



**HAL**  
open science

# Surrogate-Based Optimization of System Architectures Subject to Hidden Constraints

Jasper H Bussemaker, Paul Saves, Nathalie Bartoli, Thierry Lefebvre, Björn  
Nagel

► **To cite this version:**

Jasper H Bussemaker, Paul Saves, Nathalie Bartoli, Thierry Lefebvre, Björn Nagel. Surrogate-Based Optimization of System Architectures Subject to Hidden Constraints. 2024. hal-04462408v1

**HAL Id: hal-04462408**

**<https://hal.science/hal-04462408v1>**

Preprint submitted on 16 Feb 2024 (v1), last revised 21 Aug 2024 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Surrogate-Based Optimization of System Architectures Subject to Hidden Constraints

Jasper H. Bussemaker\*

*DLR (German Aerospace Center), Institute of System Architectures in Aeronautics, Hamburg, Germany*

Paul Saves<sup>†</sup>, Nathalie Bartoli<sup>‡</sup>, Thierry Lefebvre<sup>§</sup>  
*ONERA, DTIS, Université de Toulouse, Toulouse, France*

Björn Nagel<sup>¶</sup>

*DLR (German Aerospace Center), Institute of System Architectures in Aeronautics, Hamburg, Germany*

System architecture optimization can support the design of novel architectures by formulating the architecting process as an optimization problem. The exploration of novel architectures requires physics-based simulation due to a lack of prior experience to start from, which introduces two specific challenges for optimization algorithms: evaluations become more expensive (in time) and evaluations might fail. The former challenge is addressed by Surrogate-Based Optimization (SBO) algorithms, in particular Bayesian Optimization (BO) using Gaussian Process (GP) models (Kriging). An overview is provided of how BO can deal with challenges specific to architecture optimization, such as design variable hierarchy and multiple objectives: specific measures include ensemble infills, a multi-step infill optimization, and hierarchical sampling and correction algorithms. Evaluations might fail due to non-convergence of underlying solvers or infeasible geometry in certain areas of the design space. Such failed evaluations, also known as hidden constraints, pose a particular challenge to SBO/BO, as the surrogate model cannot be trained on empty results. This work investigates various strategies for satisfying hidden constraints in BO algorithms. Three high-level strategies are identified: *rejection* of failed points from the training set, *replacing* failed points based on non-failed points, and *predicting* the failed region. Through investigations on a set of test problems including a jet engine architecture optimization problem, it is shown that best performance is achieved with a mixed-discrete GP to predict the Probability of Viability (PoV), and by ensuring selected infill points satisfy some minimum PoV threshold. The jet engine architecture problem features a 50% failure rate and could not previously be solved by a BO algorithm. The developed BO algorithm and used test problems are available in the open-source Python library `SBOARCHOPT`.

## I. Nomenclature

$c_h$	=	Hidden constraint
$c_{v,l}$	=	Value constraint $l$
$FR$	=	Failure rate
$f_m$	=	Objective function $m$
$g_k$	=	Inequality constraint $k$
$HV$	=	Hypervolume
$IR$	=	Imputation ratio ( $d$ subscript: discrete $IR$ ; $c$ subscript: continuous $IR$ )
$k_{doe}$	=	DoE multiplier
$MRD$	=	Maximum rate diversity
$n_{c_v}$	=	Number of value constraints

---

\*Researcher, DDP Group, Department of Digital Methods for System Architecting, jasper.bussemaker@dlr.de

<sup>†</sup>Research engineer, Information Processing and Systems Department, paul.saves@onera.fr

<sup>‡</sup>Senior researcher, Information Processing and Systems Department, nathalie.bartoli@onera.fr, AIAA Member MDO TC

<sup>§</sup>Research engineer, Information Processing and Systems Department, thierry.lefebvre@onera.fr

<sup>¶</sup>Institute Director, Institute of System Architectures in Aeronautics, Hamburg, bjoern.nagel@dlr.de

$n_f$	=	Number of objectives
$n_g$	=	Number of inequality constraints
$n_{batch}$	=	Batch size for infill points
$n_{doe}$	=	Design of Experiments size
$n_{infill}$	=	Total number of infill points generated
$n_{kpls}$	=	PLS components for KPLS
$n_{parallel}$	=	Maximum number of parallel evaluations
$n_{valid,discr}$	=	Number of valid discrete design vectors in an optimization problem
$n_x$	=	Number of design variables
$n_{x_c}$	=	Number of continuous design variables
$n_{x_d}$	=	Number of discrete design variables
$N_{fe}$	=	Number of function evaluations
$N_j$	=	Number of options for discrete variable $j$
$\hat{s}$	=	Uncertainty estimate by a surrogate model
$\mathbf{x}$	=	Design vector
$x_{e,i}$	=	Continuous design variable $i$
$x_{d,j}$	=	Discrete design variable $j$
$\hat{y}$	=	Function value estimate by a surrogate model
$\delta_i$	=	Activeness function for design variable $i$

## II. Introduction

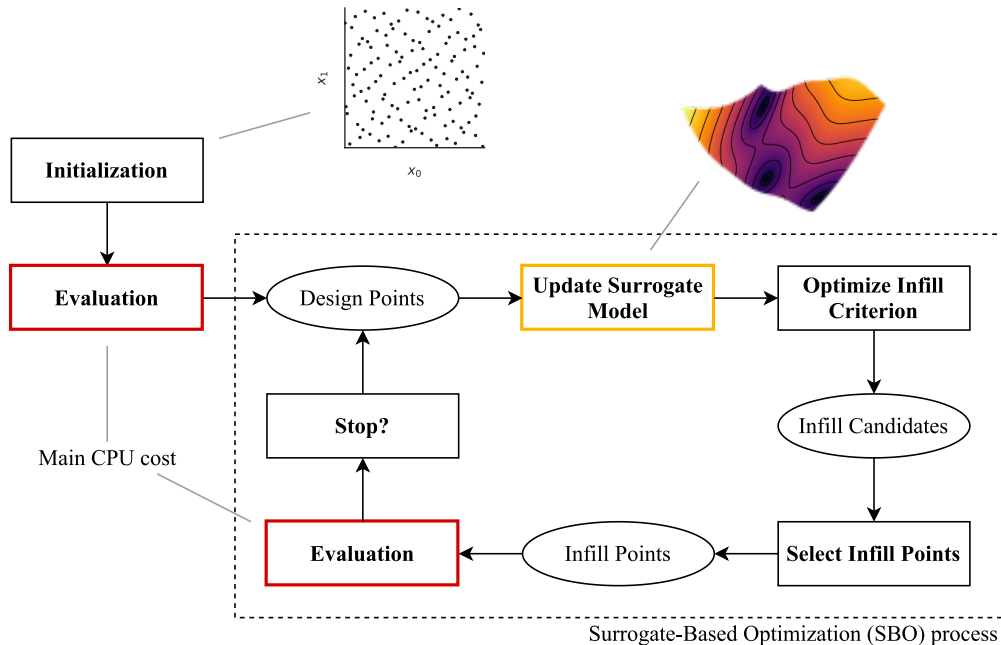
SYSTEM architecture optimization is an emerging field with the goal of partial automation of designing system architectures by formulating the architecting process as a numerical optimization problem [1]. This way, a larger design space can be explored than is possible nowadays [2], and bias towards conventional solutions can be reduced [3]. This work deals with the aspect of developing efficient optimization algorithms for solving architecture optimization problems. For more background on architecture optimization, including how to model architecture design spaces and how to setup evaluation code, the interested reader is referred to [4, 5].

System architecture optimization problems are subject to various aspects that make them challenging to solve: the presence of *mixed-discrete* design variables stemming from architectural decisions, *multiple objectives* to optimize for stemming from conflicting stakeholder needs [6], and the fact that the evaluation functions have to be treated as *black-box* functions [7]. Another salient feature of architecture optimization is design variable *hierarchy* [8]: the phenomenon that some design variables might decide whether other design variables are active or not, and might restrict the available options of other design variables. Consider for example a launch vehicle design problem where both the number of rocket stages and the fuel type of each stage are design variables [9]: it follows that the selection of the number of stages determines whether some of the fuel-type selection variables are active or not. Evolutionary algorithms such as NSGA-II are well-suited for solving such optimization algorithms and are currently available for use [1, 10–12]. When searching for novel system architectures to solve for example sustainability challenges, design expertise based on historical regression analysis may be not available, however. In general when designing new architectures, physics-based simulation is needed to accurately predict the performance of architectural candidates. This trend has two major consequences for optimization algorithms [12]:

- 1) Evaluating a design candidate is *expensive*, and therefore the optimizer should aim to find the optimum (or Pareto front) in as little function evaluations as possible;
- 2) Evaluation of a design candidate can *fail*, for example because of unstable underlying equations, infeasible physics, or infeasible geometry.

To deal with the challenges of *expensive* evaluation, Surrogate Based Optimization (SBO) algorithms have been developed. These algorithms train a regression or interpolation surrogate model of the objective  $f(\mathbf{x})$  and constraint  $g(\mathbf{x})$  functions with respect to the design variables  $\mathbf{x}$ . These models are then used to generate *infill* points for exploring the design space and search for the optimum or Pareto front. The principle of SBO is visualized in Figure 1. An especially powerful type of SBO is Bayesian Optimization (BO), where Gaussian Process (GP) models are used to provide not only an estimate of the function value  $\hat{y}(\mathbf{x})$  but also an estimate of the associated uncertainty  $\hat{s}(\mathbf{x})$ . BO is more effective at solving optimization problems than SBO algorithms with other surrogate models [13], and has been extended to work for constrained [14, 15], mixed-discrete [13, 16], hierarchical [9, 17, 18] and multi-objective [19, 20]

problems. The combination of all aspects has been demonstrated for architecture optimization [12, 21, 22].



**Fig. 1 Principle of Surrogate-Based Optimization (SBO), reproduced from [12].**

Evaluation failures manifest themselves as so-called *hidden constraints* [23], which are also known as unknown, unspecified, forgotten, virtual, and crash constraints [23, 24]. When a hidden constraint is violated (i.e. an evaluation has failed), the objective  $f$  and constraint  $g$  functions are assigned NaN (Not-a-Number). In the taxonomy of LE DIGABEL & WILD [24] hidden constraints are classified as NUSH constraints: Non-quantifiable (the degree of constraint violation is not available, only whether it is violated or not), Unrelaxable (violating the constraint yields no meaningful information about the design space), Simulation (a simulation must be run in order to find out the status), and Hidden (the existence of the constraint is not known before solving the problem). Hidden constraints are assumed to be deterministic, mainly meaning that we do not consider evaluation failures due to temporary problems such as software license availability or network problems. Finding out whether the hidden constraint is violated may take as long, if not longer, than a normal evaluation [23]. The design space can be separated in regions with satisfied and violated hidden constraints, denoted as *viable* and *failed* regions, respectively. Depending on the design problem, the failed region can take up a significant part of the design space: KRENGEL & HEPPERLE report up to 60% for a wing design problem [25], and for an airfoil design problem FORRESTER ET AL. up to 82% failed points [26]. Hidden constraints pose a particular challenge to SBO algorithms however: evaluation failures assign NaN (not-a-number) values to objective and constraints, and it is not possible to train a surrogate model on NaN's [10].

A review of dealing with hierarchical optimization challenges for BO will be given in Section III, which is extended to hidden constraints in Section IV. The paper finishes with a demonstration of a jet engine optimization problem in Section V and the conclusions in Section VI.

### III. Bayesian Optimization in Hierarchical Design Spaces

Before diving into strategies for dealing with hidden constraints, this section reviews strategies for dealing with the other challenges of architecture optimization using Bayesian Optimization (BO) algorithms as originally published in [12], and introduces the BO algorithm how it will be used in the rest of this work. The algorithm is implemented as ARCHSBO in the open-source SBARCHOPT\* Python library [27].

\*<https://sbarchopt.readthedocs.io/>

**Gaussian Process Model Selection** For non-hierarchical problems, the mixed-discrete Gaussian Process (GP) models (also known as Kriging models) implemented in the Surrogate Model Toolbox (SMT)<sup>†</sup> are used. These models support continuous, integer and categorical variables through kernels developed by SAVES ET AL. [18, 28]. GP models that support hierarchical variables have been developed in [18] and extended to also support categorical hierarchical variables in [12]. Investigations have shown that these hierarchical GP models outperform non-hierarchical GP models for problems with non-linear or non-smooth objective or constraint functions.

A big disadvantage of GP models is that it becomes computationally expensive to train these models for high-dimensional spaces [29]. One way of dealing with this is to define a feature space of lower dimension to train the GP, and provide a conversion between input and feature spaces [30]. One specific method is Kriging with Partial Least Squares (KPLS) [31], which has recently been extended to work with discrete variables [32, 33] and is implemented in SMT [18]. Investigations show that using Kriging with Partial Least Squares (KPLS) [31] works well for architecture optimization problems and significantly reduces training time, at a moderate cost of optimizer performance. As a rule-of-thumb, we apply KPLS for problems with more than 10 design variables, and set the number of KPLS components  $n_{kpls}$  to 10 in that case.

**Selecting Infill Points** Infill points are selected by optimizing infill criteria (also known as acquisition functions), and selecting the best one for a wide range of optimization problems influences BO performance significantly. It was shown that using an ensemble of infills performs best. The size of the infill batch should be set to the maximum number of parallel equations:  $n_{batch} = n_{parallel}$ . An infill ensemble combines multiple underlying infills into a multi-objective infill problem and thereby provides two main advantages [34]:

- 1) The selected infill points are a compromise of the underlying infills, thereby mitigating problems with different infill criteria suggesting to explore wildly different parts of the design space [35];
- 2) Since the infill problem itself results in a Pareto front, it is easy to select multiple infill points for batch evaluation without needing to retrain surrogate models [36].

The infill ensemble is built-up of the following underlying infills:

- For single-objective optimization: Lower Confidence Bound (LCB) [37], Expected Improvement (EI) [7], and Probability of Improvement (PoI) [38];
- For multi-objective optimization: Minimum PoI (MPoI) [39] and Minimum Euclidean PoI (MEPoI) [21].

Selecting the best infill points is an optimization problem itself, and with the same design space as the architecture optimization problem. To successfully deal with mixed-discrete hierarchical variables, the following sequential optimization procedure is applied:

- 1) Apply NSGA-II to search the mixed-discrete, hierarchical design space, solving the multi-objective infill problem:

$$\begin{aligned}
 & \text{minimize} && f_{infill,m}(\mathbf{x}) \\
 & \text{w.r.t.} && \mathbf{x}_d, \mathbf{x}_c \\
 & \text{subject to} && \hat{g}_k(\mathbf{x}) \leq 0
 \end{aligned}$$

where  $f_{infill,m}$  represents infill criteria  $m$ ,  $\mathbf{x}_d$  and  $\mathbf{x}_c$  represent the discrete and continuous design variables, and  $\hat{g}_k$  represents the predicted mean of constraint function  $k$ . Infill objectives  $f_{infill,m}$  are normalized and inverted such that they become minimization objectives.

- 2) Select  $n_{batch}$  points  $\mathbf{x}_{sel}$  from the resulting Pareto front according to:
  - If  $n_{batch} = 1$ , select a random point;
  - If  $n_{batch} > 1$ , select the points with the lowest crowding distance as used in NSGA-II [40].
- 3) For each selected point  $\mathbf{x}_{sel,i}$ , improve the active continuous variables by solving following single-objective problem using SLSQP:

$$\begin{aligned}
 & \text{minimize} && f_{impr}(\mathbf{x}) \\
 & \text{w.r.t.} && \mathbf{x}_c \cap \delta(\mathbf{x}_{sel,i}) \\
 & \text{subject to} && \hat{g}_k(\mathbf{x}) \leq 0
 \end{aligned}$$

---

<sup>†</sup><https://smt.readthedocs.io/>

where  $\mathbf{x}_c \cap \delta(\mathbf{x}_{sel,i})$  represents the active continuous design variables at point  $\mathbf{x}_{sel,i}$ , and  $f_{impr}$  the scalar improvement objective:

$$\begin{aligned}\Delta f_{infill,m}(\mathbf{x}) &= f_{infill,m}(\mathbf{x}) - f_{infill,m}(\mathbf{x}_{sel,i}) \\ f_{deviation}(\mathbf{x}) &= 100 (\max \Delta f_{infill}(\mathbf{x}) - \min \Delta f_{infill}(\mathbf{x}))^2 \\ f_{impr}(\mathbf{x}) &= \sum (\Delta f_{infill,m}(\mathbf{x})) + f_{deviation}(\mathbf{x})\end{aligned}$$

This objective promotes improvement in direction of the negative unit vector, so that all underlying infill objectives are improved simultaneously while not deviating too much from the original  $\mathbf{x}_{sel,i}$  to maintain batch diversity.

**Sampling Hierarchical Design Spaces** When sampling hierarchical design space for creating the initial Design of Experiment (DoE), care must be taken that all sections of the design space are sufficiently represented because of a possible large discrepancy between how often individual discrete design variable values appear [6]. For example, the discrete design vectors representing pure jet engines (vs turbofan engines) in the jet engine design problem of BUSSEMAKER ET AL. [10] only make up 0.02% of the design space. This discrepancy can be measured for each discrete design variable by the *rate diversity*  $RD_j$  metric, or at the problem level by the *max rate diversity*  $MRD$ , defined as:

$$Rates_j = \{Rate(j, \delta_j = 0), Rate(j, 0), \dots, Rate(j, N_j - 1)\} \quad (1)$$

$$RD_j = \max Rates_j - \min Rates_j \quad (2)$$

$$MRD = \max_{j \in 1, \dots, n_{x_d}} RD_j \quad (3)$$

with  $j$  the discrete design variable index,  $\delta_j = 0$  denoting the case where design variable  $j$  is inactive,  $N_j$  the number of possible options for discrete design variable  $j$ , and  $Rate(j, value)$  the relative occurrence rate of that value in all valid discrete design vectors. To ensure sufficient representation of design vectors subject to high rate diversities, a hierarchical sampling method is applied that works as follows:

- 1) Generate all possible valid discrete design vectors  $x_{valid, discr}$  and obtain associated activeness information  $\delta$ ;
- 2) Group design vectors by active variables  $x_{act}$ ;
- 3) Weight each group by the number of active variables  $n_{act}$ ;
- 4) Sample discrete design vectors according to the weights of each group;
- 5) Assign values to active continuous variables using Sobol' [41] sampling.

Investigations showed that this sampling method performs well for non-hierarchical problems, and performs better than random or LHS sampling for hierarchical problems. This was confirmed for the situations where the optimum either lies in the largest design vector group or the smallest, showing that groups of all sizes are represented sufficiently.

**Optimizing in Hierarchical Design Spaces** Finally, it was demonstrated that more integration of hierarchical information into the optimization problem yields better results. This mainly translates to the usage of the hierarchical sampling method described above, for which  $x_{valid, discr}$  and activeness information  $\delta$  is needed, and the application of correction and imputation as a repair operator. Correction and imputation ensure that all evaluated design vectors are valid, meaning that they satisfy value constraints and that all inactive design variables have canonical values. Value constraints occur when selecting a value for one design variable restricts the available values for another. For example, in the Apollo mission design problem [42], selecting a total crew size of 2 limits the number of crew members in the lunar module to 1 or 2, whereas a total crew size of 3 means there can be 1, 2, or 3 members assigned to the lunar module. Canonical values as applied to inactive design variables by imputation are 0 for discrete variables and center-domain for continuous variables. The principles of correction and imputation are visualized in Figure 2, showing how the declared design space (consisting of the Cartesian product of all discrete design variable values) is transformed to the valid design space (consisting of unique corrected and imputed design vectors).

Design variable	Values	Encoded	Value constraints and hierarchical activation:
$x_0$	Nr of sources $S$	$\{1, 2\}$	$\{0, 1\}$
$x_1$	Nr of consumers $C$	$\{1, 2\}$	$\{0, 1\}$
$x_2$	Choice of $S$ for $C_0$	$\{S_0, S_1\}$	$\{0, 1\}$
$x_3$	Choice of $S$ for $C_1$	$\{S_0, S_1\}$	$\{0, 1\}$

If there is only one  $S$ , all  $C$  are connected to  $S_0$ :  
if  $x_0 = 0 \rightarrow x_2 = 0$  and  $x_3 = 0$

If there is only one  $C$ , choosing  $S$  for  $C_1$  is irrelevant:  
if  $x_1 = 0 \rightarrow \delta_3 = 0$  ( $x_3$  is inactive)

① Cartesian product of all  $x$  values  
"declared design space"

$i_{dv}$	$x_0$	$x_1$	$x_2$	$x_3$
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

② Apply constraints & activation  
"corrected design space"

$i_{dv}$	$x_0$	$x_1$	$x_2$	$x_3$
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	1	0	0
6	0	1	0	0
7	0	1	0	0
8	0	1	0	0
9	1	0	0	0
10	1	0	0	0
11	1	0	1	0
12	1	0	1	0
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

③ Impute inactive variables  
"imputed design space"

$i_{dv}$	$x_0$	$x_1$	$x_2$	$x_3$
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	1	0	0
6	0	1	0	0
7	0	1	0	0
8	0	1	0	0
9	1	0	0	0
10	1	0	0	0
11	1	0	1	0
12	1	0	1	0
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

④ All unique valid design vectors  
"valid design space"

$i_{dv}$	$x_0$	$x_1$	$x_2$	$x_3$
1	0	0	0	0
2	0	1	0	0
3	1	0	0	0
4	1	0	1	0
5	1	1	0	0
6	1	1	0	1
7	1	1	1	0
8	1	1	1	1

■ corrected

■ inactive

**Fig. 2 Illustration of correction and imputation in hierarchical design spaces, showing how the different sets of design vectors relate to each other, reproduced from [12].**

Correction and imputation are made available as a standalone repair operator so that it can be used during sampling. This is important, because there might be a large difference between the declared design space as defined by the Cartesian product of all design variable values, and the valid design space containing only canonical design vectors. This difference is measured by the *imputation ratio*  $IR$ , defined as:

$$IR_d = \frac{\prod_{j=1}^{n_{x_d}} N_j}{n_{valid, discr}} \quad (4)$$

$$IR_c = \frac{n_{valid, discr} \cdot n_{x_c}}{\sum_{l=1}^{n_{valid, discr}} \sum_{i=1}^{n_{x_c}} \delta_i(\mathbf{x}_{d,l})} \quad (5)$$

$$IR = IR_d \cdot IR_c \quad (6)$$

with  $n_{x_d}$  and  $n_{x_c}$  the number of discrete and continuous design variables, respectively,  $n_{valid, discr}$  the number of valid discrete design vectors,  $\delta_i$  the activeness function for design variable  $i$ , and  $IR_d$  and  $IR_c$  the discrete and continuous imputation ratio, respectively. An imputation ratio of 1 indicates no hierarchy. Higher values represent the ratio between the declared and valid design spaces sizes and thereby design space hierarchy. Imputation ratio can also be interpreted as the amount of random vectors that need to be generated in order to find one valid canonical design vector. For example for the suborbital vehicle design problem in [43] there are  $123e3$  possible architectures, however the Cartesian product of design variables yields  $2.8e6$  design vectors: therefore, the imputation ratio  $IR = 2.8e6/123e3 = 22.8$ . This means that on average for every 22.8 randomly generated design vectors, there will be only one that represents a valid design. Without correction and imputation it will be difficult for an optimization algorithm to generate new valid design vectors due to this effect. This applies to BO as well as to evolutionary optimization algorithms. A problem-agnostic correction algorithm is used that corrects invalid discrete design vectors to the closest valid discrete design vector, as measured by the Euclidean distance. This correction algorithm, however, depends on the availability of  $x_{valid, discr}$ : if this is not available then problem-specific correction should be used.

To conclude, the BO algorithm described in this section and implemented as ARCHSBO in SBARCHOPT<sup>‡</sup> [27] features:

- Hierarchical mixed-discrete Gaussian Process models;
- Use of Kriging with Partial Least Squares (KPLS) to reduce training times for high-dimensional problems;
- Ensemble infill criteria for batch infill generation for single- and multi-objective optimization problems;
- Sequential-optimization for selecting infill points;
- Hierarchical sampling algorithm that groups and weights valid discrete design vectors to deal with rate diversity effects;
- Problem-agnostic similarity-based correction algorithm and the availability of correction and imputation as a repair operator to deal with imputation ratio effects.

## IV. Hidden Constraint Strategies in Bayesian Optimization

This section identifies and investigates various strategies for dealing with hidden constraints in Bayesian Optimization (BO) algorithms. Although we only discuss the integration with BO, the following discussions apply to any Surrogate-Based Optimization (SBO) algorithm in principle.

### A. Literature Review and Strategy Classification

Hidden constraints are encountered in many engineering problems, MÜLLER & DAY [23] provide several examples. They also provide an overview of strategies tried in the past, including for non-BO algorithms. For example, for algorithms that depend on some ranking-based selection procedure rather than surrogate model building, such as evolutionary algorithms, it suffices to apply the extreme barrier approach: replace NaN by  $+\infty$  to prevent these points being selected as parents for creating offspring for the next generation of design points. Local optimization algorithms can deal with hidden constraints by implementing some way to retrace part of their search path. For architecture optimization, however, we only consider method relevant for global optimization algorithms. For surrogate-based algorithms, they mention that one of the simplest methods is to train the surrogate models only on viable points, thereby in effect *rejecting* failed points from the training set. The disadvantage to this approach is that knowledge of the design space is ignored: the optimizer might get stuck suggesting the same infill point(s) over and over, because it cannot know that these infill points will fail to evaluate. A more advanced approach is to *replace* the failed points by some values derived from viable points, as initially suggested by FORRESTER ET AL. [26] and inspired by imputation in the sense of replacing missing data in statistical datasets. They reason that failed points actually represent missing data and can be replaced by values of close-by viable points. However, the replaced values should drive the optimizer towards the viable region of the design space, which leads them to formulate a method for finding replacement values from a Gaussian Process (GP) model trained on the viable points only:

$$y_{replace}(\mathbf{x}_{failed}) = \hat{y}(\mathbf{x}_{failed}) + \alpha \cdot \hat{\sigma}(\mathbf{x}_{failed}) \quad (7)$$

where  $y$  represents the output value to be replaced,  $\hat{y}$  and  $\hat{\sigma}$  the output and uncertainty estimates of the GP trained on viable design points, and  $\alpha$  some multiplier ( $\alpha = 1$  in [26]). They show that their approach works well for a continuous single-objective airfoil optimization problem.

Another strategy for replacing values of failed points is by simply selecting one or more neighbor points and applying some aggregation function to get one value to replace. For example, HUYER & NEUMAIER replace failed values by the max of  $n_{nb}$  neighbor points. This concept can be expanded by considering  $n_{nb} = 1$  to use the closest point to select the replacement value, or  $n_{nb} = n_{viable}$  to consider all viable points for the replacement value.

The most advanced method for dealing with hidden constraints in Surrogate-Based Optimization (SBO) algorithms is to *predict* where the failed region lies with the help of another surrogate model [44]. Here the idea is to assign binary labels to all design point based on their failure status, for example 0 meaning failed and 1 meaning viable, train a model on this data, and then use that model to predict the so-called Probability of Viability (PoV). PoV lies between 0% and 100% and represents the probability that a newly-selected infill point will be viable (i.e. will not fail). During infill optimization, PoV can be used in two ways: either as a penalty multiplier to the infill criterion [44], or as a constraint to the infill algorithm that ensures that PoV of selected infill points is at or above some threshold [45]. Different types of surrogate models have been used to predict PoV, mainly selected due to their ability for modeling such binary classification problems. Used models include Random Forest Classifiers (RFC) [44], piecewise linear Radial

<sup>‡</sup><https://sbarchopt.readthedocs.io/>



**Table 1 Comparison of strategies for dealing with hidden constraints in Bayesian Optimization.**

Strategy	Sub-strategy	Configuration	Parameter(s)
Rejection			
Replacement	Neighborhood [51]	$\left\{ \begin{array}{l} \text{Closest} \\ n\text{-nearest, mean} \\ n\text{-nearest, max} \\ \text{Global, max} \end{array} \right.$	$n$
	Predicted worst [26]		$\alpha$
Prediction	Random Forest Classifier (RFC) [44]		$\left\{ \begin{array}{l} \text{as } f_{infill} \text{ penalty} \\ \text{as } g_{infill} \end{array} \right.$
	Gaussian Process (GP) [47]		
	Support Vector Machine (SVM) [46]		
	Piecewise linear RBF [23]		
	K-Nearest Neighbor (KNN) classifier [49]		

Basis Functions (RBF) [23], Support Vector Machines (SVM) [46], SVM’s with RBF kernel [45], Gaussian Process models [47, 48], and K-Nearest Neighbors (KNN) classifiers [49, 50].

To summarize the preceding discussion, there are three high-level strategies for dealing with hidden constraints in BO, also summarized in Table 1:

- Rejection: ignore failed points when training surrogate models;
- Replacement: replace values of failed points based on values of viable points;
- Prediction: train an additional surrogate model to predict the Probability of Validity (PoV) when searching for infill points.

Replacement can be further subdivided into neighborhood replacement, taking an aggregate over various amounts of closest points, or predicted replacement, where a separate surrogate model trained on viable points is used to predict values of the failed points. Prediction can be subdivided by two aspects: the integration in the infill problem and the predictor model. Integration in the infill problem can either be done as penalty to the infill objective  $f_{infill}$  or as a separate infill constraint  $g_{infill}$  that ensures that  $PoV(\mathbf{x}) \geq PoV_{min}$ . The predictor model can in principle be any binary classification model that also exposes the probability of belonging to one of the two classes in order to express PoV.

**B. Implementation into Bayesian Optimization Algorithms**

The rejection and replacement strategies are implemented into the BO algorithm in a preprocessing step before training the GP models. The training set is separated into a viable and a failed set; for rejection the viable set is then simply discarded, whereas for replacement the points in the failed set are assigned some value for each output (i.e. each objective and constraint value) that is based on the viable points. Afterwards, the GP models are trained and the infill selection process continuous as usual. The integration of the prediction strategy is more involved, as here the infill process itself is modified. As for rejection and replacement, the training set is separated into the viable and failed set. The viable set is then directly used to train the GP models for infill search. The additional surrogate model for Probability of Viability (PoV) prediction is trained with a set containing all points and with binary labels assigned according to the viability status of the points: 0 for failed points and 1 for viable points. The infill optimization problem can then be modified by adding an inequality constraint that ensures that  $PoV(\mathbf{x}) \geq PoV_{min}$ :

$$g_{PoV}(\mathbf{x}) = PoV_{min} - PoV(\mathbf{x}) \leq 0 \tag{8}$$

where  $PoV_{min}$  represents a user-defined minimum PoV to be reached, and  $PoV(\mathbf{x})$  represents the predicted PoV for a given design point. PoV can also be integrated by modifying the infill objectives:

$$f_{m,infill,mod}(\mathbf{x}) = 1 - ((1 - f_{m,infill}(\mathbf{x})) \cdot PoV(\mathbf{x})) \tag{9}$$

where  $f_{m,infill}$  represents the  $m^{th}$  infill objective, and assuming infill objectives are normalized and to be minimized. Figure 3 presents several steps of running BO on a test problem, with an RBF model as PoV predictor used as an infill constraint at  $PoV_{min} = 50\%$ . The test problem is the single-objective problem from ALIMO ET AL. [45], modified to have its optimum at the edge to the failed region near the top left corner of the 2D design space, implemented as ALIMOEDGE in SBARCHOPT<sup>§</sup>. As can be seen, all infill points suggested satisfy the  $g_{PoV}$  constraint, however, in the earlier iterations this constraint can be inaccurate. Several infills are generated that violate the hidden constraint, and after each iteration the model gets more accurate at the edges to the failed regions for three locations in the design space where the optimizer expects the optimum to lie. This visualization additionally demonstrates that the predictor model should be able to handle one or more regions in the design space with closely-spaced failed and viable points.

### C. Comparing Strategy Performances

To compare strategy performance, we use the test problems listed in Table 2 to run the set of hidden constraint strategies listed in Table 3.

**Table 2** Test problems for comparing hidden constraint strategies. All problems are available in the SBARCHOPT library [27]. Abbreviations and symbols: *IR* = imputation ratio, *HC* = hidden constraints, *MD* = mixed-discrete, *H* = hierarchical, *MO* = multi-objective, *FR* = fail rate,  $n_{x_c}$  and  $n_{x_d}$  = number of continuous and discrete design variables, respectively,  $n_f$  = number of objectives,  $n_g$  = number of constraints.

Name	SBARCHOPT Class	Ref.	$n_{x_c}$	$n_{x_d}$	$n_f$	$n_g$	<i>IR</i>	<i>FR</i>
Branin	BRANIN	[52]	2		1			0%
HC Branin	HCBRANIN	[47]	2		1			33%
Alimo	ALIMO	[45]	2		1			51%
Alimo Edge	ALIMOEDGE		2		1			53%
HC Sphere	HCSPHERE	[46]	2		1			51%
Müller 1	MUELLER01	[23]	5		1			67%
Müller 2	MUELLER02	[23]	4		1			40%
HC CantBeam	CANTILEVEREDBEAMHC		4		1	1		83%
HC Carside Less	CARSIDEHCLESS		7		1	9		39%
HC Carside	CARSIDEHC		7		3	8		66%
MD/HC CantBeam	MDCANTILEVEREDBEAMHC		2	2	1	1		81%
MD/HC Carside	MDCARSIDEHC		3	4	3	8		66%
H Alimo	HIERALIMO		2	5	1		5.4	51%
H Alimo Edge	HIERALIMOEDGE		2	5	1		5.4	53%
H Müller 2	HIERMUELLER02		4	4	1		5.4	37%
H/HC Rosenbrock	HIERARCHICALROSENBRÖCKHC	[53]	8	5	1	1	1.5	21%
MO/H/HC Rosenbrock	MOHIERARCHICALROSENBRÖCKHC	[53]	8	5	2	1	1.5	60%

<sup>§</sup><https://sbarchopt.readthedocs.io/>

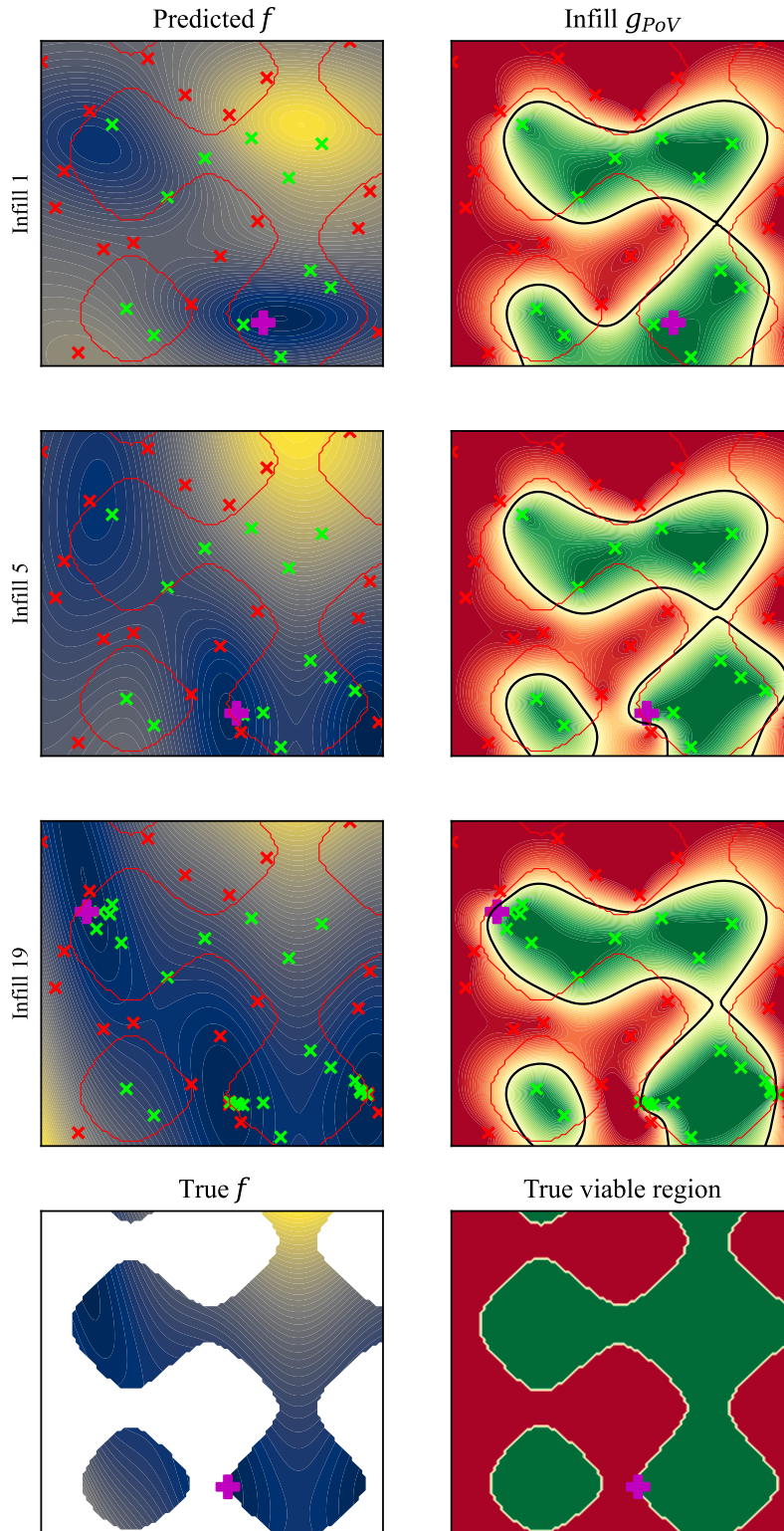


Fig. 3 Several optimization steps between iteration 1 and 20 of BO executed on a test problem with its optimum lying at the edge to the failed region, as shown in the bottom row. The main GP is shown on the left (darker means a lower, more optimal value), and the RBF model for predicting PoV is shown on the right. The RBF model is used as an infill constraint with  $PoV_{min} = 50\%$ , showing green and red contours for satisfied and violated constraint values, respectively. Green, red, and magenta points represent viable, failed, and selected infill points, respectively.

Compared to the previously identified predictor models, we additionally test a Variational GP and the mixed-discrete GP developed in [54]. A Variational GP does not assume a Gaussian distribution and therefore might be able to more accurately model discontinuous functions as seen in classification problems [55]. We use the implementation provided in TRIESTE<sup>¶</sup> [56]. Through a quick pre-study, we additionally discard Support Vector Machine (SVM) and piecewise linear RBF classifiers due to their bad performance in higher-dimensional and mixed-discrete test problems.

**Table 3** Tested Bayesian Optimization hidden constraint strategies. Abbreviations: GP = Gaussian Process.

Strategy	Sub-strategy	Configuration	Model Implementation
Rejection			
Replacement	Neighborhood	Global, max	
		Local	
		5-nearest, max	
		5-nearest, mean	
		Predicted worst	$\alpha = 1$
		$\alpha = 2$	SMT*
Prediction	Random Forest Classifier	$PoV_{min} = 50\%$	scikit-learn <sup>†</sup>
	K-Nearest Neighbors	$PoV_{min} = 50\%$	scikit-learn <sup>†</sup>
	Radial Basis Function (RBF)	$PoV_{min} = 50\%$	Scipy <sup>‡</sup>
	Gaussian Process Classifier	$PoV_{min} = 50\%$	scikit-learn <sup>†</sup>
	Variational GP	$PoV_{min} = 50\%$	Trieste <sup>§</sup>
	Mixed-discrete GP	$PoV_{min} = 50\%$	SMT*

To ensure there are enough viable points to train models on for the infill search, the DoE size of problems containing hidden constraints should be increased:

$$n_{doe} = \frac{k_{doe} \cdot n_x}{1 - FR_{exp}} \quad (10)$$

where  $k_{doe}$  is the DoE multiplier,  $n_x$  the number of design variables, and  $FR_{exp}$  the expected fail rate. We assume an expected fail rate of 60% and use  $k_{doe} = 2$  for following tests. Each optimization is executed with  $n_{infill} = 50$  and is repeated 8 times. Optimization performance is compared using  $\Delta HV$  regret, a measure that represents the distance to the known optimum or Pareto-front ( $\Delta HV$ ) cumulatively over the course of an optimization (regret). Lower  $\Delta HV$  regret is better, as it shows that the optimum was approached more closely and/or reached sooner. Strategies are then ranked by the ranking procedure outlined in [12].

Table 4 presents optimization performance results. It can be seen that prediction with a Random Forest Classifier (RFC) performs best (rank 1) or good (rank  $\leq 2$ ) more compared to other strategies, with mixed-discrete (MD) GP prediction and predicted worst replacement ( $\alpha = 1$ ) closely following. Table 5 presents some performance measures averaged over all test problems, relative to the rejection strategy. A reduction in  $\Delta HV$  regret is usually combined with a reduction in failure rate: this makes sense, as a lower failure rate indicates a better capability of avoiding the failed region and therefore more resources spent on exploring the viable region. Also shown is the change in training and infill times. Replacement strategies approximately double the training and infill cycle time, due to the fact that the trained GP models (one for each  $f$  and  $g$ ) contain the complete set of design points as training values, whereas for rejection and prediction only the viable points are used. Prediction strategies train an extra model to predict the PoV, which leads to a moderate increase in training and infill time. Predicted worst replacement, prediction with RFC, and prediction with MD GP are selected as most promising candidates. These three strategies are further investigated to find out the influence of their parameter settings:  $\alpha$  for predicted worst replacement, and  $PoV_{min}$  for the prediction strategies. In addition, the prediction strategies are tested with integration as  $f$ -infill penalty (Eq. (9)). Tests are run with the same settings as the previous experiment.

<sup>¶</sup><https://secondmind-labs.github.io/trieste/>

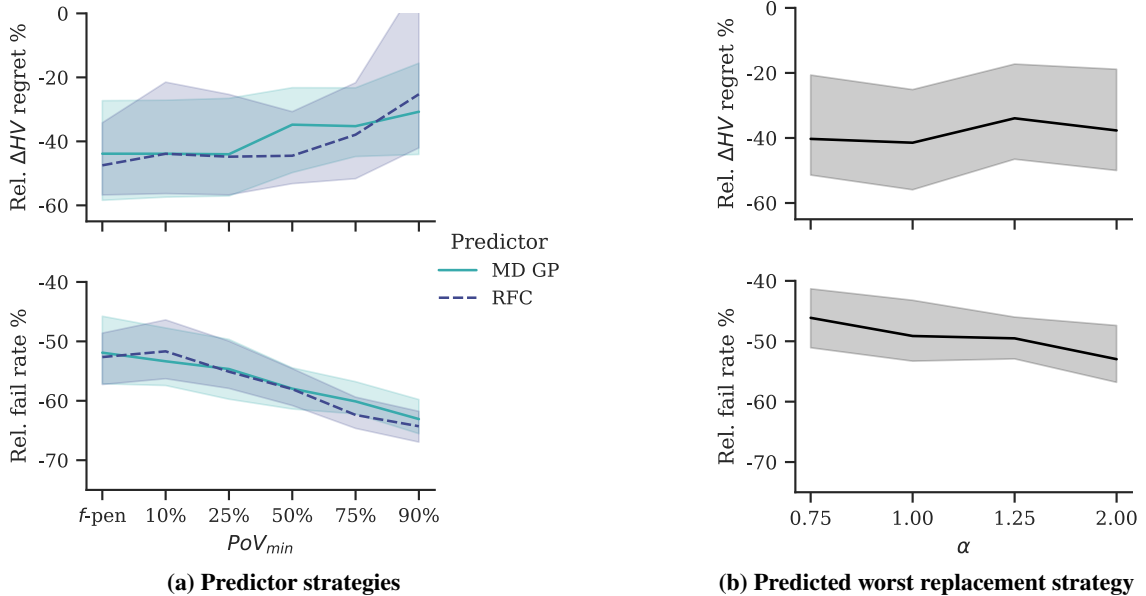
**Table 4 Comparison of hidden constraint strategies on various test problems, ranked by  $\Delta HV$  regret (lower rank / darker color is better). Best performing strategy is underlined. Abbreviations: HC = hidden constraints, MD = mixed-discrete, H = hierarchical, MO = multi-objective, RFC = Random Forest Classifier, KNN = K-Nearest Neighbors, RBF = Radial Basis Functions, GP = Gaussian Process.**

	Branin	HC Branin	Alimo	Alimo Edge	Müller 2	HC Sphere	Müller 1	HC CantB	HC Carside Less
Rejection	1	2	3	3	1	2	2	4	5
Replacement	1	2	2	3	1	2	1	1	3
Replacement	2	2	2	3	3	1	3	3	4
Replacement	1	2	1	2	2	1	2	2	3
Replacement	2	2	1	2	2	1	2	3	2
Replacement	1	1	2	2	2	1	1	1	2
Replacement	1	1	2	2	1	2	1	2	2
Prediction	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>2</u>	<u>1</u>	<u>1</u>	<u>2</u>
Prediction	1	1	2	2	1	2	2	2	2
Prediction	1	1	1	1	1	2	3	2	1
Prediction	1	1	1	1	1	2	1	1	3
Prediction	1	1	2	1	1	2	3	1	3
Prediction	1	1	1	1	1	2	2	2	1

HC Carside	MD/HC CantB	MD/HC Carside	H Alimo	H Alimo Edge	H Müller 2	H/HC Rosenbr.	MO/H/HC Rbr.	Rank 1	Rank $\leq 2$
4	6	5	4	4	3	1	2	18%	41%
3	4	3	1	2	1	2	3	35%	65%
4	5	4	3	3	3	1	3	12%	29%
3	3	2	1	2	2	1	2	29%	82%
1	5	2	2	2	1	1	1	35%	88%
1	1	1	2	1	2	1	1	65%	100%
2	2	1	1	3	1	1	2	47%	94%
2	2	1	1	1	2	1	1	71%	100%
3	2	3	3	4	1	1	1	35%	76%
2	3	1	1	3	2	1	1	59%	82%
3	2	2	2	3	2	1	2	47%	82%
3	1	2	1	3	1	1	1	59%	76%
1	2	1	1	1	2	1	3	65%	94%

**Table 5 Performance of hidden constraint strategies relative to the rejection strategy, averaged over all test problems at the end of the optimization runs. Darker color is better for  $\Delta HV$  regret and Fail rate; worse for time columns. Abbreviations: HV = hypervolume, RFC = Random Forest Classifier, KNN = K-Nearest Neighbors, RBF = Radial Basis Functions, GP = Gaussian Process.**

Strategy	Sub-strategy	$\Delta HV$ regret	Fail rate	Training time	Infill time	Training + infill time
Rejection		+0%	+0%	+0%	+0%	+0%
Replacement	Global max	-20%	-63%	+176%	+87%	+107%
Replacement	Local	-1%	-14%	+174%	+99%	+109%
Replacement	5-nearest, max	-34%	-46%	+171%	+89%	+102%
Replacement	5-nearest, mean	-32%	-26%	+174%	+92%	+103%
Replacement	Predicted worst	-42%	-47%	+171%	+78%	+99%
Replacement	Pred. worst ( $\alpha = 2$ )	-37%	-53%	+167%	+72%	+96%
Prediction	RFC	-41%	-61%	+81%	+118%	+102%
Prediction	KNN	-29%	-39%	+39%	+55%	+43%
Prediction	RBF	-38%	-62%	+73%	+58%	+54%
Prediction	GP Classifier	-36%	-58%	+77%	+86%	+70%
Prediction	Variational GP	-38%	-61%	+60%	+259%	+187%
Prediction	MD GP	-37%	-63%	+85%	+67%	+68%



**Fig. 4 Comparison of hidden constraint strategy settings relative to the rejection strategy, averaged over all test problems at the end of the optimization runs. Abbreviations: HV = hypervolume, MD = mixed-discrete, GP = Gaussian Process, RFC = Random Forest Classifier.**

**Table 6 Comparison of hidden constraint strategy settings on various test problems, ranked by  $\Delta HV$  regret (lower rank / darker color is better). Best performing strategy is underlined. Abbreviations: HC = hidden constraints, MD = mixed-discrete, H = hierarchical, MO = multi-objective, RFC = Random Forest Classifier, KNN = K-Nearest Neighbors, RBF = Radial Basis Functions, GP = Gaussian Process.**

	Alimo	Alimo Edge	Mtiller 1	MD/HC Carside	H Alimo	H Müller 2	H/HC Rbr.	MO/H/HC Rbr.	Rank 1	Rank $\leq 2$
Predicted Worst	<u>1</u>	2	1	1	2	3	1	2	50%	88%
Predicted Worst	1	2	1	1	2	2	1	2	50%	100%
Predicted Worst	2	3	2	2	2	2	1	2	12%	88%
Predicted Worst	1	3	1	1	2	2	1	3	50%	75%
<u>MD-GP</u>	<u>1</u>	2	1	1	2	1	2	1	62%	100%
<u>MD-GP</u>	<u>1</u>	1	2	1	2	1	1	2	62%	100%
<u>MD-GP</u>	<u>1</u>	1	1	1	2	1	2	2	62%	100%
MD-GP	1	2	1	1	2	1	1	4	62%	88%
MD-GP	1	2	1	1	3	1	1	4	62%	75%
MD-GP	2	2	1	2	2	1	2	4	25%	88%
RFC	1	1	3	3	1	1	1	1	75%	75%
RFC	1	1	1	2	2	1	1	2	62%	100%
RFC	1	1	3	2	1	1	1	2	62%	88%
RFC	1	1	2	2	2	1	2	2	38%	100%
RFC	1	2	3	2	2	2	2	2	12%	88%
RFC	1	3	4	2	2	1	1	4	38%	62%

Table 6 presents ranking of algorithm performance and Figure 4 shows the relative improvement over the rejection strategy for the tested strategy configurations. Fail rate is reduced significantly for higher values of  $PoV_{min}$  and  $\alpha$ , which is expected as higher values result in a more conservative approach and therefore less exploration of the failed region. A reduction in failure rate, however, decreases optimizer performance (seen by an increase in  $\Delta HV$  regret), showing that a certain amount of failed evaluations is needed to sufficiently explore the design space. The best performing strategies are the prediction strategies at low  $PoV_{min}$  values. It also shows that  $f$ -infill penalty behaves similar as low  $PoV_{min}$  values. Prediction with MD GP or RFC are behaving similarly-well, although MD GP for a little wider range of  $PoV_{min}$  than RFC. From these results, we recommend that either MD GP or RFC prediction should be used to deal with hidden constraints in Bayesian Optimization. PoV should be integrated as a constraint, because it allows more control over exploration vs exploitation compared to integration as  $f$ -infill penalty.  $PoV_{min}$  should be kept relatively low to promote sufficient exploration: a value of  $PoV_{min} = 25\%$  will be used subsequently.

## V. Application: Jet Engine Architecture Optimization

To demonstrate the application of the hidden constraint optimization strategies presented in this work, the open-source jet engine optimization problem presented in [10] is solved using the BO algorithm. The problem features several architectural choices, such as whether to include a fan and bypass flow, the number of stages, the addition of a gearbox, and where to apply bleed and power offtakes. Each possible architecture is evaluated using a pyCycle [57] OpenMDAO [58] problem. The Multidisciplinary Design Optimization (MDO) problem is automatically constructed from a class-based definition, easing the integration with architecture generation platforms. Next to thermodynamic cycle analysis disciplines, additional disciplines include weight estimation, length and diameter calculation, and noise and NOx estimations. We solve the bi-objective realistic jet engine problem, featuring 11 discrete and 30 continuous variables ( $n_x = 41$ ), 2 objectives (Thrust-Specific Fuel Consumption and weight minimization), and 15 design constraints. The imputation ratio  $IR$  is 9.1 and there are a little over 142 thousand valid discrete design vectors. Approximately 65% of evaluations fail in a DoE, showing the clear need for a hidden constraint optimization strategy. All code is available open source, for more information the reader is referred to [10].

The BO algorithm (as ARCHSBO) and test problem (as REALISTICTURBOFANARCH) have been implemented in the SBARCHOPT<sup>||</sup> library [27]: an open-source Python library for solving system architecture optimization problems. The library features a problem definition interface, hierarchical sampling and correction algorithms, bindings to optimization algorithms, and a test problem database. The BO algorithm is executed 10 times with  $n_{doe} = 510$ ,  $n_{infill} = 400$  (total budget is 910 points), and  $n_{batch} = 4$ . KPLS is applied with  $n_{kpls} = 10$ . Both hidden constraint strategies use  $PoV_{min} = 25\%$ .

Figure 5 presents the results of the jet engine optimization. The MD GP predictor clearly outperforms the RFC predictor: it progresses quicker towards the known optimum and finds a better optimum value after the 400 infill points. The optimum has been found in 77% less function evaluations compared to NSGA-II, where a budget of 4000 evaluations was used in [10].

## VI. Conclusions and Outlook

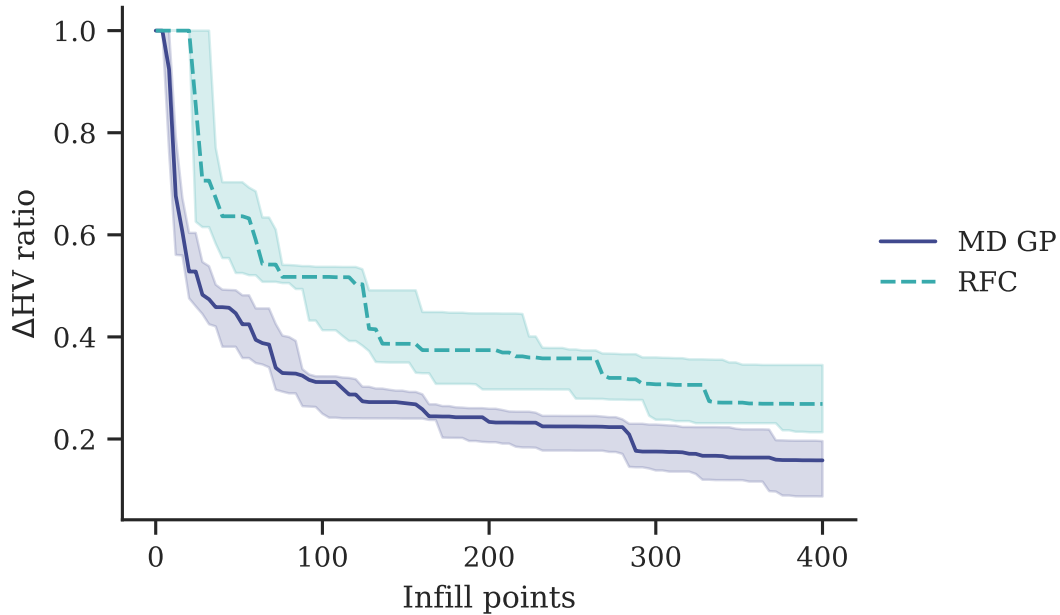
System architecture optimization problems are challenging to solve due to their mixed-discrete and hierarchical design spaces combined with constrained, multi-objective, black-box, and expensive-to-evaluate solution spaces that potentially are subject to hidden constraints. Bayesian Optimization (BO) algorithms are gradient-free, global optimization algorithms that are specifically developed to optimize with a small evaluation budget. A BO algorithm is presented that has been developed for optimization in hierarchical design spaces, featuring:

- Hierarchical mixed-discrete Gaussian Process models;
- Use of Kriging with Partial Least Squares (KPLS) to reduce training times for high-dimensional problems;
- Ensemble infill criteria for batch infill generation for single- and multi-objective optimization problems;
- Sequential optimization for selecting infill points;
- Hierarchical sampling algorithm that groups and weights valid discrete design vectors to deal with rate diversity effects;
- Problem-agnostic similarity-based correction algorithm and the availability of correction and imputation as a repair operator to deal with imputation ratio effects.

---

<sup>||</sup> <https://sbarchopt.readthedocs.io/>





**Fig. 5 Optimization progression of the BO algorithm with the prediction hidden constraint strategy with two different predictors for the jet engine optimization problem. Abbreviations: MD = mixed-discrete, GP = Gaussian Process, RFC = Random Forest Classifier.**

The presented algorithm is extended to deal with hidden constraints. First, three high-level strategies are identified: rejection (ignore failed points), replacement (replace output values of failed points by some value derived from viable points), and prediction (predict the Probability of Viability). Replacement strategies are subdivided into replacement from neighboring points and replacement by prediction. Prediction strategies are subdivided based on the used classification models. It is demonstrated how prediction strategies can be implemented in BO, either as a penalty to the infill objectives or as additional inequality constraints to the infill problem parameterized by a lower threshold  $PoV_{min}$ . Then, multiple hidden constraint strategies are tested on a database of test problems. It is shown that predicted worst replacement, prediction with a Random Forest Classifier (RFC), and prediction with a mixed-discrete Gaussian Process (GP) perform best. Compared to rejection, training and infill times are increased due to the increase in training points for replacement strategies and training of the PoV prediction model for prediction strategies. These three strategies are further tested with different values of  $PoV_{min}$  (for prediction) and  $\alpha$  for predicted worst replacement. It is shown that prediction with MD GP and RFC perform best, and with lower  $PoV_{min}$  values. A default value of  $PoV_{min} = 25\%$  is recommended. Finally, a jet engine architecture optimization problem is solved with the two predictor strategies, showing that prediction with MD GP performs best. The extended BO algorithm and all used test problems are available in the open-source SBARCHOPT library.

This work has shown that hidden constraints can be effectively dealt with by BO algorithms in the context of system architecture optimization. To validate its applicability, BO should be applied to more aerospace and non-aerospace architecture optimization problems. In future work the use of BO for architecture optimization should be expanded, specifically to larger design space sizes with tens to hundreds of design variables [59], for example using dimension reduction techniques such as EGORSE [29], and multi-fidelity BO should be considered as a way to make BO more effective and further reduce required computational resources.

To support users in applying the developed algorithms for architecture optimization, it should be made easy to select, configure, and execute optimizations directly from system architecting tools. For example, architecture problems formulated using ADORE [5] could be translated to an SBARCHOPT problem definition without additional user effort. From there, any algorithm implemented in SBARCHOPT could be selected and executed to solve the defined optimization problem.

## Acknowledgments

The research presented in this paper has been performed in the framework of the COLOSSUS project (Collaborative System of Systems Exploration of Aviation Products, Services and Business Models) and has received funding from the European Union Horizon Europe Programme under grant agreement n° 101097120. The authors would like to recognize Luca Boggero and Björn Nagel for supporting the secondment of Jasper Bussemaker at ONERA DTIS in Toulouse, during which the research presented in this paper was mostly produced.

## References

- [1] Judt, D., and Lawson, C., “Development of an automated aircraft subsystem architecture generation and analysis tool,” *Engineering Computations*, Vol. 33, No. 5, 2016, pp. 1327–1352. <https://doi.org/10.1108/EC-02-2014-0033>.
- [2] Iacobucci, J., “Rapid Architecture Alternative Modeling (Raam): a Framework for Capability-Based Analysis of System of Systems Architectures,” Ph.D. thesis, Georgia Institute of Technology, 2012.
- [3] McDermott, T., Folds, D., and Hallo, L., “Addressing Cognitive Bias in Systems Engineering Teams,” *30th Annual INCOSE International Symposium*, Virtual Event, 2020. <https://doi.org/10.1002/j.2334-5837.2020.00721.x>.
- [4] Bussemaker, J. H., and Ciampa, P., “MBSE in Architecture Design Space Exploration,” *Handbook of Model-Based Systems Engineering*, edited by A. Madni, N. Augustine, and M. Sievers, Springer, Switzerland, 2022. [https://doi.org/10.1007/978-3-030-27486-3\\_36-1](https://doi.org/10.1007/978-3-030-27486-3_36-1).
- [5] Bussemaker, J. H., Boggero, L., and Ciampa, P. D., “From System Architecting to System Design and Optimization: A Link Between MBSE and MDAO,” *32nd Annual INCOSE International Symposium*, Detroit, MI, USA, 2022. <https://doi.org/10.1002/iis2.12935>.
- [6] Crawley, E., Cameron, B., and Selva, D., *System architecture: strategy and product development for complex systems*, Pearson Education, England, 2015. <https://doi.org/10.1007/978-1-4020-4399-4>.
- [7] Jones, D., Schonlau, M., and Welch, W., “Efficient Global Optimization of Expensive Black-Box Functions,” *Journal of Global Optimization*, Vol. 13, 1998, pp. 455–492. <https://doi.org/10.1023/A:1008306431147>.
- [8] Zaefferer, M., and Horn, D., “A First Analysis of Kernels for Kriging-Based Optimization in Hierarchical Search Spaces,” *Parallel Problem Solving from Nature, PPSN XI*, Vol. 1, Springer Berlin Heidelberg, Berlin, Heidelberg, 2018, pp. 399–410. [https://doi.org/10.1007/978-3-319-99259-4\\_32](https://doi.org/10.1007/978-3-319-99259-4_32).
- [9] Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E., and Guerin, Y., “Bayesian optimization of variable-size design space problems,” *Optimization and Engineering*, 2020. <https://doi.org/10.1007/s11081-020-09520-z>.
- [10] Bussemaker, J. H., De Smedt, T., La Rocca, G., Ciampa, P. D., and Nagel, B., “System Architecture Optimization: An Open Source Multidisciplinary Aircraft Jet Engine Architecting Problem,” *AIAA AVIATION 2021 FORUM*, Virtual Event, 2021. <https://doi.org/10.2514/6.2021-3078>.
- [11] Gamot, J., Balesdent, M., Tremolet, A., Wuilbercq, R., Melab, N., and Talbi, E.-G., “Hidden-variables genetic algorithm for variable-size design space optimal layout problems with application to aerospace vehicles,” *Engineering Applications of Artificial Intelligence*, Vol. 121, 2023, p. 105941.
- [12] Bussemaker, J., Saves, P., Bartoli, N., Lefebvre, T., Lafage, R., Boggero, L., and Nagel, B., “System Architecture Optimization Strategies: Dealing with Expensive Hierarchical Multi-Objective Problems,” *Journal of Structural and Multidisciplinary Optimization*, 2024.
- [13] Garrido-Merchán, E., and Hernández-Lobato, D., “Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes,” *Neurocomputing*, Vol. 380, 2020, pp. 20–35. <https://doi.org/10.1016/j.neucom.2019.11.004>.
- [14] Schonlau, M., Welch, W. J., and Jones, D. R., “Global versus local search in constrained optimization of computer models,” *Lecture Notes - Monograph Series*, Institute of Mathematical Statistics, online, 1998, pp. 11–25. <https://doi.org/10.1214/lnms/1215456182>.
- [15] Sasena, M., “Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations,” Ph.D. thesis, University of Michigan, 2002.

- [16] Zuniga, M., and Sinoquet, D., “Global optimization for mixed categorical-continuous variables based on Gaussian process models with a randomized categorical space exploration step,” *INFOR: Information Systems and Operational Research*, Vol. 58, No. 2, 2020, pp. 310–341. <https://doi.org/10.1080/03155986.2020.1730677>.
- [17] Audet, C., Hallé-Hannan, E., and Digabel, S. L., “A general mathematical framework for constrained mixed-variable blackbox optimization problems with meta and categorical variables,” 2022. <https://doi.org/10.48550/ARXIV.2204.00886>.
- [18] Saves, P., Lafage, R., Bartoli, N., Diouane, Y., Bussemaker, J. H., Lefebvre, T., Hwang, J. T., Morlier, J., and Martins, J. R. R. A., “SMT 2.0: A Surrogate Modeling Toolbox with a focus on Hierarchical and Mixed Variables Gaussian Processes,” *Advances in Engineering Software*, Vol. 188, 2024, p. 103571. <https://doi.org/https://doi.org/10.1016/j.advengsoft.2023.103571>.
- [19] Knowles, J., “ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems,” *IEEE Transactions on Evolutionary Computation*, Vol. 10, No. 1, 2006, pp. 50–66. <https://doi.org/10.1109/TEVC.2005.851274>.
- [20] Rojas-Gonzalez, S., and Van Nieuwenhuysse, I., “A Survey on Kriging-Based Infill Algorithms for Multiobjective Simulation Optimization,” *Computers & Operations Research*, Vol. 116, 2019, p. 104869. <https://doi.org/10.1016/j.cor.2019.104869>.
- [21] Bussemaker, J. H., Bartoli, N., Lefebvre, T., Ciampa, P. D., and Nagel, B., “Effectiveness of Surrogate-Based Optimization Algorithms for System Architecture Optimization,” *AIAA AVIATION 2021 FORUM*, Virtual Event, 2021. <https://doi.org/10.2514/6.2021-3095>.
- [22] Bussemaker, J. H., García Sánchez, R., Fouda, M., Boggero, L., and Nagel, B., “Function-Based Architecture Optimization: An Application to Hybrid-Electric Propulsion Systems,” *33rd Annual INCOSE International Symposium*, Honolulu, HI, USA, 2023. <https://doi.org/10.1002/iis2.13020>.
- [23] Müller, J., and Day, M., “Surrogate Optimization of Computationally Expensive Black-Box Problems with Hidden Constraints,” *INFORMS Journal on Computing*, Vol. 31, No. 4, 2019, pp. 689–702. <https://doi.org/10.1287/ijoc.2018.0864>.
- [24] Le Digabel, S., and Wild, S. M., “A taxonomy of constraints in black-box simulation-based optimization,” *Optimization and Engineering*, 2023. <https://doi.org/10.1007/s11081-023-09839-3>.
- [25] Krengel, M., and Hepperle, M., “Effects of Wing Elasticity and Basic Load Alleviation on Conceptual Aircraft Designs,” *AIAA SCITECH 2022 Forum*, 2022. <https://doi.org/10.2514/6.2022-0126>.
- [26] Forrester, A. I., Sóbester, A., and Keane, A. J., “Optimization with missing data,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Vol. 462, No. 2067, 2006, pp. 935–945. <https://doi.org/10.1098/rspa.2005.1608>.
- [27] Bussemaker, J. H., “SBArchOpt: Surrogate-Based Architecture Optimization,” *Journal of Open Source Software*, Vol. 8, No. 89, 2023, p. 5564. <https://doi.org/10.21105/joss.05564>.
- [28] Saves, P., Diouane, Y., Bartoli, N., Lefebvre, T., and Morlier, J., “A mixed-categorical correlation kernel for Gaussian process,” *Neurocomputing*, Vol. 550, 2023, p. 126472. <https://doi.org/10.1016/j.neucom.2023.126472>.
- [29] Priem, R., Bartoli, N., Diouane, Y., Dubreuil, S., and Saves, P., “High-dimensional efficient global optimization using both random and supervised embeddings,” *AIAA AVIATION 2023 Forum*, American Institute of Aeronautics and Astronautics, San Diego, CA, USA, 2023. <https://doi.org/10.2514/6.2023-4448>.
- [30] Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P., “Manifold Gaussian Processes for regression,” *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Vancouver, CA, 2016. <https://doi.org/10.1109/ijcnn.2016.7727626>.
- [31] Bouhleb, M., Bartoli, N., Otsmane, A., and Morlier, J., “Improving Kriging surrogates of high-dimensional design models by Partial Least Squares dimension reduction,” *Structural and Multidisciplinary Optimization*, Vol. 53, No. 5, 2016, pp. 935–952. <https://doi.org/10.1007/s00158-015-1395-9>.
- [32] Saves, P., Bartoli, N., Diouane, Y., Lefebvre, T., Morlier, J., David, C., Van, E. N., and Defoort, S., “Multidisciplinary design optimization with mixed categorical variables for aircraft design,” *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, San Diego, CA, USA, 2022. <https://doi.org/10.2514/6.2022-0082>.
- [33] Charayron, R., Lefebvre, T., Bartoli, N., and Morlier, J., “Multi-fidelity Bayesian optimization strategy applied to Overall Drone Design,” *AIAA SCITECH 2023 Forum*, American Institute of Aeronautics and Astronautics, National Harbor, MD, USA, 2023. <https://doi.org/10.2514/6.2023-2366>.

- [34] Cowen-Rivers, A. I., Lyu, W., Tutunov, R., Wang, Z., Grosnit, A., Griffiths, R. R., Maraval, A. M., Jianye, H., Wang, J., Peters, J., and Ammar, H. B., “HEBO: Pushing The Limits of Sample-Efficient Hyperparameter Optimisation,” 2020. <https://doi.org/10.48550/ARXIV.2012.03826>.
- [35] Lyu, W., Yang, F., Yan, C., Zhou, D., and Zeng, X., “Batch Bayesian Optimization via Multi-objective Acquisition Ensemble for Automated Analog Circuit Design,” *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Stockholm, SE, 2018.
- [36] Garnett, R., *Bayesian Optimization*, Cambridge University Press, Cambridge, UK, 2023. <https://doi.org/10.1017/9781108348973>.
- [37] Cox, D., and John, S., “A statistical method for global optimization,” *1992 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, Chicago, IL, USA, 1992. <https://doi.org/10.1109/icsmc.1992.271617>.
- [38] Hawe, G., and Sykulski, J., “An Enhanced Probability of Improvement Utility Function for Locating Pareto Optimal Solutions,” *16th Conference on the Computation of Electromagnetic Fields, COMPUMAG, Aachen, Germany*, Vol. 3, 2007, pp. 965–966.
- [39] Rahat, A., Everson, R., and Fieldsend, J., “Alternative infill strategies for expensive multi-objective optimisation,” *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '17*, ACM Press, New York, USA, 2017, pp. 873–880. <https://doi.org/10.1145/3071178.3071276>.
- [40] Deb, K., “Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction,” *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, Springer, London, UK, 2011, pp. 3–34. [https://doi.org/10.1007/978-0-85729-652-8\\_1](https://doi.org/10.1007/978-0-85729-652-8_1).
- [41] Renardy, M., Joslyn, L. R., Millar, J. A., and Kirschner, D. E., “To Sobol or not to Sobol? The effects of sampling schemes in systems biology applications,” *Mathematical Biosciences*, Vol. 337, 2021, p. 108593. <https://doi.org/10.1016/j.mbs.2021.108593>.
- [42] Simmons, W., “A Framework for Decision Support in Systems Architecting,” Ph.D. thesis, Massachusetts Institute of Technology, 2008.
- [43] Frank, C., Marlier, R., Pinon-Fischer, O., and Mavris, D., “An Evolutionary Multi-Architecture Multi-Objective Optimization Algorithm for Design Space Exploration,” *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Reston, Virginia, 2016, pp. 1–19. <https://doi.org/10.2514/6.2016-0414>.
- [44] Lee, H., Gramacy, R., Linkletter, C., and Gray, G., “Optimization Subject to Hidden Constraints via Statistical Emulation,” techreport UCSC-SOE-10-10, UC Santa Cruz, Apr. 2010.
- [45] Alimo, S. R., Beyhaghi, P., and Bewley, T. R., “Delaunay-Based Global Optimization in Nonconvex Domains Defined by Hidden Constraints,” *Computational Methods in Applied Sciences*, Springer International Publishing, 2018, pp. 261–271. [https://doi.org/10.1007/978-3-319-89890-2\\_17](https://doi.org/10.1007/978-3-319-89890-2_17).
- [46] Sacher, M., Duvigneau, R., Maître, O. L., Durand, M., Berrini, É., Hauville, F., and Astolfi, J.-A., “A classification approach to efficient global optimization in presence of non-computable domains,” *Structural and Multidisciplinary Optimization*, Vol. 58, No. 4, 2018, pp. 1537–1557. <https://doi.org/10.1007/s00158-018-1981-8>.
- [47] Gelbart, M. A., Snoek, J., and Adams, R. P., “Bayesian Optimization with Unknown Constraints,” *UAI'14: Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, Arlington, VA, USA, 2014.
- [48] Bachoc, F., Helbert, C., and Picheny, V., “Gaussian process optimization with failures: classification and convergence proof,” *Journal of Global Optimization*, Vol. 78, No. 3, 2020, pp. 483–506. <https://doi.org/10.1007/s10898-020-00920-0>.
- [49] Audet, C., Caporossi, G., and Jacquet, S., “Binary, unrelaxable and hidden constraints in blackbox optimization,” *Operations Research Letters*, Vol. 48, No. 4, 2020, pp. 467–471. <https://doi.org/10.1016/j.orl.2020.05.011>.
- [50] Tfaily, A., Kokkolaras, M., Bartoli, N., and Diouane, Y., “Efficient Acquisition Functions for Bayesian Optimization in the Presence of Hidden Constraints,” *AIAA Aviation 2023 Forum*, San Diego, USA, 2023. <https://doi.org/10.2514/6.2023-4261>.
- [51] Huyer, W., and Neumaier, A., “SNOBFIT – Stable Noisy Optimization by Branch and Fit,” *ACM Transactions on Mathematical Software*, Vol. 35, No. 2, 2008, pp. 1–25. <https://doi.org/10.1145/1377612.1377613>.
- [52] Forrester, A. I. J., Sóbester, A., and Keane, A. J., *Engineering Design via Surrogate Modelling*, 2008. <https://doi.org/10.1002/9780470770801>.

- [53] Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E., and Guerin, Y., “Overview and Comparison of Gaussian Process-Based Surrogate Models for Mixed Continuous and Discrete Variables: Application on Aerospace Design Problems,” *High-Performance Simulation-Based Optimization*, Studies in Computational Intelligence, Vol. 833, Springer International Publishing, Cham, 2020, pp. 189–224. [https://doi.org/10.1007/978-3-030-18764-4\\_9](https://doi.org/10.1007/978-3-030-18764-4_9).
- [54] Saves, P., Diouane, Y., Bartoli, N., Lefebvre, T., and Morlier, J., “A general square exponential kernel to handle mixed-categorical variables for Gaussian process,” *AIAA AVIATION 2022 Forum*, American Institute of Aeronautics and Astronautics, Chicago, IL, USA, 2022. <https://doi.org/10.2514/6.2022-3870>.
- [55] Hensman, J., Matthews, A., and Ghahramani, Z., “Scalable Variational Gaussian Process Classification,” *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 38, PMLR, San Diego, California, USA, 2015, pp. 351–360.
- [56] Picheny, V., Berkeley, J., Moss, H. B., Stojic, H., Granta, U., Ober, S. W., Artemev, A., Ghani, K., Goodall, A., Paleyes, A., Vakili, S., Pascual-Diaz, S., Markou, S., Qing, J., Loka, N. R. B. S., and Couckuyt, I., “Trieste: Efficiently Exploring The Depths of Black-box Functions with TensorFlow,” 2023. <https://doi.org/10.48550/ARXIV.2302.08436>.
- [57] Hendricks, E., and Gray, J., “pyCycle: A Tool for Efficient Optimization of Gas Turbine Engine Cycles,” *Aerospace*, Vol. 6, No. 8, 2019, p. 87. <https://doi.org/10.3390/aerospace6080087>.
- [58] Gray, J., Hwang, J., Martins, J., Moore, K., and Naylor, B., “OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization,” *Structural and Multidisciplinary Optimization*, Vol. 59, No. 4, 2019, pp. 1075–1104. <https://doi.org/10.1007/s00158-019-02211-z>.
- [59] Donelli, G., Ciampa, P. D., Mello, J. M., Odaguil, F. I., Cuco, A. P., and van der Laan, T., “A Value-driven Concurrent Approach for Aircraft Design-Manufacturing-Supply Chain,” *Production amp; Manufacturing Research*, Vol. 11, No. 1, 2023. <https://doi.org/10.1080/21693277.2023.2279709>.