life problems to which surrogate algorithms are usually applied can roughly be divided into computer science problems and engineering problems, or digital and physical problems. Examples of the former are automated algorithm configuration [8] and hyperparameter tuning for machine learning [28], while the latter deal with (simulators of) a physical problem such as aerodynamic optimisation [29]. Even though surrogate models are used in both problem domains, these two communities often stay separate: most benchmark libraries that contain *expensive* real-life optimisation problems only deal with one of the two types, for example in automated machine learning [30]–[34] or computational fluid dynamics [35]. A notable exception is Nevergrad [36], which contains a wide variety of problems varying from power plant simulation to neural control of robots. The problem with focusing on only one of the two domains is that domain-specific techniques such as early stopping of machine learning algorithms [37] or adding gradient information from differential equations [38], [39] are exploited when designing new surrogate algorithms, making it difficult to transfer the domain-independent scientific progress in surrogate algorithms from one domain to the other.

Most of the benchmark libraries do not contain any benchmark *solutions* in the form of surrogate algorithms, and often not even any type of solution at all. Besides Nevergrad, benchmark libraries that contain more than one surrogate algorithm and more than one expensive problem are computer science libraries such as HPOlib, BayesMark, AMLB, and AClib, which do not contain any engineering or physical problems. One exception is SUMO [40], a commercial toolbox that contains many surrogate models, and a wide variety of applications. Unfortunately, this Matlab tool is over 10 years old, and only a restricted version is available for researchers, making it less suitable for benchmarking. It only supports low-dimensional continuous problems, and newer surrogate algorithms that were developed in the last decade are not implemented.

What is currently missing is a modern benchmark library that is aimed at real-life expensive benchmark functions not just from computer science but also from engineering, and that also contains baseline surrogate algorithms that can easily be applied to these benchmarks such as SMAC, HyperOpt, and Bayesian optimisation with Gaussian processses.

III. PROPOSED BENCHMARK LIBRARY: EXPOBENCH

In this section we introduce EXPObench: an EXPensive Optimisation benchmark library. We propose a benchmark

TABLE II
RELATED BENCHMARK ENVIRONMENTS

Name	Contains expensive problems	Contains engineering problems	Contains computer science problems	Implemented surrogate algorithms
CoCo [15]				0
Heuristiclab [23]				0
ParadisEO [24]				0
HyFlex [25]			✓	0
SOS [26]		✓		0
IOHprofiler [27]		✓	✓	0
GBEA [34]	✓		✓	0
CFD [35]	✓	✓		0
CompModels [41]	✓	✓		0
NAS-Bench [31]–[33]	✓		✓	0
DAC-Bench [42]	✓		✓	0
RBFopt [43]		✓		1
HPObench [44]	✓		✓	2
AMLB [30]	✓		✓	2
Nevergrad [36]	✓	✓	✓	2
BayesMark [45]	✓		✓	3
MATSuMoTo [46]				4
AClib2 [47]	✓		✓	4
PySOT [48]				5
EXPObench[1]	✓	✓	✓	**7**
SUMO [40]	✓	✓	✓	9
SMT [39]		✓		11

suite focusing on single-objective, expensive, real-world problems, consisting of many integer, categorical, and continuous variables or mixtures thereof. The problems come from different engineering and computer science applications, and we include seven baseline surrogate algorithms to solve them. See Table II for details on how EXPObench compares to related benchmark environments.

The simple framework of this benchmark library makes it possible for researchers in surrogate models to compare their algorithms on a standardised set of real-life problems, while researchers with expensive optimisation problems can easily try a standard set of surrogate algorithms on their problems. This way, our benchmark library advances the field of surrogate-based optimisation.

It should be noted that synthetic benchmark functions are still useful, as they are less time-consuming and have known properties. We therefore still include synthetic benchmarks in our library, though we do not discuss them in this work. We encourage researchers in surrogate models to use synthetic benchmarks when designing and investigating their algorithm, and then use the real-life benchmarks presented in this work as a stress test to see how their algorithms hold up against more complex and time-consuming problems.

In the remainder of this section, we describe the problems and the approaches to solve these problems that we have added to EXPObench.

A. Included Expensive Benchmark Problems

The problems that were included in EXPObench were selected in such a way that they contain a variety of applications,

dimensions, and search spaces. To encourage the development of surrogate algorithms for applications other than computer science, we included several engineering problems, one of which was first introduced in the CFD benchmark library [35]. The problem dimensions were chosen to be difficult for standard surrogate algorithms: Bayesian optimisation is typically applied to problems with less than 10 variables. Two of our problems have 10 variables, though it is possible to scale them up, while the other problems contain tens or even over 100 variables. This is in line with our view of designing surrogate algorithms using easy, synthetic functions, and then testing them on more complicated real-life applications. Since *discrete* expensive problems are also an active research area, we included one discrete problem and even a problem with a mix of discrete and continuous variables.

The problems were carefully selected to have expensive objectives that take longer to evaluate than synthetic functions, but not so long that benchmarking becomes impossible. On our hardware (see Section IV-A), the time it takes to evaluate the objective function varies between 2 and 60 seconds depending on the problem.

We now give a short description of the four real-life expensive optimisation problems that are present in EXPObench.

1) Wind Farm Layout Optimisation (Windwake): This benchmark utilises a wake simulator called FLORIS [49] to determine the amount of power a given wind farm layout produces. The wake effect of wind turbines can cause other turbines on a wind farm to produce less power due to turbulence. Compared to other simulators, FLORIS does not model complex effects and is therefore computationally cheaper to run.

The original simulator calculates the power of the wind farm for a given layout and randomly generated wind rose data. To make the layout more robust to different wind conditions, we decided to use as output instead the power averaged over multiple scenarios, where each scenario uses randomly generated wind rose data, generated with the same distribution. That is, we look at a Monte Carlo simulation to compute the average power, in line with applications in for example logistics [9], [50], [51]. This causes the objective to be stochastic, something that many traditional optimisation algorithms struggle with.

The computational cost of the objective is dependent on the number of wind turbines w, as well as the number of generated scenarios s. A solution is represented by a sequence of w pairs of continuous (x, y) coordinates for each wind turbine, which can take on continuous values. The output is -1 times the power averaged over multiple scenarios, which takes about 15 seconds to compute on our hardware for $w = 5, s = 5$.

It should be noted that this particular problem has constraints besides upper and lower bounds for the position of each wind turbine: turbines are not allowed to be located within a factor of two of each others' radius. As the goal of this work is not to compare different ways to handle constraints, we use the naive approach of incorporating the constraint in the objective. The objective simply returns 0 when constraints are violated.

2) Pipe Shape Optimisation (Pitzdaily): One of the engineering benchmark problems proposed in the CFD library [17], called PitzDaily, is pipe shape optimisation. This benchmark uses a computational fluid dynamics simulator to calculate the pressure loss for a given pipe shape. The pipe shape can be specified using 5 control points, giving 10 continuous variables in total. The time to compute the pressure loss varies from 2 to 60 seconds on our hardware. Although the search space is continuous, there are constraints to this problem: violating these constraints returns an objective value of 2, which is higher than the objective value of feasible solutions.

3) Electrostatic Precipitator (ESP): This engineering benchmark contains only discrete variables. The ESP is used in industrial gas filters to filter pollution. The spread of the gas is controlled by metal plates referred to as baffles. Each of these baffles can be solid, porous, angled, or even missing entirely. This categorical choice of configuration for each baffle constitutes the search space for this problem.

There are 49 baffle slots in total, that each have 8 categorical options. The output is calculated using a computational fluid dynamics simulator [52], which takes about 28 seconds to return the output value on our hardware.

4) Hyperparameter Optimisation and Preprocessing for XGBoost (HPO): This automated machine learning benchmark is a hyperparameter optimisation problem: the approach, namely an XGBoost [53] classifier, has already been selected, but contains a significant number of configuration parameters of various types, including parameters on the pre-processing step. Variables are not only continuous, integer or categorical, but also conditional: some of them remain unused depending on the value of other variables. In total, there are 135 variables, most of which are categorical.

The configuration is evaluated by 5-fold cross-validation on the Steel Plates Faults dataset[2], and the output of the objective uses this value multiplied with -1. Since there can be a trade-off between accuracy and computation time for different configurations, we set a time limit of 8 seconds, as this was roughly equal to twice the time it takes to use a default configuration on our hardware. Configurations for which the time limit is violated, return an objective value of 0.

B. Approaches

In this section we show the approaches that are considered in the benchmark library. We limit ourselves to popular single-objective surrogate algorithms that are easily implemented and open-source, and that do not focus on extensions of the expensive optimisation problem such as a batch setting, multi-fidelity setting, highly constrained problems, etc. These include a Bayesian optimisation algorithm [1], [12], which uses Gaussian processes with a Matérn 5/2 kernel, SMAC [8], and HyperOpt [6]. We also include our own earlier work [5], [9], [10], with the DONE, IDONE and MVRSM algorithms. These make use of either random Fourier features or piece-wise linear functions as the surrogate model. A modern variant of Bayesian optimisation, namely CoCaBO [13], is also included in the benchmark library, but not presented in this work due

to the required computation time. The baseline with which all algorithms are compared is random search [54], for which we use HyperOpt's implementation. We also include several local and global search algorithms in our library (Nelder-Mead, Powell's method and basin-hopping among others), but these failed to outperform random search on all of our benchmark problems, and are therefore not presented in this work.

Not all of these algorithms can deal with all types of variables, although often naive implementations are possible: discretisation to let discrete surrogates deal with continuous variables, rounding to let continuous surrogates deal with discrete variables, and/or ignoring the conditional aspect of a variable entirely. Table I shows the types of variables that are directly supported by the surrogate models used in each algorithm.

IV. RESULTS

The different surrogate algorithms are objectively compared on all four different real-life expensive benchmark problems of EXPObench. The goals of the experiments are three-fold:

- gain insight into the strengths and weaknesses of existing surrogate algorithms and verify existing knowledge from literature,
- investigate how algorithm performance depends on the available computational resources and the cost of the expensive objective,
- separate the effects of the choice of surrogate model and the acquisition step of the different algorithms.

The results of comparing the different surrogate algorithms on the problems of EXPObench provide a new dataset that we use for these three goals, and that we make available publicly.[3]

This dataset includes the points in the search space chosen for evaluation by each algorithm, the resulting value of the expensive objective, the computation time used to evaluate the objective, and the computation time used by the algorithm to suggest the candidate point. The latter includes both the training and acquisition steps of the algorithms, as it was not easy to separate these two for all algorithms. Although we perform some initial analysis on this meta-algorithmic dataset, it can also be used by future researchers in, for example, instance space analysis [55], meta-learning [56], or building new surrogate benchmarks from this tabular data [33].

We start this section by giving the experimental details, followed by the results on the four benchmark problems. We then investigate the influence of the computational budget and cost of the expensive objective, followed by a separate investigation of the choice of surrogate model.

A. Experiment details

1) Hardware: We use the same hardware when running the different surrogate algorithms on the different benchmark problems. All these experiments are performed in Python, on a Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz with 32 GB of RAM. Each approach and evaluation was performed using only a single CPU core.

---

[2]http://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults

[3]The dataset can be found online at https://doi.org/10.4121/14247179

2) Hyperparameter settings: All methods use their default hyperparameters with the exception of SMAC, which we set to deterministic mode to avoid repeating the exact same function evaluations, which drastically decreased performance in our experience. For the MVRSM method, we set the number of basis functions in purely continuous problems to 1000. We have not adapted IDONE for continuous or mixed problems.

3) Normalisation: To make comparison between benchmarks easier, we normalise the best objective value found by each algorithm at each iteration in the figures shown in this section. This is done as follows: using the best objective value found by random search as a baseline, let r_0 be the average of this baseline after 1 iteration, and let r_1 be the average of this baseline after the number of random initial guesses R that each algorithm used. Then all objective values f are normalised as

$$f_{norm} = (f - r_0)/(r_1 - r_0), \tag{3}$$

meaning that r_0 corresponds to a normalised objective of 0 and r_1 corresponds to a normalised objective of 1, and a higher normalised objective is better. Note that this is only used in Figure 1. This normalisation is possible since all surrogate algorithms start with the same number of random evaluations R, which we omit from the figures. Other visualisation tools that are popular in black-box optimisation, such as ECDF curves, are less suitable for our results since the optimum is unknown for our benchmarks and we look at only one benchmark at a time.

4) Software environment: EXPObench is available as a public github repository[4] and is implemented in the Python programming language. To stimulate future users to add their own problems and approaches to this library, we have taken care to make this as easy as possible and provide documentation to achieve this. We also provide an interface that can easily run one or multiple approaches on a problem in the benchmark suite using the command line interface in `run_experiment.py`. An example is the following code:

```
python run_experiment.py --repetitions=7
--out-path=./results/esp --max-eval=1000
--rand-evals-all=24 esp randomsearch
hyperopt bayesianoptimization
```

This runs random search, HyperOpt and Bayesian optimisation on the ESP problem for 1000 iterations, of which the first 24 iterations are random, repeated seven times, and outputs the results in a certain folder.

B. Benchmark results

We now share the results of applying all algorithms in EXPObench to the different benchmark problems. The IDONE algorithm is only applied to the ESP problem since it does not support continuous variables. To investigate statistical significance of the results, we also report p-values of a pairwise Student's T-test at the last iteration on unnormalised data.

[4]https://github.com/AlgTUDelft/ExpensiveOptimBenchmark

1) Wind Farm Layout: For the wind farm layout optimisation problem, Figure 1a shows the normalised best objective value found at each iteration by the different algorithms, as well as the computation time used by the algorithms at every iteration. All algorithms started with $R = 20$ random samples not shown in the figure. None of the algorithms use more computation time than the expensive objective itself, which took about 15 seconds per function evaluation. While random search is the fastest method, it fails to provide good results, as is expected for a method that does not use any model or heuristic to guide the search. Interestingly, Bayesian optimisation (BO) does not outperform random search on this problem ($p > 0.6$) and is outperformed by all other methods ($p < 0.01$), even though it is designed for problems with continuous variables. In contrast, MVRSM and SMAC both have quite a good performance on this problem while they are designed for problems with mixed variables, though they both do take up more computational resources. DONE, another algorithm designed for continuous problems, performs similar to MVRSM and SMAC ($p > 0.1$).

2) Pitzdaily: Figure 1b shows the results of the Pitzdaily pipe shape optimisation problem with $R = 20$. It can be seen that DONE fails to provide meaningful results. Upon inspection of the proposed candidate solutions, it turns out that the algorithm gets stuck on parts of the search space that violate the constraints. This happens even despite finding feasible solutions early on and despite the penalty for violating the constraints. SMAC, HyperOpt (HO) and MVRSM are the best performing methods on average, outperforming the other three methods ($p < 0.05$) but not each other ($p > 0.6$).

3) ESP: In this discrete problem, algorithms that only deal with continuous variables resort to rounding when calling the expensive objective. This is considered suboptimal in literature, however earlier work shows that this is not necessarily the case for the ESP problem [57]. Indeed, we see in Figure 1c that Bayesian optimisation is the best performing method on this problem, outperforming all methods ($p < 0.03$) except MVRSM and SMAC ($p > 0.2$). This counters the general belief that Bayesian optimisation with Gaussian processes is only adequate on low-dimensional problems with only continuous variables. Another observation is that MVRSM performs much better than IDONE ($p < 0.01$), which fails to significantly outperform random search ($p > 0.9$) even though IDONE is designed for discrete problems. The surrogate algorithms also use less computation time than the expensive objective which took about 28 seconds per iteration to evaluate.

4) XGBoost Hyperparameter Optimisation: Like in the previous benchmark, the algorithms that only deal with continuous variables use rounding for the discrete part of the search space in this problem. For dealing with conditional variables with algorithms that do not support them we use a naive approach: changing such a variable simply has no effect on the objective function when it disappears from the search space, resulting in a larger search space than necessary. Figure 1d shows the results for this benchmark. This time, results are less surprising as SMAC and HyperOpt, two algorithms designed for hyperparameter optimisation with conditional variables, give the best performance. Though they perform similar to
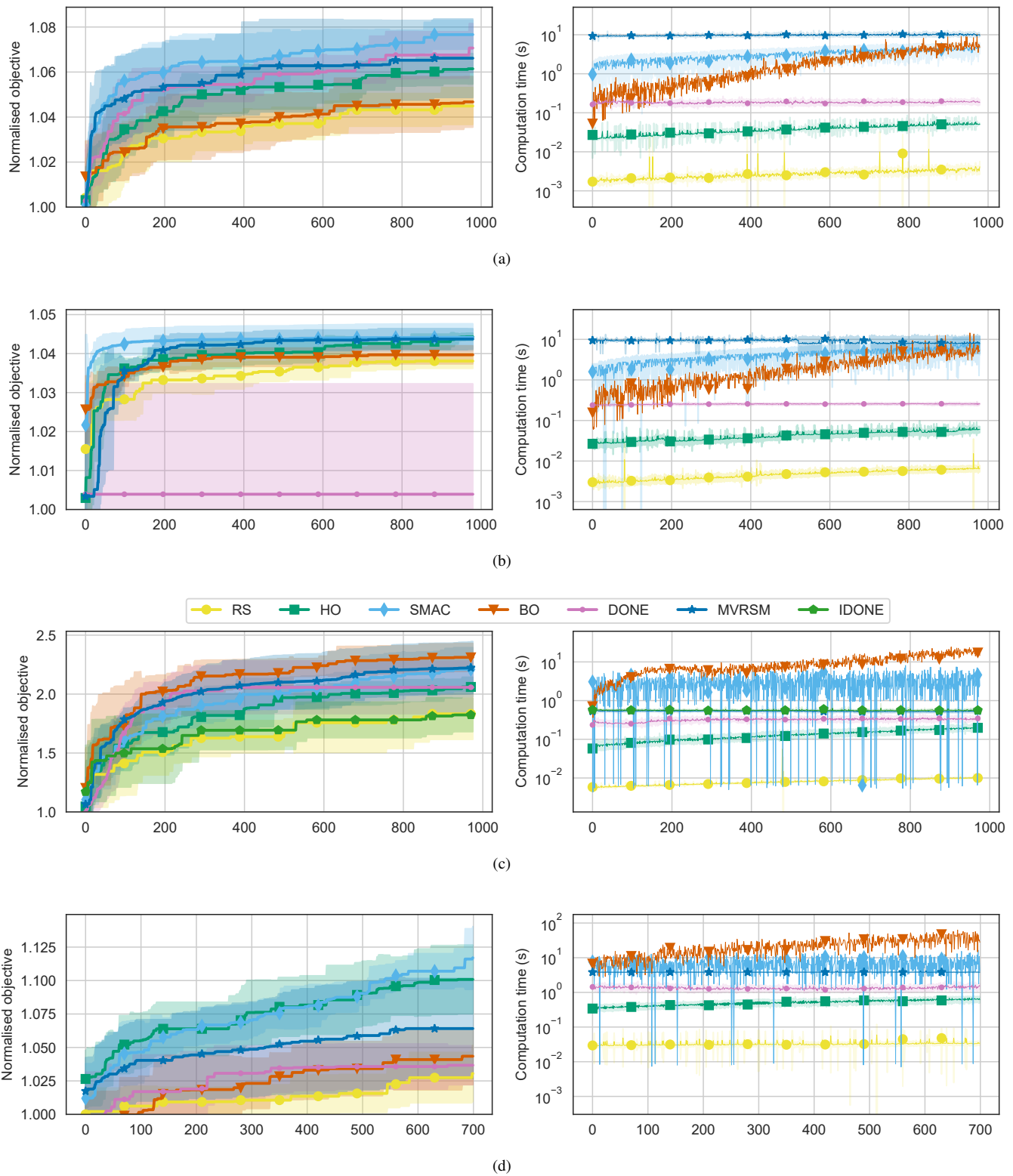
Fig. 1. Results on the different benchmark problems, averaged over T runs, after starting with R random samples. The shaded area indicates one standard deviation, the horizontal axis indicates the iteration of the algorithm, and all figures use the legend shown in the middle. The computation time on the right does not contain the time it takes to evaluate the objective. The benchmark problems are: (a) wind farm layout optimisation, 10 continuous variables, $T = 10$, $R = 20$; (b) Pitzdaily, 10 continuous variables, $T = 5$, $R = 20$; (c) electrostatic precipitator, 49 discrete variables, $T = 7$, $R = 24$; (d) simultaneous hyperparameter tuning and preprocessing for XGBoost, 117 categorical, 7 integer, 11 continuous variables, $T = 10$, $R = 300$.

each other ($p > 0.1$), they outperform all other methods ($p < 0.03$). MVRSM is designed for mixed-variable search spaces like in this problem, but not for conditional variables, and outperforms random search ($p < 0.03$) but not BO and DONE ($p > 0.05$). BO and DONE both fail to outperform random search ($p > 0.1$). If we also consider computation time, HyperOpt appears to be a better choice than SMAC, being faster by more than an order of magnitude.

C. Varying time budget and function evaluation time

In real-life optimisation problems that have limitations on the time to solve the task, the computation time of the optimisation algorithms is important to take into consideration. Hence, we investigate how the algorithms perform with various time budgets and different objective evaluation times. More specifically, instead of restricting the number of evaluations as done up until now, the algorithms are stopped if their runtime exceeds a fixed time budget. The runtime includes both the total function evaluation time as well as the computation time required for the training and acquisition steps of the algorithm.

This experiment extends the results of the benchmark by also putting emphasis on the computation time of the algorithm in addition to their respective sample efficiency. On top of that, it provides information that can be used to decide which algorithm is suitable given a time budget and how expensive the objective function is.

To investigate this in practice, we use the data gathered in the experiments shown in this section by artificially changing the time budget and evaluation time of the expensive objective functions as in earlier work [57]. Because we know the algorithms' computation times from each iteration in the experiments, it is possible to simulate what the total runtime would be if the function evaluation time is adjusted. Then, we report which algorithm returns the best solution when the time budget has been reached for various time budgets and evaluations times.

The evaluation time ranges from 0.12 ms to 36 hours, while the time budget ranges from 0.49 ms to 36 hours. In case the time budget is not reached within the maximum number of iterations that we have observed from the other experiments, for at least one of the algorithms, no results are reported.

Figure 2 displays which algorithm returns the best solution at each problem for a variety of time budgets (x-axis) and function evaluation times (y-axis). Each algorithm has a different marker, and the colour indicates the objective value of the best found solution (without normalisation, so lower is better). As expected, we observe that the objective value decreases when the time budget increases and the evaluation time remains fixed. However, it appears that different algorithms perform well in regions with certain time budgets and evaluation times.

For the Windwake problem we see that BO, HyperOpt, SMAC, DONE and random search all perform the best in different settings. BO seems to perform best when the number of iterations is low no matter the time budget, SMAC performs best for larger time budgets and evaluation times, and random search performs best for low evaluation times. HyperOpt and DONE perform well on semi-expensive objective functions in the $10 - 1000$ ms range.

The observations are similar for the PitzDaily and ESP problems, except that DONE had a poor performance on the PitzDaily problem and SMAC gets outperformed by BO on the ESP problem.

Lastly, for the hyperparameter optimisation problem, it can be seen that HyperOpt is favoured over SMAC due to its computational efficiency, though SMAC performs well with cheaper objective functions. Given a low enough time budget, random search gives the best results, even for expensive objective functions.

D. Offline learning of surrogates

As a final experiment we investigate the choice of surrogate model in the different surrogate algorithms. Though an extensive investigation would require significant adaptations to the algorithms and their implementations, as the choice of surrogate model is heavily intertwined with the choice of the acquisition step in each algorithm, we show how the dataset generated in this work can be used in a simple offline supervised learning framework. This is achieved by training and testing different models on the data and considering the resulting errors, to discover how well different models fit the data for different problem domains.

We limit the scope to the Pitzdaily and ESP problems here, and generate different training sets for each (more data, as well as standard deviations, can be found in the appendix). Each training set consists of the first 500 candidate solutions and objective function values gathered by one run of a specific algorithm, including the first random iterations. We then train a variety of machine learning models on this dataset, with the goal of predicting the (unnormalised) objective function value corresponding to the candidate solution. Using a quadratic loss function, this results in a number of machine learning models equal to the number of algorithms times the number of runs, for each type of machine learning model. The models we used are taken from the Python scikit-learn library [58], and we also add XG-Boost and the piece-wise linear model used by the IDONE and MVRSM algorithms, giving seven models in total: a linear regression model (Linear), polynomial regression with degree 2 (Quadratic), the piece-wise linear model used by MVRSM and IDONE (PWL), a random forest with default hyperparameters (RF), an XGBoost model with default hyperparameters (XGBoost), the Gaussian process used by Bayesian optimisation (GP), and a multi-layer perceptron with default hyperparameters (MLP).

As a test set, for each problem we concatenate all the candidate points and function evaluations that were evaluated by each surrogate algorithm for every run, and keep the 1000 points with the best objective value. As the global optimum is unknown in our benchmark problems, this shows how the different models would perform in good regions of the search space.

If we only train on data gathered with random search we can already see that some models are prone to overfitting. Table III shows the results of training each model on data gathered by random search, averaged over different runs of random search. The models with the smallest training errors
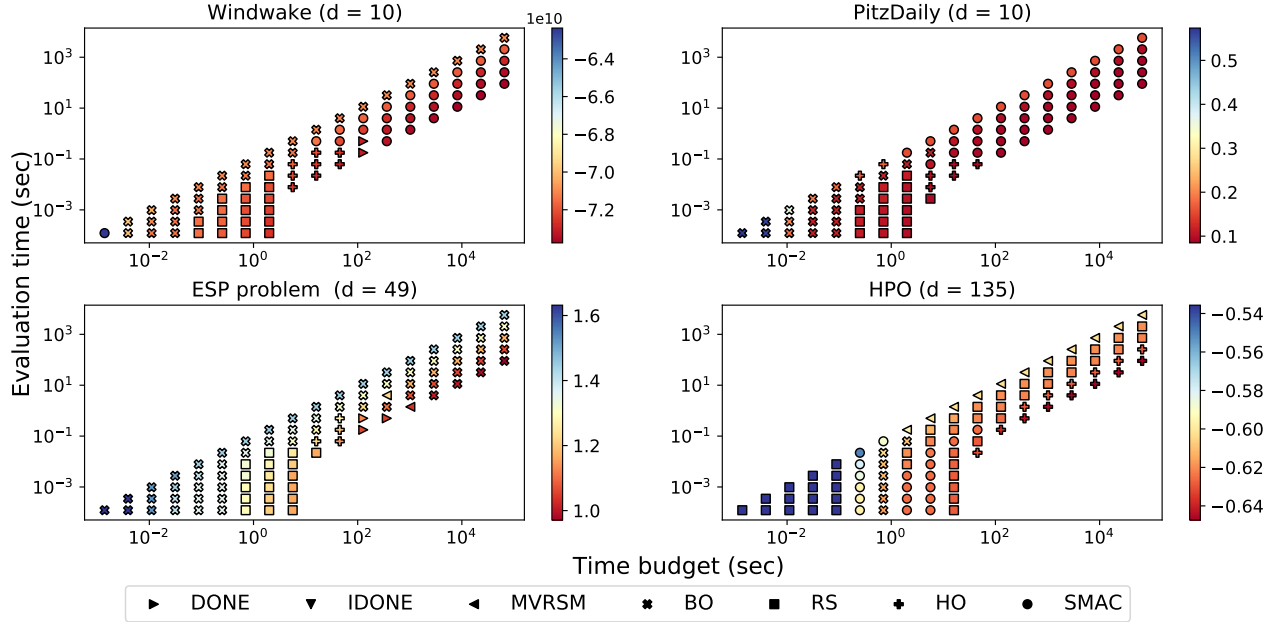
Fig. 2. The best surrogate algorithm for the case that the evaluation time of the objective is artificially changed (vertical axis), and for different time budgets (horizontal axis). The different marker shapes indicate which of the surrogate algorithms achieved the best objective value, while the colour shows the corresponding objective value (not normalised). Other black-box optimisation algorithms such as population-based methods are expected to dominate the empty bottom right region, where the time budget is large but the function evaluation time is small.

are not necessarily the most accurate near the optimum, and may even be outperformed by a simple linear regression model there. For example, the quadratic model has more parameters than training data points for the ESP problem, which causes this expected behavior as it does not make use of regularisation. Furthermore, discrete models such as random forest and XGBoost with their default hyperparameters have a good generalisation performance, not just on the discrete ESP problem but also on the continuous Pitzdaily problem, even though their training error is a bit higher than that of other models.

If we train models on data gathered by a surrogate algorithm that uses that model or an approximation thereof, we get the results shown in Table IV. The piece-wise linear model (PWL) used for training is exactly the same surrogate model used by the IDONE and MVRSM algorithms, and the Gaussian process (GP) is exactly the same as the one used by Bayesian optimisation (BO). The random forest (RF) is not exactly the one used by SMAC, but uses default hyperparameters, and the model used by DONE is only an approximation of a Gaussian process. We see that the training error on data gathered by DONE can get very low, but this does not mean that DONE is a good surrogate algorithm, as we saw it perform very poorly on the Pitzdaily problem. A likely explanation is that the acquisition is not leading to the right data points. More interesting are the test errors: though a Gaussian process trained on data gathered by a surrogate algorithm that uses this model (BO) receives a low test error, an XGBoost model trained on data gathered by random search can get an even lower test error as seen in Table III. The test error for XGBoost trained on data gathered by BO, not shown in these tables,

TABLE III
MEAN AVERAGE ERROR FOR MODELS TRAINED ON DATA GATHERED BY RANDOM SEARCH, AVERAGED OVER DIFFERENT RANDOM SEARCH RUNS.

Benchmark	Pitzdaily		ESP	
Model	Train	Test	Train	Test
Linear	0.697	1.229	2.651	1.808
Quadratic	0.504	1.218	$3 \cdot 10^{-14}$	4.400
PWL	0.006	1.323	$5 \cdot 10^{-9}$	8.869
RF	0.151	**0.758**	0.574	0.972
XGBoost	0.279	0.849	0.153	**0.711**
GP	$8 \cdot 10^{-7}$	1.147	$1 \cdot 10^{-6}$	1.297
MLP	0.733	1.364	0.420	0.906

TABLE IV
MEAN AVERAGE ERROR FOR MODELS TRAINED ON DATA GATHERED BY A SURROGATE ALGORITHM THAT USES THAT MODEL, AVERAGED OVER DIFFERENT RUNS.

Benchmark	Pitzdaily		ESP	
Model	Train	Test	Train	Test
PWL on IDONE	-	-	$2 \cdot 10^{-4}$	11.16
PWL on MVRSM	0.047	4.092	0.002	11.58
RF on SMAC	0.101	1.151	0.148	0.855
GP on BO	$9 \cdot 10^{-7}$	**0.915**	$5.5 \cdot 10^{-7}$	**0.835**
GP on DONE	$5 \cdot 10^{-8}$	1.888	$4.8 \cdot 10^{-7}$	0.910

is 0.997 for the Pitzdaily problem and 0.701 for the ESP problem. Lastly, surrogate models do not always achieve a lower training error on data gathered by a method that uses that model than on data gathered by random search.

V. Discussion

Based on the observations of the previous section, we highlight the most important insights that were obtained. First of all, the *type of variable* a surrogate model is designed for, is not necessarily a good indicator of the performance of the surrogate algorithm in case of a real-life problem: discrete surrogates can perform well on continuous problems, and vice versa. We saw this on the wind farm layout optimisation problem, a continuous problem where a discrete surrogate model (SMAC's random forest) had the best performance, and on the ESP problem, a discrete problem where the continuous Gaussian process surrogate model had the best performance even though it was unable to outperform random search on the wind farm layout optimisation problem. Part of these insights were known from previous work [57], but we extended these insights to continuous problems and to more benchmark problems and surrogate algorithms. The experiments using offline learning of machine learning models also showed that discrete models such as XGBoost can have lower generalisation error than continuous models, even on data coming from a continuous problem like Pitzdaily.

Second, our observations lead us to believe that *exploration is more important than model accuracy* in surrogate algorithms. The offline learning experiments showed that surrogate models trained on data gathered by an algorithm that uses that model are not necessarily more accurate than surrogate models trained on data gathered by random search, a high-exploration method. The use of random search should also not be underestimated, as the experiment where we artificially change the evaluation time of the objective shows that for all considered benchmarks there are situations where random search outperforms all surrogate algorithms, mainly when the objective evaluation time is low. Furthermore, on the ESP problem, MVRSM had a much better performance than IDONE, even though they use exactly the same piece-wise linear surrogate model on that problem. The only difference between the two algorithms is that MVRSM has a higher exploration rate. The low training error of the piece-wise linear surrogate model shows that a highly accurate model does not necessarily lead to a better performance of the surrogate algorithm using that model.

Finally, *the available time budget and the evaluation time of the objective* strongly influence which algorithm is the best choice for a certain problem. This can be seen from the experiment where we artificially change the function evaluation time, where the best performing algorithm changes depending on the available time budget and function evaluation time. Given that most algorithms had a lower computation time than the function evaluation time in our experiments, there is room for improvement to spend resources on improving the exploration strategy or on using an ensemble method. Using an ensemble of surrogate models or acquisition strategies should also help with robustness, as many algorithms failed to outperform random search for at least one problem. Alternatively, speeding up the surrogate algorithms should make them more suitable for less expensive problems than the benchmarks in this paper. Having a fixed computation time for every iteration, which is the case for the DONE, IDONE and MVRSM algorithms, should help in this aspect.

VI. Conclusion and future Work

We have benchmarked six surrogate algorithm on four real-life expensive benchmark problems, which gave rise to:

- a public benchmark library called EXPObench that contains real-life expensive benchmark functions and baseline algorithms that can solve these benchmarks;
- insights into how different aspects of the problems and algorithms influence the algorithm performance;
- a public dataset containing the algorithm performance on these benchmarks.

The benchmark library fills an important gap in the current landscape of optimisation benchmark libraries, that mostly consists of cheap to evaluate benchmark functions or of expensive problems with no or limited baseline solutions from surrogate model literature. A first analysis of the dataset showed how the best choice of algorithm for a certain problem depends on the available time budget and the evaluation time of the objective, and we provided a method to extrapolate such results to real-life problems that contain expensive objective functions with different costs. The dataset also allowed us to train surrogate models offline rather than online, giving insight into the generalisation capabilities of the surrogate models and showing the potential of models such as XGBoost to be used in new surrogate algorithms in the future. Finally, we showed how continuous models can work well for discrete problems and vice versa, and we highlighted the important role of exploration in surrogate algorithms. In future work we will focus on methods that can deal with the constraints present in some of the benchmark problems from this work, as well as make a comparison with surrogate-assisted evolutionary methods.

Appendix
Offline learning results

Here we show more data of the offline learning experiment presented in Section IV-D, including the standard deviations. See Table V. No results are given for the hyperparameter optimisation benchmark, as not all supervised learning models are able to deal with the conditional variables present in this benchmark.

TABLE V

Mean average error for models trained on data gathered by one surrogate algorithm, averaged over different runs.

Benchmark	Wind farm layout		Pitzdaily		ESP	
Model+method	Train	Test	Train	Test	Train	Test
Linear on RS	$3.3 \cdot 10^{10} \pm 3 \cdot 10^8$	$3.7 \cdot 10^{10} \pm 3 \cdot 10^9$	0.70 ± 0.03	1.23 ± 0.15	2.7 ± 6.0	1.8 ± 2.3
Quadratic on RS	$2.8 \cdot 10^{10} \pm 6 \cdot 10^8$	$\mathbf{1.9 \cdot 10^{10}} \pm 5 \cdot 10^9$	0.50 ± 0.01	1.22 ± 0.13	$\mathbf{3 \cdot 10^{-14}} \pm 8 \cdot 10^{-14}$	4.4 ± 8.8
PWL on RS	$3.1 \cdot 10^{10} \pm 1 \cdot 10^9$	$2.0 \cdot 10^{10} \pm 4 \cdot 10^9$	$6 \cdot 10^{-3} \pm 6 \cdot 10^{-4}$	1.32 ± 0.44	$5 \cdot 10^{-9} \pm 1 \cdot 10^{-8}$	8.9 ± 16.9
RF on RS	$1.2 \cdot 10^{10} \pm 2 \cdot 10^8$	$3.5 \cdot 10^{10} \pm 2 \cdot 10^9$	0.15 ± 0.01	$\mathbf{0.76} \pm 0.17$	0.6 ± 1.2	1.0 ± 0.4
XGBoost on RS	$1.6 \cdot 10^{10} \pm 1 \cdot 10^9$	$3.7 \cdot 10^{10} \pm 4 \cdot 10^9$	0.28 ± 0.01	0.85 ± 0.23	0.2 ± 0.2	$\mathbf{0.7} \pm 0.1$
GP on RS	$\mathbf{3 \cdot 10^4} \pm 1 \cdot 10^2$	$3.7 \cdot 10^{10} \pm 2 \cdot 10^9$	$\mathbf{8 \cdot 10^{-7}} \pm 3 \cdot 10^{-8}$	1.15 ± 0.09	$1 \cdot 10^{-6} \pm 2 \cdot 10^{-6}$	1.3 ± 1.0
MLP on RS	$3.5 \cdot 10^{10} \pm 2 \cdot 10^9$	$7.3 \cdot 10^{10} \pm 2 \cdot 10^2$	0.73 ± 0.03	1.36 ± 0.02	0.4 ± 0.9	0.9 ± 0.6
Linear on BO	$3.2 \cdot 10^{10} \pm 2 \cdot 10^9$	$3.7 \cdot 10^{10} \pm 5 \cdot 10^9$	0.77 ± 0.04	0.95 ± 0.35	$0.2 \pm 5 \cdot 10^{-2}$	1.2 ± 0.2
Quadratic on BO	$2.6 \cdot 10^{10} \pm 2 \cdot 10^9$	$2.4 \cdot 10^{10} \pm 1.0 \cdot 10^{10}$	0.53 ± 0.15	1.34 ± 0.21	$\mathbf{5 \cdot 10^{-15}} \pm 3 \cdot 10^{-15}$	2.0 ± 2.0
PWL on BO	$2.9 \cdot 10^{10} \pm 2 \cdot 10^9$	$\mathbf{2.0 \cdot 10^{10}} \pm 3 \cdot 10^9$	$0.01 \pm 2 \cdot 10^{-3}$	1.80 ± 0.93	$1 \cdot 10^{-9} \pm 7 \cdot 10^{-10}$	4.0 ± 2.1
RF on BO	$1.1 \cdot 10^{10} \pm 1 \cdot 10^9$	$3.7 \cdot 10^{10} \pm 3 \cdot 10^9$	0.15 ± 0.01	1.05 ± 0.15	$7 \cdot 10^{-2} \pm 2 \cdot 10^{-2}$	1.0 ± 0.3
XGBoost on BO	$1.5 \cdot 10^{10} \pm 2 \cdot 10^9$	$4.1 \cdot 10^{10} \pm 9 \cdot 10^9$	0.24 ± 0.02	1.00 ± 0.28	$8 \cdot 10^{-2} \pm 7 \cdot 10^{-3}$	$\mathbf{0.7} \pm 9 \cdot 10^{-2}$
GP on BO	$\mathbf{3 \cdot 10^4} \pm 1 \cdot 10^3$	$3.4 \cdot 10^{10} \pm 5 \cdot 10^9$	$\mathbf{9 \cdot 10^{-7}} \pm 2 \cdot 10^{-8}$	$\mathbf{0.92} \pm 0.05$	$5 \cdot 10^{-7} \pm 3 \cdot 10^{-7}$	0.8 ± 0.2
MLP on BO	$3.9 \cdot 10^{10} \pm 5 \cdot 10^9$	$7.3 \cdot 10^{10} \pm 1 \cdot 10^2$	0.68 ± 0.04	1.10 ± 0.28	$0.1 \pm 5 \cdot 10^{-2}$	1.3 ± 0.5
Linear on HO	$2.6 \cdot 10^{10} \pm 2 \cdot 10^9$	$4.6 \cdot 10^{10} \pm 5 \cdot 10^9$	0.77 ± 0.04	0.95 ± 0.35	0.2 ± 0.1	0.8 ± 0.2
Quadratic on HO	$2.2 \cdot 10^{10} \pm 2 \cdot 10^9$	$3.7 \cdot 10^{10} \pm 9 \cdot 10^9$	0.53 ± 0.15	1.34 ± 0.21	$2 \cdot 10^{-3} \pm 2 \cdot 10^{-3}$	$3 \cdot 10^9 \pm 6 \cdot 10^9$
PWL on HO	$2.5 \cdot 10^{10} \pm 2 \cdot 10^9$	$2.9 \cdot 10^{10} \pm 9 \cdot 10^9$	$0.01 \pm 2 \cdot 10^{-3}$	1.80 ± 0.93	$3 \cdot 10^{-5} \pm 6 \cdot 10^{-5}$	4.3 ± 3.4
RF on HO	$1.0 \cdot 10^{10} \pm 8 \cdot 10^8$	$3.6 \cdot 10^{10} \pm 2 \cdot 10^9$	0.15 ± 0.01	1.05 ± 0.15	$6 \cdot 10^{-2} \pm 3 \cdot 10^{-2}$	$0.7 \pm 5 \cdot 10^{-2}$
XGBoost on HO	$1.1 \cdot 10^{10} \pm 2 \cdot 10^9$	$4.3 \cdot 10^{10} \pm 5 \cdot 10^9$	0.24 ± 0.02	1.00 ± 0.28	$7 \cdot 10^{-2} \pm 10^{-2}$	$\mathbf{0.6} \pm 8 \cdot 10^{-2}$
GP on HO	$\mathbf{3 \cdot 10^4} \pm 3 \cdot 10^3$	$\mathbf{2.5 \cdot 10^{10}} \pm 2 \cdot 10^9$	$\mathbf{9 \cdot 10^{-7}} \pm 2 \cdot 10^{-8}$	$\mathbf{0.92} \pm 0.05$	$3 \cdot 10^{-7} \pm 2 \cdot 10^{-7}$	0.7 ± 0.1
MLP on HO	$4.8 \cdot 10^{10} \pm 3 \cdot 10^9$	$7.3 \cdot 10^{10} \pm 1 \cdot 10^2$	0.68 ± 0.04	1.10 ± 0.28	$6 \cdot 10^{-2} \pm 10^{-2}$	0.7 ± 0.1
Linear on SMAC	$1.2 \cdot 10^{10} \pm 1 \cdot 10^9$	$3.0 \cdot 10^{10} \pm 4 \cdot 10^9$	0.47 ± 0.10	0.83 ± 0.48	0.4 ± 0.3	1.0 ± 0.4
Quadratic on SMAC	$9 \cdot 10^9 \pm 1 \cdot 10^9$	$5.0 \cdot 10^{10} \pm 1.8 \cdot 10^{10}$	0.39 ± 0.08	1.24 ± 0.82	$\mathbf{6 \cdot 10^{-15}} \pm 5 \cdot 10^{-15}$	0.9 ± 0.4
PWL on SMAC	$1.2 \cdot 10^{10} \pm 1 \cdot 10^9$	$1.8 \cdot 10^{10} \pm 4 \cdot 10^9$	0.09 ± 0.03	2.91 ± 0.91	$1 \cdot 10^{-9} \pm 5 \cdot 10^{-10}$	2.5 ± 1.3
RF on SMAC	$4 \cdot 10^9 \pm 4 \cdot 10^8$	$3.4 \cdot 10^{10} \pm 3 \cdot 10^9$	0.10 ± 0.02	1.15 ± 0.27	0.1 ± 0.1	0.9 ± 0.1
XGBoost on SMAC	$4 \cdot 10^9 \pm 5 \cdot 10^8$	$4.4 \cdot 10^{10} \pm 8 \cdot 10^9$	0.11 ± 0.03	1.11 ± 0.31	$0.1 \pm 3 \cdot 10^{-2}$	$\mathbf{0.7} \pm 6 \cdot 10^{-2}$
GP on SMAC	$\mathbf{8 \cdot 10^4} \pm 5 \cdot 10^4$	$\mathbf{1.3 \cdot 10^{10}} \pm 1 \cdot 10^9$	$\mathbf{9 \cdot 10^{-7}} \pm 3 \cdot 10^{-7}$	$\mathbf{0.44} \pm 0.09$	$3 \cdot 10^{-7} \pm 2 \cdot 10^{-7}$	0.8 ± 0.1
MLP on SMAC	$6.0 \cdot 10^{10} \pm 1 \cdot 10^9$	$7.3 \cdot 10^{10} \pm 2 \cdot 10^2$	0.46 ± 0.08	0.83 ± 0.45	$6 \cdot 10^{-2} \pm 3 \cdot 10^{-2}$	0.9 ± 0.4
Linear on DONE	$2.6 \cdot 10^{10} \pm 10^9$	$5.5 \cdot 10^{10} \pm 3 \cdot 10^9$	0.04 ± 0.01	1.90 ± 0.008	$0.2 \pm 2 \cdot 10^{-2}$	0.9 ± 0.1
Quadratic on DONE	$2.3 \cdot 10^{10} \pm 10^9$	$5.8 \cdot 10^{10} \pm 2 \cdot 10^9$	0.08 ± 0.02	$\mathbf{1.55} \pm 0.13$	$\mathbf{4 \cdot 10^{-15}} \pm 2 \cdot 10^{-15}$	1.0 ± 0.2
PWL on DONE	$2.5 \cdot 10^{10} \pm 10^9$	$5.5 \cdot 10^{10} \pm 2 \cdot 10^9$	$6 \cdot 10^{-4} \pm 10^{-4}$	1.78 ± 0.10	$8 \cdot 10^{-10} \pm 9 \cdot 10^{-11}$	2.1 ± 1.0
RF on DONE	$9 \cdot 10^9 \pm 4 \cdot 10^8$	$\mathbf{5.2 \cdot 10^{10}} \pm 3 \cdot 10^9$	$0.01 \pm 3 \cdot 10^{-3}$	1.70 ± 0.09	$5 \cdot 10^{-2} \pm 9 \cdot 10^{-3}$	$0.8 \pm 6 \cdot 10^{-2}$
XGBoost on DONE	$1.0 \cdot 10^{10} \pm 8 \cdot 10^8$	$5.4 \cdot 10^{10} \pm 5 \cdot 10^9$	$0.05 \pm 2 \cdot 10^{-3}$	1.60 ± 0.14	$6 \cdot 10^{-2} \pm 4 \cdot 10^{-3}$	$\mathbf{0.7} \pm 6 \cdot 10^{-2}$
GP on DONE	$\mathbf{3 \cdot 10^4} \pm 10^3$	$5.5 \cdot 10^{10} \pm 2 \cdot 10^9$	$\mathbf{5 \cdot 10^{-8}} \pm 2 \cdot 10^{-8}$	1.89 ± 0.07	$5 \cdot 10^{-7} \pm 2 \cdot 10^{-7}$	$0.9 \pm 4 \cdot 10^{-2}$
MLP on DONE	$1.8 \cdot 10^{10} \pm 10^9$	$7.3 \cdot 10^{10} \pm 2 \cdot 10^2$	$0.08 \pm 5 \cdot 10^{-3}$	1.91 ± 0.009	$0.1 \pm 7 \cdot 10^{-2}$	0.8 ± 0.2
Linear on IDONE	-	-	-	-	$0.2 \pm 3 \cdot 10^{-2}$	1.0 ± 0.2
Quadratic on IDONE	-	-	-	-	$\mathbf{4 \cdot 10^{-15}} \pm 3 \cdot 10^{-15}$	1.6 ± 1.3
PWL on IDONE	-	-	-	-	$2 \cdot 10^{-4} \pm 3 \cdot 10^{-4}$	11.2 ± 2.6
RF on IDONE	-	-	-	-	$4 \cdot 10^{-2} \pm 9 \cdot 10^{-3}$	1.2 ± 0.5
XGBoost on IDONE	-	-	-	-	$6 \cdot 10^{-2} \pm 10^{-2}$	$\mathbf{0.8} \pm 0.1$
GP on IDONE	-	-	-	-	$6 \cdot 10^{-7} \pm 4 \cdot 10^{-7}$	0.9 ± 0.1
MLP on IDONE	-	-	-	-	$8 \cdot 10^{-2} \pm 4 \cdot 10^{-2}$	1.0 ± 0.4
Linear on MVRSM	$7 \cdot 10^9 \pm 2 \cdot 10^9$	$4.5 \cdot 10^{10} \pm 1.1 \cdot 10^{10}$	0.48 ± 0.07	1.16 ± 0.35	$0.1 \pm 3 \cdot 10^{-2}$	0.8 ± 0.2
Quadratic on MVRSM	$8 \cdot 10^9 \pm 4 \cdot 10^9$	$1.75 \cdot 10^{11} \pm 6.1 \cdot 10^{10}$	0.40 ± 0.15	0.96 ± 0.96	$2 \cdot 10^{-4} \pm 3 \cdot 10^{-4}$	$5 \cdot 10^7 \pm 5 \cdot 10^7$
PWL on MVRSM	$7 \cdot 10^9 \pm 2 \cdot 10^9$	$5.2 \cdot 10^{10} \pm 1.3 \cdot 10^{10}$	0.05 ± 0.02	4.09 ± 1.90	$2 \cdot 10^{-3} \pm 3 \cdot 10^{-3}$	11.6 ± 2.8
RF on MVRSM	$2 \cdot 10^9 \pm 5 \cdot 10^8$	$4.4 \cdot 10^{10} \pm 6 \cdot 10^9$	0.06 ± 10^{-3}	1.21 ± 0.29	$3 \cdot 10^{-2} \pm 7 \cdot 10^{-3}$	0.8 ± 0.2
XGBoost on MVRSM	$2 \cdot 10^9 \pm 2 \cdot 10^8$	$5.1 \cdot 10^{10} \pm 1.0 \cdot 10^{10}$	$0.06 \pm 7 \cdot 10^{-3}$	1.16 ± 0.40	$4 \cdot 10^{-2} \pm 10^{-2}$	0.7 ± 0.3
GP on MVRSM	$\mathbf{2 \cdot 10^5} \pm 2 \cdot 10^5$	$\mathbf{1.3 \cdot 10^{10}} \pm 6 \cdot 10^9$	$\mathbf{8 \cdot 10^{-7}} \pm 2 \cdot 10^{-7}$	$\mathbf{0.53} \pm 0.07$	$\mathbf{5 \cdot 10^{-7}} \pm 3 \cdot 10^{-7}$	$\mathbf{0.5} \pm 0.2$
MLP on MVRSM	$6.4 \cdot 10^{10} \pm 3 \cdot 10^9$	$7.3 \cdot 10^{10} \pm 1 \cdot 10^2$	0.37 ± 0.09	1.18 ± 0.34	$9 \cdot 10^{-2} \pm 3 \cdot 10^{-2}$	0.7 ± 0.3

REFERENCES

[1] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. D. Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, pp. 148–175, 2016.

[2] Y. Zhou, Y. Koyama, M. Goto, and T. Igarashi, "Generative melody composition with human-in-the-loop Bayesian optimization," 2020.

[3] H. Q. Liang, A. E. Gongora, Z. Ren, Z. Liu, A. Tiihonen, S. Sun, F. Mekki-Berrada, S. A. Khan, D. Bash, K. Hippalgaonkar, J. R. Deneault, B. Maruyama, K. Brown, J. Fisher, and T. Buonassisi, "Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains," in *Advances in Neural Information Processing Systems*, 2020.

[4] M. Fiducioso, S. Curi, B. Schumacher, M. Gwerder, and A. Krause, "Safe contextual Bayesian optimization for sustainable room temperature PID control tuning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 7 2019, pp. 5850–5856. [Online]. Available: https://doi.org/10.24963/ijcai.2019/811

[5] L. Bliek, H. R. G. W. Verstraete, M. Verhaegen, and S. Wahls, "Online optimization with costly and noisy measurements using random Fourier expansions," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 1, pp. 167–182, 2018.

[6] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*, 2013, pp. 115–123.

[7] J. Močkus, "On Bayesian methods for seeking the extremum," in *Optimization techniques IFIP technical conference*. Springer, 1975, pp. 400–404.

[8] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[9] L. Bliek, S. Verwer, and M. de Weerdt, "Black-box combinatorial optimization using models with integer-valued minima," *Annals of Mathematics and Artificial Intelligence*, pp. 1–15, 2020.

[10] ——, "Black-box mixed-variable optimisation using a surrogate model that satisfies integer constraints," *arXiv preprint arXiv:2006.04508*, 2020.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration (extended version)," *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*, 2010.

[12] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014–. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[13] B. Ru, A. Alvi, V. Nguyen, M. A. Osborne, and S. Roberts, "Bayesian optimisation over multiple continuous and categorical inputs," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 8276–8285. [Online]. Available: http://proceedings.mlr.press/v119/ru20a.html

[14] K. van der Blom, T. M. Deist, V. Volz, M. Marchi, Y. Nojima, B. Naujoks, A. Oyama, and T. Tušar, "Identifying properties of real-world optimisation problems through a questionnaire," *arXiv preprint arXiv:2011.05547*, 2020. [Online]. Available: https://arxiv.org/abs/2011.05547

[15] N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger, "COmparing Continuous Optimizers: numbbo/COCO on Github," Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2594848

[16] P. S. Palar, R. P. Liem, L. R. Zuhal, and K. Shimoyama, "On the use of surrogate models in engineering design optimization and exploration: The key issues," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1592–1602. [Online]. Available: https://doi.org/10.1145/3319619.3326813

[17] S. J. Daniels, A. A. Rahat, R. M. Everson, G. R. Tabor, and J. E. Fieldsend, "A suite of computationally expensive shape optimisation problems using computational fluid dynamics," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 296–307.

[18] V. Volz and B. Naujoks, "On benchmarking surrogate-assisted evolutionary algorithms," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019.

[19] A. Tangherloni, S. Spolaor, P. Cazzaniga, D. Besozzi, L. Rundo, G. Mauri, and M. S. Nobile, "Biochemical parameter estimation vs. benchmark functions: A comparative study of optimization performance and representation design," *Appl. Soft Comput.*, vol. 81, 2019.

[20] J. M. Dieterich and B. Hartke, "Empirical review of standard benchmark functions using evolutionary global optimization," *arXiv preprint arXiv:1207.4318*, 2012.

[21] C. Doerr, J. Dréo, and P. Kerschke, "Making a case for (hyper-)parameter tuning as benchmark problems," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019.

[22] T. Bartz-Beielstein and M. Zaefferer, "Model-based methods for continuous and discrete global optimization," *Applied Soft Computing*, vol. 55, pp. 154–167, 2017.

[23] S. Wagner and M. Affenzeller, "HeuristicLab: A generic and extensible optimization environment," in *Adaptive and Natural Computing Algorithms*, B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds. Vienna: Springer Vienna, 2005, pp. 538–541.

[24] J. Humeau, A. Liefooghe, E. Talbi, and S. Vérel, "ParadisEO-MO: from fitness landscape analysis to efficient local search algorithms," *Journal of Heuristics*, vol. 19, pp. 881–915, 2013.

[25] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic *et al.*, "HyFlex: A benchmark framework for cross-domain heuristic search," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2012, pp. 136–147.

[26] F. Caraffini and G. Iacca, "The SOS platform: Designing, tuning and statistically benchmarking optimisation algorithms," *Mathematics*, vol. 8, no. 5, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/5/785

[27] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck, "IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv e-prints:1810.05281*, Oct. 2018. [Online]. Available: https://arxiv.org/abs/1810.05281

[28] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[29] J. Liu, Z.-H. Han, and W. Song, "Comparison of infill sampling criteria in kriging-based aerodynamic optimization," *28th Congress of the International Council of the Aeronautical Sciences 2012, ICAS 2012*, vol. 2, pp. 1625–1634, 01 2012.

[30] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren, "An open source AutoML benchmark," *arXiv preprint arXiv:1907.00909 [cs.LG]*, 2019, accepted at AutoML Workshop at ICML 2019. [Online]. Available: https://arxiv.org/abs/1907.00909

[31] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *ICML*, 2019.

[32] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," *ArXiv*, vol. abs/2001.00326, 2020.

[33] J. N. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search," *ArXiv*, vol. abs/2008.09777, 2020.

[34] V. Volz, B. Naujoks, P. Kerschke, and T. Tusar, "Single- and multi-objective game-benchmark for evolutionary algorithms," *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[35] S. J. Daniels, A. A. M. Rahat, R. Everson, G. R. Tabor, and J. Fieldsend, "A suite of computationally expensive shape optimisation problems using computational fluid dynamics," in *PPSN*, 2018.

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

[37] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 1436–1445.

[38] Z. hua Han, S. Görtz, and R. Zimmermann, "Improving variable-fidelity surrogate modeling via gradient-enhanced kriging and a generalized hybrid bridge function," *Aerospace Science and Technology*, vol. 25, pp. 177–189, 2013.

[39] M. A. Bouhlel, J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. R. A. Martins, "A Python surrogate modeling framework with derivatives," *Advances in Engineering Software*, p. 102662, 2019.

[40] D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq, "A surrogate modeling and adaptive sampling toolbox for computer based design," *J. Mach. Learn. Res.*, vol. 11, pp. 2051–2055, 2010.

[41] T. Pourmohamad, "CompModels: A suite of computer model test functions for Bayesian optimization," *arXiv: Computation*, 2020.

[42] T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer, "Dacbench: A benchmark library for dynamic algorithm configuration," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*. ijcai.org, Aug. 2021.

[43] A. Costa and G. Nannicini, "RBFOpt: an open-source library for black-box optimization with costly function evaluations," *Mathematical Programming Computation*, vol. 10, pp. 597–629, 2018.

[44] K. Eggensperger, M. Feurer, A. Klein, and S. Falkner, "HPObench," https://github.com/automl/HPOBench, 2016.

[45] R. Turner and D. Eriksson, "BayesMark," https://github.com/uber/bayesmark, 2018.

[46] J. Mueller, "MATSuMoTo," https://github.com/Piiloblondie/MATSuMoTo, 2014.

[47] M. Lindauer, "AClib2," https://bitbucket.org/mlindauer/aclib2, 2016.

[48] D. Eriksson, D. Bindel, and C. A. Shoemaker, "pySOT and POAP: An event-driven asynchronous framework for surrogate optimization," *arXiv preprint arXiv:1908.00420*, 2019.

[49] NREL, "FLORIS. Version 2.1.1," 2020. [Online]. Available: https://github.com/NREL/floris

[50] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf, "Solving the stochastic time-dependent orienteering problem with time windows," *European Journal of Operational Research*, vol. 255, no. 3, pp. 699–718, 2016. [Online]. Available: https://lirias.kuleuven.be/retrieve/397154

[51] R. D. Pasquale and J. Marenco, "Machine learning for improving performance in an evolutionary algorithm for minimum path with uncertain costs given by massively simulated scenarios," *ArXiv*, vol. abs/2102.01830, 2021.

[52] F. Rehbach, M. Zaefferer, J. Stork, and T. Bartz-Beielstein, "Comparison of parallel surrogate-assisted optimization approaches," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 1348–1355.

[53] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *ACM SIGKDD*, 2016, pp. 785–794.

[54] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[55] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, "Towards objective measures of algorithm performance across instance space," *Computers & Operations Research*, vol. 45, pp. 12–24, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054813003389

[56] S. Thrun and L. Pratt, *Learning to Learn: Introduction and Overview*. Boston, MA: Springer US, 1998, pp. 3–17. [Online]. Available: https://doi.org/10.1007/978-1-4615-5529-2_1

[57] R. Karlsson, L. Bliek, S. Verwer, and M. de Weerdt, "Continuous surrogate-based optimization algorithms are well-suited for expensive discrete problems," in *Proceedings of the Benelux Conference on Artificial Intelligence*, 2020, pp. 88–102.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.