Benchmarking surrogate-based optimisation algorithms on expensive black-box functions

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

DOI
10.1016/j.asoc.2023.110744

Publication date
2023

Document Version
Final published version

Published in
Applied Soft Computing

Citation (APA)
Bliek, L., Guijt, A., Karlsson, R., Verwer, S., & de Weerdt, M. (2023). Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *Applied Soft Computing*, *147*, Article 110744. https://doi.org/10.1016/j.asoc.2023.110744

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Applied Soft Computing 147 (2023) 110744

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Benchmarking surrogate-based optimisation algorithms on expensive black-box functions

Laurens Bliek [a,*], Arthur Guijt [b], Rickard Karlsson [b], Sicco Verwer [b], Mathijs de Weerdt [b]

[a] School of Industrial Engineering, Eindhoven University of Technology, PO Box 513, 5600 MB, Eindhoven, The Netherlands
[b] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, PO Box 5, 2600 AA, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 5 December 2022
Received in revised form 15 June 2023
Accepted 3 August 2023
Available online 15 August 2023

Dataset link: https://doi.org/10.4121/14247179.v2

Keywords:
Expensive optimisation
Surrogate-based optimisation
Bayesian optimisation
Benchmarking

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 and to give substantial advice on which method to use when. 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. We also provide rules of thumb for which surrogate algorithm to use in which situation. 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 results of the six algorithms on all evaluated problem instances. This unique new dataset lowers the bar for researching new methods as the number of expensive evaluations required for comparison and for the creation of new surrogate models is significantly reduced.

1. 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 materials science [2], temperature control [3], building design [4], aerodynamics [5], optics [6], and computer vision [7], as well as many sustainability-related applications [8,9]. Many of these problems contain a large number of non-linear complex dependencies or constraints. Furthermore, establishing the correctness and quality of candidate solutions requires computationally expensive simulators or algorithms. Maximising the power output of a solar panel or wind turbine, for instance, requires running a physics simulator every time a new configuration is tried. Surrogate models can help to reduce the number of runs for these expensive simulators or algorithms.

On the one hand, the training and usage of a surrogate model that approximates the objective function is typically more computationally intensive than the use of black-box optimisation heuristics such as local search or population-based methods. On the other hand, by making use of a surrogate model, surrogate-based algorithms achieve good results even with a low number of function evaluations, which is crucial when dealing with expensive objective functions. What is missing is a good understanding of when to use surrogate-based optimisation, and which model and algorithm to use then. Ultimately, this should come in the form of guidelines that, given the properties of an expensive optimisation problem, tell which type of algorithm would be the most suitable.

The current way of benchmarking surrogate algorithms does not give complete insight into the strengths and weaknesses of the different algorithms, such as their computational efficiency or accuracy for different types of problems. The most important reason for this is the lack of a standard benchmark set of problems that come from real-life applications and that also have expensive objective functions. As identified by Palar et al. [10]: "*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. It is therefore imperative to establish a library of benchmarking problems based on real-world problems that are accessible to researchers.*" This lack of a real-world benchmarking library also

* Corresponding author.
E-mail address: l.bliek@tue.nl (L. Bliek).

https://doi.org/10.1016/j.asoc.2023.110744

makes it difficult to further develop our understanding of which surrogate algorithm to use on what kind of real-life applications.

In this work, we aim to tackle this gap by comparing 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 which includes for several real-life applications the candidate solutions chosen by each algorithm, the resulting objective value, and the computation time used to find and to evaluate this candidate.
- insight into the strengths and weaknesses of existing surrogate algorithms depending on problem properties, 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,
- easy to interpret rules of thumb for when to use which surrogate algorithm.

We furthermore show that continuous models can be used on discrete problems and vice versa. This is an important result as in practice, the types of variables in a problem (e.g. continuous, integer, categorical, etc.) are often used to determine which surrogate model to choose or discard. 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.

2. 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.

2.1. Surrogate-based optimisation algorithms

The goal of surrogate-based optimisation [11–13] 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. We also consider that the problem is *stochastic*, meaning we only have access to noisy measurements $y_i = f(x_i) + \epsilon_i$, where the random noise variable ϵ_i is the result of randomness in the underlying simulator or algorithm.

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

1. (*Evaluation*) Evaluate $y_i = f(x_i) + \epsilon_i$ 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:

$$x_{i+1} = \operatorname*{argmax}_{x\in X} \alpha(g(x)). \tag{2}$$

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

By far the most common surrogate algorithm is Bayesian optimisation [1,14], which typically uses a Gaussian process surrogate model. Other common surrogate models are random forests, as used in the SMAC algorithm [12], and Parzen estimators, as used in HyperOpt [7]. Our own earlier work contains random Fourier features as surrogate models in the DONE algorithm [6] and piece-wise linear surrogate models in the IDONE and MVRSM algorithms [15,16]. An overview of different methods and their surrogate models is given in Table 1. Details about which methods are included in the comparison are given in Section 3.2.

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

Surrogate models appear to be useful to solve problems with expensive objective functions, and a questionnaire on real-life optimisation problems confirms that this type of objective function often appears in practice [20]. 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 [21] 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 [10].

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 [10,22–25]. 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. 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 which can simply be described with a mathematical function.

2.3. 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 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

Table 1
Surrogate-based approaches in this benchmark environment, and whether they natively support continuous (cont.), integer (int.), categorical (cat.) and conditional (cond.) variables.

Name	Surrogate model	Cont.	Int.	Cat.	Cond.
SMAC [12,17]	Random forest	✓	✓	✓	✓
HyperOpt [7]	Parzen estimator	✓	✓	✓	✓
Bayesian Opt. [1,18]	Gaussian process (GP)	✓			
CoCaBO [19]	GP+multi-armed bandit	✓		✓	
DONE [6]	Random Fourier	✓			
IDONE [15]	Piece-wise linear		✓		
MVRSM [16]	Piece-wise linear	✓	✓		

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. This can be seen for example in the hyperparameter tuning problem in this work, where the hyperparameters can be evaluated faster than the slowest surrogate algorithm can suggest new values for the hyperparameters. 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 [23]: *"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 [16], 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 [10]. Yet for most surrogate algorithms, it is not clear how they would perform for different computational budgets.

2.4. 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 number of continuous or integer variables, evaluation time, etc.) and possibly of baseline algorithms that can be applied to the problems. For non-expensive problems, many such libraries exist [21,26,27], particularly with synthetic functions. Some of these libraries also contain real-life functions that are not expensive [28–30]. See Table 2 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 algorithm configuration problems [12], while the latter deal with (simulators of) a physical problem such as aerodynamic optimisation [31]. 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 [32–35] or computational fluid dynamics [22]. The problem with focusing on only one of the two domains is that domain-specific techniques such as early stopping of machine learning algorithms [36] or adding gradient information from differential equations [37,38] 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. Benchmarking surrogate algorithms in multiple problem domains would be beneficial for all these domains.

Though all benchmark libraries contain benchmark problems, not all of them contain *solutions* in the form of surrogate algorithms, and some of them do not even contain any type of solution at all. One library that does contain many surrogate algorithms is SUMO [39], a commercial toolbox with a wide variety of applications both in computer science and engineering. 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 processes.

3. Proposed benchmark library: EXPObench

In this section we introduce EXPObench: an EXPensive Optimisation benchmark library.[1] We propose a benchmark 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 2 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.

[1] Our code is available publicly at https://github.com/AlgTUDelft/ExpensiveOptimBenchmark.

Table 2
Related benchmark environments.

Name	Contains expensive problems	Contains engineering problems	Contains computer science problems	Implemented surrogate algorithms
HyFlex [28]			✓	0
SOS [29]		✓		0
IOHprofiler [30]		✓	✓	0
GBEA [40]	✓		✓	0
CFD [22]	✓	✓		0
NAS-Bench [33–35]	✓		✓	0
DAC-Bench [41]	✓		✓	0
RBFopt [42]		✓		1
CompModels [43]	✓	✓		1
HPObench [44]	✓		✓	2
AClib2 [45]	✓		✓	2
Nevergrad [46]	✓	✓	✓	2
BayesMark [47]	✓		✓	3
MATSuMoTo [48]				4
AMLB [32]	✓		✓	4
PySOT [49]				5
EXPObench	✓	✓	✓	**7**
SUMO [39]	✓	✓	✓	9
SMT [38]		✓		14

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

3.1. 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 [22]. Many surrogate algorithms claim to work on a wide variety of expensive optimisation problems. However, benchmarking is often limited to synthetic problems, or real-life problems from one domain, casting doubt on the validity of these claims. To verify the general usefulness of surrogate algorithms, it is important to test them on problems from different domains.

The problem dimensions in this work were chosen to be difficult for standard surrogate algorithms: Bayesian optimisation with Gaussian processes 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 5.1), the time it takes to evaluate the objective function varies between 2 and 60 s depending on the problem. However, all included benchmarks capture the properties of even more expensive problems, or can be made more expensive by changing the corresponding data or problem parameters. We also provide a method to simulate situations where evaluating the objective requires significantly more time.

By the nature of the chosen problems, there are no pre-constructed and unrealistic relations between variables. Hence, the additional complexity stemming from real-world objectives makes the benchmarking more interesting as well, such as the aforementioned greater magnitude of 'plateaus' in the ESP problem compared to typical synthetic functions. Additionally, since these objectives are more difficult to analyse, it is harder to come up with a solver that exploits any knowledge of the problem.

Table 3
Included expensive optimisation problems, and the approximate evaluation time of the objective (eval.), dimension or total number of variables (dim.), corresponding number of continuous (cont.), integer (int.), and categorical (cat.) variables, and whether some variables are conditional (cond.) variables.

Problem	Eval.	Dim.	Cont.	Int.	Cat.	Cond.
Windwake	15 s	10	10	–	–	–
Pitzdaily	2–60 s	10	10	–	–	–
ESP	28 s	49	–	–	49	–
HPO	1–8 s	135	11	7	117	Yes

We now give a short description of the four real-life expensive optimisation problems that are present in EXPObench. This information is summarised in Table 3.

3.1.1. Wind farm layout optimisation (Windwake)

This benchmark utilises a wake simulator called FLORIS [50] to determine the amount of power a given wind farm layout produces. To make the layout more robust to different wind conditions, we decided to use as output the power averaged over multiple scenarios, where each scenario uses randomly generated wind rose data, generated with the same distribution. A solution is represented by a sequence of pairs of 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 s to compute on our hardware for 5 scenarios and 5 wind turbines. 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. This is not just for modelling a realistic situation: the simulator fails to provide accurate results if this overlap is present. With the relatively low packing density present in this problem, this gives a realistic and interesting fitness landscape. 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. While more complex wake simulators exist, the problem also becomes more expensive when the number of wind turbines and the number of scenarios are increased.

3.1.2. Pipe shape optimisation (Pitzdaily)

One of the engineering benchmark problems proposed in the CFD library [22], 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 s 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. This problem becomes more expensive with an increase in the number of control points.

3.1.3. 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 [51], which takes about 28 s to return the output value on our hardware. The underlying simulator becomes more expensive when using a more fine-grained simulation mesh [51].

3.1.4. Hyperparameter optimisation and preprocessing for XGBoost (HPO)

This automated machine learning benchmark is a hyperparameter optimisation problem. The approach, namely an XGBoost [52] classifier, has already been selected. It is one of the most common machine learning models for tabular data. The model 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 s, 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. This problem becomes more expensive when using a larger dataset that requires a longer training time.

3.2. Approaches

In this section we show the approaches that are considered in the benchmark library. We limit ourselves to popular single-objective surrogate algorithms due to the limited evaluation budget that usually accompanies an expensive objective function. Furthermore, the approaches are easily implemented and open-source, and do not focus on extensions of the expensive optimisation problem such as a batch setting, multi-fidelity or multi-objective setting, highly constrained problems, etc. These include a Bayesian optimisation algorithm [1,18], which uses Gaussian processes with a Matérn 5/2 kernel and upper confidence bound acquisition function ($\beta = 2.576$), SMAC [12], and HyperOpt [7]. We also include our own earlier work [6,15,16], with the DONE, IDONE and MVRSM algorithms. A recent variant of Bayesian optimisation, namely CoCaBO [19], 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 [53], 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 1 shows the types of variables that are directly supported by the surrogate models used in each algorithm.

4. Methodology

One of the goals of this study is to obtain insight into the strengths and weaknesses of existing surrogate algorithms. In order to do this, we run the six selected approaches and random search on the four real-life optimisation problems discussed in the previous section. Due to the nature of the expensive problems, we only run every approach for 1000 iterations, and repeat the runs for 5 to 10 times depending on the expensiveness of the problem. Due to the low number of repetitions, it will be difficult to provide clear conclusions, but nevertheless we will take a hypothesis testing approach to support our claims.

The null hypothesis H_0 is that, after 1000 iterations, there is no statistical difference between the approaches (using a significance level of 95%), no matter the optimisation problem. To investigate statistical significance of the results, we report p-values of pair-wise Student's T-tests for the best objective value found at iteration 1000. Other benchmarking techniques such as showing empirical cumulative distribution functions or performance profiles [54] are not possible due to the nature of the chosen problems: there is only a limited number of problems, a limited number of iterations and repetitions, and the optimal solution and target objective are unknown.

Another goal of this study is to verify existing knowledge from literature on surrogate algorithms, which we will define in the form of alternative hypotheses $H_1 - H_7$:

- H_1: all surrogate-based approaches outperform random search on all problems.
- H_2: surrogate-based approaches with discrete surrogates outperform approaches with continuous surrogates on problems with discrete or mixed variables.
- H_3: surrogate-based approaches with continuous surrogates outperform approaches with discrete surrogates on problems with only continuous variables.
- H_4: Bayesian optimisation with Gaussian processes (BO) outperforms other approaches on continuous problems with at most 20 variables in total.
- H_5: BO is outperformed by all other approaches on discrete or mixed-variable problems and on problems with more than 20 variables in total.
- H_6: SMAC and HyperOpt outperform the other approaches on the hyperparameter optimisation problem.
- H_7: there is no statistical difference between MVRSM and IDONE for problems with only discrete variables.

Although theoretical guarantees are difficult to obtain, H_1 is often hypothesised in practice, which follows from BO often being compared against a random search baseline, see e.g. [2]. H_2 follows from the fact that using discrete surrogates is one of the options to deal with discrete search spaces, see for example the third strategy mentioned in [55], although it must be mentioned that our own earlier work counters H_2 [56]. The related H_3, although often not explicitly stated in literature, follows naturally

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

from H_2. The properties of Gaussian processes lead to H_4 and H_5, as seen in e.g. [57–59]. SMAC and HyperOpt are designed for tuning the configuration of (machine learning) algorithm hyperparameters, and naturally deal with conditional variables due to their tree-structured surrogates, leading to H_6. Finally, H_7 follows from our knowledge of the algorithms MVRSM and IDONE: they use exactly the same surrogate for problems with only discrete variables, and their only difference is in the way exploration is performed.

Besides investigating these hypotheses at iteration 1000, we are also interested in the performance of the different approaches for other situations. Instead of only varying the number of iterations, we choose to vary the available computation time for each algorithm, while also artificially varying the computational cost of evaluating the objective function. This way, we can extrapolate our results to situations where the evaluation of the objective function takes much shorter or longer, which is valuable for practical applications with a limited computational budget. The next section explains how this experiment is done exactly, and also shows several rules of thumb that were obtained by making use of a decision tree classifier. All of this is done based on the data obtained during the hypothesis testing approach.

Finally, we conduct an offline supervised learning experiment to more thoroughly investigate different surrogate models, as explained in the next section. This way, we can separately investigate the accuracy of different surrogate models, as well as their generalisation capabilities, for various problems.

5. 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 to (1) gain insight into the strengths and weaknesses of existing surrogate algorithms and verify existing knowledge from literature, (2) investigate how algorithm performance depends on the available computational resources and the cost of the expensive objective, and (3) 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 [60] or building new surrogate benchmarks from this tabular data [35].

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. The section ends with a summary of the obtained insights.

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

5.1. Experiment details

5.1.1. Hardware

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

5.1.2. 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.

5.1.3. 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.[4] 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 Fig. 1. This normalisation is possible since all surrogate algorithms start with the same number of random evaluations R, which we omit from the figures. Another metric, namely the area under the curve, is shown in Appendix A.

5.1.4. Software environment

EXPObench is available as a public github repository[5] 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.

5.2. 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.

[4] Note: we used the same uniform distribution for the random initial guesses of all methods, and new samples were drawn at every run to remove dependence on the initialisation. The samples themselves are different across algorithms.

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

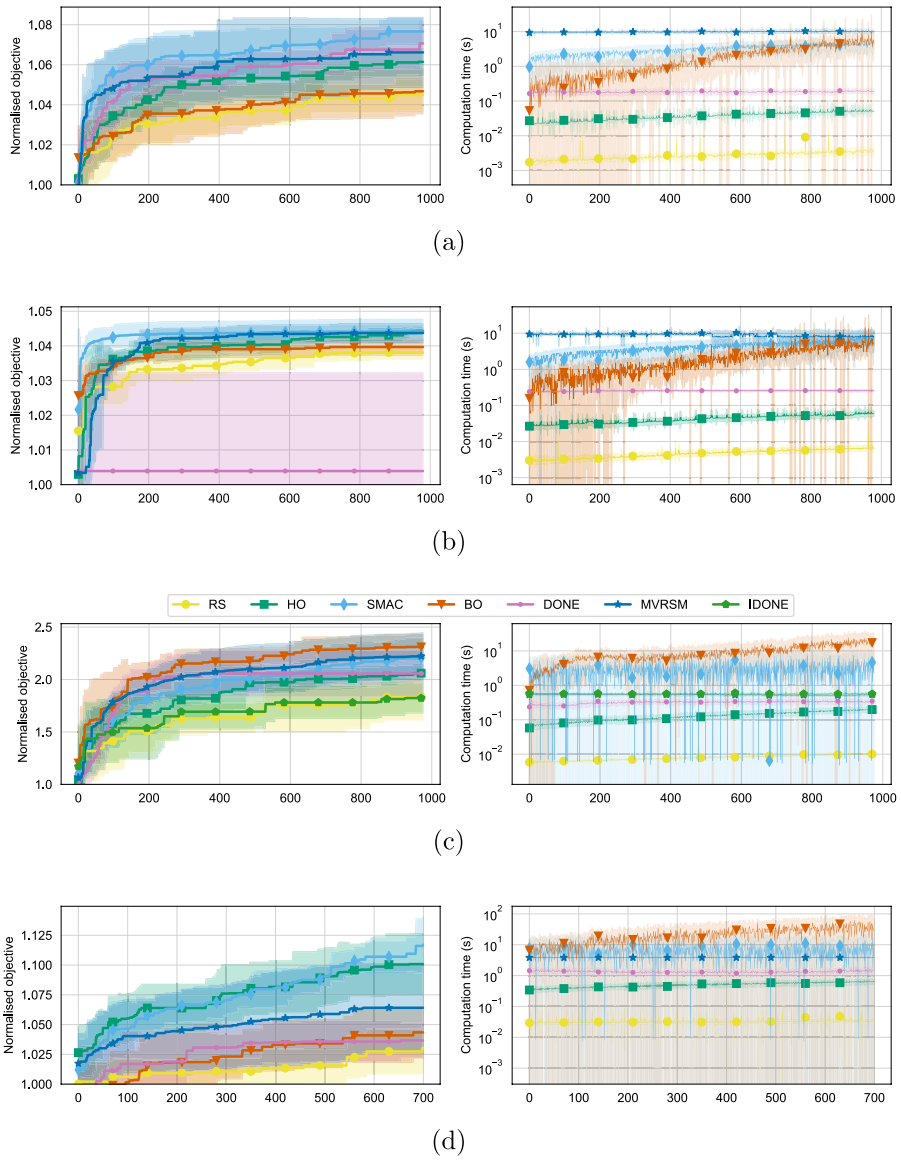

Fig. 1. Results on the different benchmark problems, averaged over T runs, after starting with R random samples. T is varied due to the different degrees of expensiveness of the problems. The shaded area indicates one standard deviation, the horizontal axis indicates the iteration of the algorithm, and 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, $T = 10$, $R = 300$.

5.2.1. Windwake

For the wind farm layout optimisation problem, Fig. 1(a) 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 s 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 ($p > 0.1$). These results lead us to reject the null hypothesis H_0, and to also reject H_1 (as not all algorithms outperform random search), as well as H_3 and H_4 (as BO did not outperform the algorithms with discrete surrogate models).

5.2.2. Pitzdaily

Fig. 1(b) 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 ($p > 0.6$). Again, Bayesian optimisation (BO) does not

outperform random search on this problem ($p > 0.6$). This supports the rejection of the same hypotheses as for the windwake problem (H_0, H_1, H_3, H_4).

5.2.3. ESP

In this discrete problem, algorithms that only deal with continuous variables resort to rounding when calling the expensive objective. However, we see in Fig. 1(c) 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, and causes H_2 and H_5 to be rejected. 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. This causes H_7 to be rejected. The surrogate algorithms use less computation time than the expensive objective which took about 28 s per iteration to evaluate. The null hypothesis H_0 is again rejected.

5.2.4. 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. Fig. 1(d) 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 each other ($p > 0.1$), they outperform all other methods ($p < 0.03$), leading us to accept H_6. 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, while BO is even slower than SMAC. The null hypothesis H_0 is again rejected.

Looking at the results of all four problems, only hypothesis H_6 is accepted, while all the other hypotheses are rejected. Rejecting H_1 does not mean that no surrogate algorithm outperforms random search, just that for all problems we can find a surrogate algorithm that does not outperform random search. The only surrogate algorithms that outperform random search on all four problems are SMAC, HyperOpt, and MVRSM.

5.3. Varying time budget and function evaluation time

In this experiment 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. This 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 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 [56]. Because we know the 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 evaluation 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.

Fig. 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 almost all algorithms 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.

5.3.1. Rules of thumb

To turn these insights into easily interpretable rules of thumb, we have trained a decision tree classifier on the data points of Fig. 2, and the corresponding decision rules are indicated by regions separated by black lines. The decision tree takes as input the time budget and evaluation time, and two problem features: one feature that indicates whether the problem is a 10-dimensional continuous problem (true for Windwake and PitzDaily, false for ESP and HPO), and a feature that indicates whether the problem contains a computational fluid dynamics (CFD) simulator (true for PitzDaily and ESP, false for ESP and HPO). These features were chosen to prevent problem-specific features: now, at least two features are needed to get a decision for one specific problem. The class label output of the decision tree is the best surrogate algorithm according to the data, which it was able to predict with a training accuracy of 0.63 and a test accuracy of 0.71 after a 80%−20% train-test split and repeating the training procedure 10 times and keeping the tree with the highest test accuracy. This procedure took less than one second of computation time. The decision tree had a maximum depth of 5 and a maximum of 6 leaf nodes, while otherwise using default hyperparameter settings from Python's scikit-learn package.

The resulting decision tree led to the following rules of thumb for which surrogate algorithm to choose:

1. For cheap objective functions (at most tens of milliseconds evaluation time) with a tight time budget (around 1 s or less), BO is a good option, with random search as a close second.

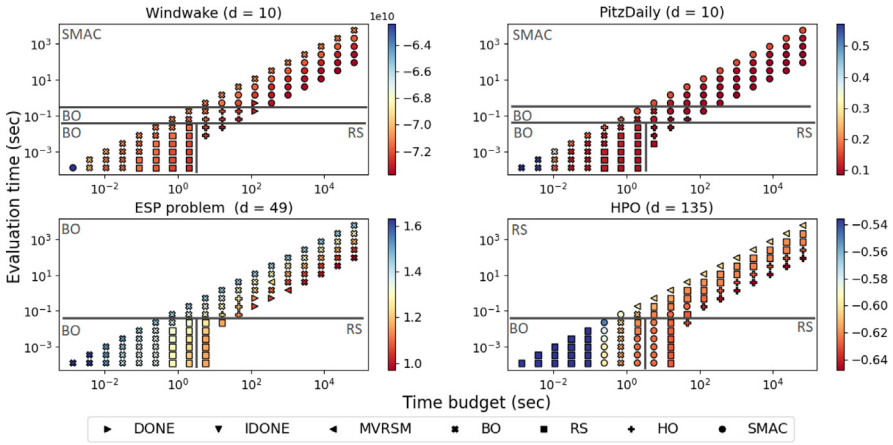

Fig. 2. The best surrogate algorithm for the case that the evaluation time of the objective is artificially changed (vertical axis), and for different time budgets (horizontal axis). The different marker shapes indicate which of the surrogate algorithms achieved the best objective value, while the colour shows the corresponding objective value (not normalised, lower is better). The regions divided by black lines show which algorithm would perform best according to a decision tree trained on the data. 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.

2. For cheap objective functions with a high time budget (at least seconds), random search is the best option, meaning no surrogate algorithm is required. This is the typical setting for black-box optimisation, so we expect many algorithms to outperform surrogate algorithms in this case.
3. For expensive high-dimensional non-continuous problems that do not make use of a CFD simulator (like hyperparameter optimisation), random search is a good option, with HyperOpt and MVRSM as close alternatives.
4. For expensive high-dimensional non-continuous problems that make use of a CFD simulator (like ESP), BO is the best option.
5. For semi-expensive (tens to hundreds of milliseconds) continuous 10-dimensional problems, BO is a good option, with SMAC and HyperOpt as close alternatives.
6. For expensive continuous 10-dimensional problems, SMAC is the best option.

Together, these rules of thumb can give practical insights which would otherwise require familiarity with the literature on surrogate algorithms. Besides this, while most of these rules of thumb are in line with existing literature, rules 4 and 6 actually oppose existing knowledge. This is because BO in this work makes use of a Gaussian process, which is often claimed to work well only on continuous problems with 20 variables or less, see e.g. [58,59], while rule 4 shows it works well on high-dimensional non-continuous problems such as ESP. On the other hand, rule 6 shows that SMAC, which uses a random forest surrogate, works well mostly for continuous 10-dimensional problems, while random forests are well-suited for high-dimensional non-continuous search spaces. This gives further experimental evidence against our hypotheses H_4 and H_5.

Finally, several problem settings have been identified where random search shows a strong performance, such as those in rules 1, 2 and 3. It is likely that other black-box optimisation algorithms that do not make use of a surrogate, such as evolutionary algorithms, would outperform surrogate-based methods in these cases, though more research is necessary to verify this. Though rules 1 and 2 deal with cheap objective functions, where our earlier hypotheses do not apply, rule 3 gives further experimental evidence against hypothesis H_1.

5.4. Offline learning of surrogates

As a final experiment we investigate the choice of surrogate model in the different surrogate algorithms. We show how the dataset generated in this work can be used in a simple offline supervised learning framework by training and testing different models on the data and considering the resulting errors. 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 Appendix B). 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 [61], and we also add XGBoost and the piece-wise linear model used by the IDONE and MVRSM algorithms, giving the following models: linear regression model (Linear), piece-wise linear model (PWL), random forest with default hyperparameters (RF), XGBoost with default hyperparameters (XGBoost), and the Gaussian process used by Bayesian optimisation (GP).

As a test set 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 for each problem. As the global optimum is unknown in these problems, this shows how the different models would perform in good regions of the search space.

Table 4 shows the results of training each model on data gathered by random search, averaged over different runs. We can immediately see that some models are prone to overfitting: the models with the smallest training errors are not necessarily the most accurate near the optimum, and may even be outperformed by a simple linear regression model there. Furthermore, discrete models such as random forest and XGBoost with 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 5. The models PWL and GP are exactly the same as the ones used in the corresponding surrogate algorithms (IDONE/MVRSM and BO respectively), while SMAC uses a random forest with different hyperparameters than the RF model

Table 4
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
PWL	0.006	1.323	$\mathbf{5 \cdot 10^{-9}}$	8.869
RF	0.151	**0.758**	0.574	0.972
XGBoost	0.279	0.849	0.153	**0.711**
GP	$\mathbf{8 \cdot 10^{-7}}$	1.147	$1 \cdot 10^{-6}$	1.297

Table 5
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	$\mathbf{5 \cdot 10^{-8}}$	1.888	$\mathbf{4.8 \cdot 10^{-7}}$	0.910

used here, and DONE only uses an approximation of a Gaussian process. 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 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 the GP 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; see Table 4. The test error for XGBoost trained on data gathered by BO, not shown in these tables, is 0.997 for the Pitzdaily problem and 0.701 for the ESP problem.

5.5. Summary of obtained insights

Based on the experimental results, 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. For the latter problem, changing the kernel type or parameters of the Gaussian process might improve results, however we expected the chosen settings to work well for the problem. Part of these insights were known from previous work [56], but we extended these insights to continuous problems and to more benchmark problems and surrogate algorithms. The claim is supported by the rejection of H_4 and H_5 and by rules of thumb 4 and 6 of Section 5.3.1, while the other rules of thumb were more in line with expectations. The experiments using offline learning of machine learning models also showed that discrete models such as random forests can have lower generalisation error than continuous models, even on data coming from a continuous problem like Pitzdaily. This result is surprising, considering random forests are known to have poor extrapolation capabilities. Our way of benchmarking has shown that discrete models can be an efficient and accurate choice for real-life expensive problems, even continuous ones, while continuous models can be useful for discrete problems in practice.

Second, our observations lead us to believe that *exploration is more important than model accuracy* in surrogate algorithms. The offline learning experiments, a unique addition to our way of benchmarking, 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. This is supported by the first three rules of thumb in Section 5.3.1. 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. For discrete problems, 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 for the considered problems, 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: the best performing algorithm then varies depending on the available time budget and function evaluation time. This claim is supported by all rules of thumb in Section 5.3.1. In fact, the evaluation time of the objective was the most important feature of the decision tree classifier that we used to generate the rules of thumb. Our way of benchmarking has allowed us to obtain such valuable insights concerning time budget and evaluation time, which were not thoroughly investigated in this way before.

6. Conclusion and future work

We proposed a public benchmark library called EXPObench, which 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. This resulted in a dataset containing the results of running multiple surrogate-based optimisation algorithms on several expensive problems, which can be used to create tabular or surrogate benchmarks or for meta-learning. A first analysis of this 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. We also provided easy to interpret rules of thumb based on our analysis, showing when to use which surrogate algorithm. Furthermore, the dataset 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 , particularly for multi-objective problems.

Table A.6
Area under the curve metrics after 500 and 1000 iterations (higher is better), averaged over multiple runs.

Benchmark	Windwake		Pitzdaily		ESP		HPO	
Algorithm at iteration	500	1000	500	1000	500	1000	500	1000
RS	0.963844	0.968033	0.976079	0.983226	0.773124	0.804465	0.930773	0.935801
BO	0.965312	0.969517	0.986475	0.989424	**0.872638**	**0.909874**	0.925708	0.936327
HO	0.970493	0.975888	0.986173	0.990593	0.810760	0.849704	**0.937372**	**0.952699**
SMAC	**0.980197**	**0.984984**	**0.989619**	**0.993185**	0.827741	0.870288	0.932997	0.951097
DONE	0.974548	0.979858	0.953432	0.955938	0.840280	0.869861	0.923643	0.935636
IDONE	–	–	–	–	0.787532	0.812625	–	–
MVRSM	0.976244	0.980730	0.987027	0.991660	0.850981	0.887566	0.933505	0.945031

Table B.7
Mean average error for models trained on data gathered by one surrogate algorithm, averaged over different runs.

Benchmark	Windwake		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	$\mathbf{3 \cdot 10^{4}} \pm 3 \cdot 10^{3}$	$\mathbf{2.5 \cdot 10^{10}} \pm 2 \cdot 10^{9}$	$\mathbf{9 \cdot 10^{-7}} \pm 2 \cdot 10^{-8}$	$\mathbf{0.92} \pm 0.05$	$\mathbf{3 \cdot 10^{-7}} \pm 2 \cdot 10^{-7}$	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	$2 \cdot 10^{-3} \pm 10^{-3}$	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

CRediT authorship contribution statement

Laurens Bliek: Conceptualization, Methodology, Validation, Formal analysis, Data curation, Writing, Visualization, Project administration. **Arthur Guijt:** Software, Investigation, Data curation, Writing. **Rickard Karlsson:** Investigation, Writing. **Sicco Verwer:** Conceptualization, Writing – review & editing, Supervision, Funding acquisition. **Mathijs de Weerdt:** Conceptualization, Writing – review & editing, supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Link to dataset: https://doi.org/10.4121/14247179.v2.

Acknowledgements

This work is part of the research programme Real-time data-driven maintenance logistics with project number 628.009.012, which is financed by the Dutch Research Council (NWO, The Netherlands).

Appendix A. Area under curve metrics

This section shows the area under the curve (AUC) of the best found objective value for each method on each problem in the benchmark library. See Table A.6. Note that here we used a different normalisation procedure before calculating the AUC: the random initial samples were included, and then for every problem, results were normalised as follows:

$$f_{norm} = (f - f_{max})/(f_{min} - f_{max}), \tag{A.1}$$

where f_{max} (f_{min}) is the best (worst) found objective function across all methods and iterations for a particular problem.

Appendix B. Offline learning results

Here we show more data of the offline learning experiment presented in Section 5.4, including the standard deviations. See Table B.7. 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. New models that were not explained in the main text are two models from scikit-learn: a polynomial model with degree 2 (Quadratic), and a multi-layer perceptron with default hyperparameters (MLP).

References

[1] B. Shahriari, K. Swersky, Z. Wang, R. Adams, N.D. Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proc. IEEE 104 (2016) 148–175.
[2] Q. Liang, A.E. Gongora, Z. Ren, A. Tiihonen, Z. Liu, S. Sun, J.R. Deneault, D. Bash, F. Mekki-Berrada, S.A. Khan, K. Hippalgaonkar, B. Maruyama, K.A. Brown, J.W.F. Iii, T. Buonassisi, Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains, npj Comput. Mater. 7 (2021) 1–10.
[3] M. Fiducioso, S. Curi, B. Schumacher, M. Gwerder, 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, 2019, pp. 5850–5856, http://dx.doi.org/10.24963/ijcai.2019/811.
[4] F. Bre, N.D. Roman, V.D. Fachinotti, An efficient metamodel-based method to carry out multi-objective building performance optimizations, Energy Build. 206 (2020) 109576.
[5] A.J. Keane, I.I. Voutchkov, Surrogate approaches for aerodynamic section performance modeling, AIAA J. 58 (2020) 16–24.
[6] L. Bliek, H.R.G.W. Verstraete, M. Verhaegen, S. Wahls, Online optimization with costly and noisy measurements using random Fourier expansions, IEEE Trans. Neural Netw. Learn. Syst. 29 (1) (2018) 167–182.
[7] J. Bergstra, D. Yamins, 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.
[8] L. Bliek, A survey on sustainable surrogate-based optimisation, Sustainability 14 (7) (2022) http://dx.doi.org/10.3390/su14073867, URL https://www.mdpi.com/2071-1050/14/7/3867.
[9] S.P. Hellan, C.G. Lucas, N.H. Goddard, Bayesian optimisation against climate change: Applications and benchmarks, 2023, arXiv:2306.04343.
[10] P.S. Palar, R.P. Liem, L.R. Zuhal, 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, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1592–1602, http://dx.doi.org/10.1145/3319619.3326813.
[11] A. Bhosekar, M.G. Ierapetritou, Advances in surrogate based modeling, feasibility analysis, and optimization: A review, Comput. Chem. Eng. 108 (2018) 250–267.
[12] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.
[13] R. Alizadeh, J.K. Allen, F. Mistree, Managing computational complexity using surrogate models: a critical review, Res. Eng. Des. 31 (2020) 275–298.
[14] J. Močkus, On Bayesian methods for seeking the extremum, in: Optimization Techniques IFIP Technical Conference, Springer, 1975, pp. 400–404.
[15] L. Bliek, S. Verwer, M. de Weerdt, Black-box combinatorial optimization using models with integer-valued minima, Ann. Math. Artif. Intell. (2020) 1–15, http://dx.doi.org/10.1007/s10472-020-09712-4.
[16] L. Bliek, A. Guijt, S. Verwer, M. de Weerdt, Black-box mixed-variable optimisation using a surrogate model that satisfies integer constraints, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1851–1859, http://dx.doi.org/10.1145/3449726.3463136.
[17] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential Model-Based Optimization for General Algorithm Configuration (Extended Version), Technical Report TR-2010–10, Tech. Rep., University of British Columbia, Computer Science, 2010.
[18] F. Nogueira, Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–, URL https://github.com/fmfn/BayesianOptimization.
[19] B. Ru, A. Alvi, V. Nguyen, M.A. Osborne, S. Roberts, Bayesian optimisation over multiple continuous and categorical inputs, in: H.D. III, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 8276–8285.
[20] K. van der Blom, T.M. Deist, V. Volz, M. Marchi, Y. Nojima, B. Naujoks, A. Oyama, T. Tušar, Identifying properties of real-world optimisation problems through a questionnaire, 2020, arXiv preprint arXiv:2011.05547. arXiv:2011.05547.
[21] 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, A. Auger, COmparing Continuous Optimizers: numbbo/COCO on Github, 2019, http://dx.doi.org/10.5281/zenodo.2594848.
[22] S.J. Daniels, A.A. Rahat, R.M. Everson, G.R. Tabor, 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.
[23] V. Volz, B. Naujoks, On benchmarking surrogate-assisted evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019.
[24] A. Tangherloni, S. Spolaor, P. Cazzaniga, D. Besozzi, L. Rundo, G. Mauri, M.S. Nobile, Biochemical parameter estimation vs. benchmark functions: A comparative study of optimization performance and representation design, Appl. Soft Comput. 81 (2019).
[25] J.M. Dieterich, B. Hartke, Empirical review of standard benchmark functions using evolutionary global optimization, 2012, arXiv preprint arXiv:1207.4318. arXiv:1207.4318.
[26] S. Wagner, M. Affenzeller, HeuristicLab: A generic and extensible optimization environment, in: B. Ribeiro, R.F. Albrecht, A. Dobnikar, D.W. Pearson, N.C. Steele (Eds.), Adaptive and Natural Computing Algorithms, Springer Vienna, Vienna, 2005, pp. 538–541.
[27] J. Humeau, A. Liefooghe, E. Talbi, S. Vérel, ParadisEO-MO: from fitness landscape analysis to efficient local search algorithms, J. Heuristics 19 (2013) 881–915.
[28] 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.
[29] F. Caraffini, G. Iacca, The SOS platform: Designing, tuning and statistically benchmarking optimisation algorithms, Mathematics 8 (5) (2020) http://dx.doi.org/10.3390/math8050785, URL https://www.mdpi.com/2227-7390/8/5/785.
[30] C. Doerr, H. Wang, F. Ye, S. van Rijn, T. Bäck, IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics, 2018, arXiv preprint arXiv:1810.05281.

[31] J. Liu, Z.-H. Han, W. Song, Comparison of infill sampling criteria in Kriging-based aerodynamic optimization, in: 28th Congress of the International Council of the Aeronautical Sciences 2012, ICAS 2012, Vol. 2, 2012, pp. 1625–1634.
[32] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, J. Vanschoren, An open source AutoML benchmark, 2019, arXiv preprint arXiv:1907.00909 [cs.LG]. Accepted at AutoML Workshop at ICML 2019. URL https://arxiv.org/abs/1907.00909.
[33] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, F. Hutter, NAS-Bench-101: Towards reproducible neural architecture search, in: ICML, 2019, pp. 7105–7114.
[34] X. Dong, Y. Yang, NAS-Bench-201: Extending the scope of reproducible neural architecture search, 2020, ArXiv abs/2001.00326.
[35] J.N. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, F. Hutter, NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search, 2020, ArXiv abs/2008.09777.
[36] S. Falkner, A. Klein, F. Hutter, BOHB: Robust and efficient hyperparameter optimization at scale, in: Proceedings of the 35th International Conference on Machine Learning, 2018, pp. 1436–1445.
[37] Z. hua Han, S. Görtz, R. Zimmermann, Improving variable-fidelity surrogate modeling via gradient-enhanced kriging and a generalized hybrid bridge function, Aerosp. Sci. Technol. 25 (2013) 177–189.
[38] M.A. Bouhlel, J.T. Hwang, N. Bartoli, R. Lafage, J. Morlier, J.R.R.A. Martins, A Python surrogate modeling framework with derivatives, Adv. Eng. Softw. (2019) 102662, http://dx.doi.org/10.1016/j.advengsoft.2019.03.005.
[39] D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, K. Crombecq, A surrogate modeling and adaptive sampling toolbox for computer based design, J. Mach. Learn. Res. 11 (2010) 2051–2055.
[40] V. Volz, B. Naujoks, P. Kerschke, T. Tusar, Single- and multi-objective game-benchmark for evolutionary algorithms, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019.
[41] T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, 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, 2021, pp. 1668–1674.
[42] A. Costa, G. Nannicini, RBFOpt: an open-source library for black-box optimization with costly function evaluations, Math. Program. Comput. 10 (2018) 597–629.
[43] T. Pourmohamad, CompModels: A suite of computer model test functions for Bayesian optimization, 2020, arXiv: Computation.
[44] K. Eggensperger, M. Feurer, A. Klein, S. Falkner, HPObench, 2016, https://github.com/automl/HPOBench.
[45] M. Lindauer, AClib2, 2016, https://bitbucket.org/mlindauer/aclib2.
[46] J. Rapin, O. Teytaud, Nevergrad - A gradient-free optimization platform, 2018, https://GitHub.com/FacebookResearch/Nevergrad.
[47] R. Turner, D. Eriksson, BayesMark, 2018, https://github.com/uber/bayesmark.
[48] J. Mueller, MATSuMoTo, 2014, https://github.com/Piiloblondie/MATSuMoTo.
[49] D. Eriksson, D. Bindel, C.A. Shoemaker, pySOT and POAP: An event-driven asynchronous framework for surrogate optimization, 2019, arXiv preprint arXiv:1908.00420.
[50] NREL, FLORIS. Version 2.1.1, 2020, URL https://github.com/NREL/floris.
[51] F. Rehbach, M. Zaefferer, J. Stork, T. Bartz-Beielstein, Comparison of parallel surrogate-assisted optimization approaches, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2018, pp. 1348–1355.
[52] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: ACM SIGKDD, 2016, pp. 785–794.
[53] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305.
[54] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. López-Ibáñez, K.M. Malan, J.H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, T. Weise, Benchmarking in optimization: Best practice and open issues, 2020, ArXiv abs/2007.03488.
[55] T. Bartz-Beielstein, M. Zaefferer, Model-based methods for continuous and discrete global optimization, Appl. Soft Comput. 55 (2017) 154–167, http://dx.doi.org/10.1016/j.asoc.2017.01.039, URL https://www.sciencedirect.com/science/article/pii/S1568494617300546.
[56] R. Karlsson, L. Bliek, S. Verwer, 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.
[57] B. Letham, R. Calandra, A. Rai, E. Bakshy, Re-examining linear embeddings for high-dimensional Bayesian optimization, 2020, ArXiv abs/2001.11659.
[58] C. Oh, J. Tomczak, E. Gavves, M. Welling, Combinatorial Bayesian optimization using the graph cartesian product, in: Advances in Neural Information Processing Systems, Vol. 32, 2019, pp. 1–11, URL https://proceedings.neurips.cc/paper/2019/file/2cb6b10338a7fc4117a80da24b582060-Paper.pdf.
[59] R. Moriconi, M.P. Deisenroth, K.S.S. Kumar, High-dimensional Bayesian optimization using low-dimensional feature spaces, Mach. Learn. 109 (2020) 1925–1943.
[60] K. Smith-Miles, D. Baatar, B. Wreford, R. Lewis, Towards objective measures of algorithm performance across instance space, Comput. Oper. Res. 45 (2014) 12–24, http://dx.doi.org/10.1016/j.cor.2013.11.015.
[61] 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, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.