Title

A taxonomy of constraints in black-box simulation-based optimization

Permalink

https://escholarship.org/uc/item/8kf319cb

Journal

Optimization and Engineering, 25(2)

ISSN

1389-4420

Authors

Le Digabel, Sébastien
Wild, Stefan M

Publication Date

2024-06-01

DOI

10.1007/s11081-023-09839-3

Peer reviewed

A Taxonomy of Constraints in Black-Box Simulation-Based Optimization

Sébastien Le Digabel · Stefan M. Wild

Abstract The types of constraints encountered in black-box simulation-based optimization problems differ significantly from those addressed in nonlinear programming. We introduce a characterization of constraints to address this situation. We provide formal definitions for several constraint classes and present illustrative examples in the context of the resulting taxonomy. This taxonomy, denoted KARQ, is useful for modeling and problem formulation, as well as optimization software development and deployment. It can also be used as the basis for a dialog with practitioners in moving problems to increasingly solvable branches of optimization.

Keywords Taxonomy of constraints · Black-box optimization · Simulation-based optimization.

1 Introduction

This paper focuses on specifications of the feasible set $\Omega \subset \mathbb{R}^n$ for the general optimization problem

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

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ denotes an extended-value objective function. Traditional classifications of constraints in optimization have focused on distinguishing between inequality-, equality-, and set-based constraints, further subdividing these constraints into categories such as linear, conic, convex, and discrete.

S. Le Digabel
GERAD and Départment de Mathématique et de Génie Industriel, Polytechnique Montréal, Montréal, QC H3C 3A7, Canada. https://www.gerad.ca/Sebastien.Le.Digabel.

S. M. Wild
Applied Mathematics & Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA. Industrial Engineering & Management Sciences Department, Northwestern University, Evanston, IL 60208, USA. https://wildsm.github.io.

There is an added layer of complexity in specifying constraints in the fields of *black-box optimization* (BBO) [10] and *simulation-based optimization* (SBO). In (BBO) SBO, the objective function f and/or some constraints defining an instance of Ω depend on the outputs of one or more (black-box) simulations.

The particular way in which a specified constraint set depends on simulation output, along with the nature of the simulation itself, is the focus of this paper. We address issues associated with this dependence, propose a taxonomy of constraints, and unite language in the hope that the taxonomy will provide a bridge between algorithmic research and practitioners. Our taxonomy addresses a specific instance (or "specification") of Ω. This instance, rather than the mathematical problem (1), will be passed to an optimization solver (which may do some preprocessing of its own and then tackle a different instance).

Identifying the specific simulation dependence in a problem is crucial because, in typical scenarios, evaluating the simulation(s) becomes a bottleneck for optimization algorithms. The time needed to evaluate algebraic terms linked with other constraints or the objective can be insignificant compared to the time required for evaluating the simulation components. Furthermore, simulations may sometimes fail to return a value, even for points inside Ω.

To illustrate the distinction between a problem and an instance, we consider the two-dimensional linear problem

$$\min_{\mathbf{x}\in\mathbb{R}^2} \{x_1 + x_2 : x_1 \geq 0,\ x_2 \geq 0\}\,. \tag{2}$$

In fact, many instances of the feasible set Ω share a solution set with (2). For example, a different specification can result in the same feasible set, either by chance,

$$\Omega_1 = \{\mathbf{x} \in \mathbb{R}^2 : x_1 \geq 0,\ x_1 x_2 \geq 0\},$$

or as a result of some redundancy,

$$\Omega_2 = \{\mathbf{x} \in \mathbb{R}^2 : x_1 \geq 0,\ x_2 \geq 0,\ 2x_1 + x_2 \geq 0\}.$$

Alternatively, the feasible sets may differ from one instance to another, but the minimizers of f over these sets remain the same. This is illustrated with

$$\Omega_3 = \{\mathbf{x} \in \mathbb{R}^2 : x_1 \geq 0,\ x_1 + 2x_2 \geq 0,\ x_1 + x_2 \leq 1\}.$$

In situations similar to these examples, one would generally anticipate that modern solvers, modeling languages, or even classical techniques like Fourier-Motzkin elimination would undertake preprocessing. This preprocessing aims to rectify redundancies, inefficiencies, and similar issues before deploying the solver's most intricate mechanisms. When the problem involves some black-box or simulation component, however, the situation, and hence such preprocessing, can be considerably more difficult.

The classification proposed in Section 2 is not absolute: it depends on the entire set of constraints specified in the instance and on the information provided by the problem/simulation designer. For example, a simple bound constraint on a decision variable may be included as an output of a black-box simulation rather than expressed algebraically, leading to two different classes in the taxonomy. Other examples of different constraints changing class are described in Section 3, along with examples from the literature of each constraint type.

Formally, we assume that a finite-dimensional instance Ω is defined by a finite collection of equalities, inequalities, and sets:

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : c_i(\mathbf{x}) = 0, \forall i \in \mathcal{I}; \quad c_j(\mathbf{x}) \leq 0, \forall j \in \mathcal{J}; \quad c_k(\mathbf{x}) \in \mathcal{A}, \forall k \in \mathcal{K}\}, \tag{3}$$

where $\mathcal{I}, \mathcal{J}, \mathcal{K}$ are finite and possibly empty sets, $\mathcal{A} \subseteq \mathbb{R}^n$, and $c_i, c_j, c_k : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ for all $i \in \mathcal{I}$, $j \in \mathcal{J}$, and $k \in \mathcal{K}$. Semi-infinite problems can be treated by such a taxonomy but are not specifically addressed in this paper. Similarly, multi-objective optimization problems are easily encapsulated in our taxonomy but are not discussed specifically. Note that the form of Ω in (3) is general enough to include cases when a variable changes the total number of decision variables (such as when determining the number of groundwater wells to build as well as their locations; see, e.g., [47]).

In fields such as derivative-free optimization (DFO), simulation optimization, and PDE-constrained optimization, a disconnect often exists between what algorithm designers assume about a simulation and what problem/simulation designers provide. In these communities, many different terms coexist for the same concepts, and unification is needed. Section 4 puts the taxonomy in perspective with the existing literature; placing the literature review toward the end of the paper is deliberate and eases the presentation. The proposed taxonomy of constraints consolidates many previous terms such as *soft, virtual, hard, hidden, difficult, easy, open, closed*, and *implicit*. Its purpose is to introduce a common language in order to facilitate dialog between algorithm developers, optimization theoreticians, software users, and application scientists specifying problems.

2 Classes of constraints

This section introduces the KARQ taxonomy, which we present graphically through the tree in Figure 1. An alternative and equivalent representation of the taxonomy using the same notations is provided by the Venn diagram in Figure 2.

Each leaf of the tree in Figure 1 is identified with a sequence of four letters, where each entry can take one of two possible values. The acronym of a leaf reads from the bottom to the root of the tree. As we discuss later, not all 16 possible combinations of these letters are present in the taxonomy, because hidden constraints take a special form. The nine possible constraint classes in the taxonomy are summarized in Table 1.

The letters used to define the acronyms of the taxonomy correspond to four types of left branches in the tree: Q is for Quantifiable, R is for Relaxable,

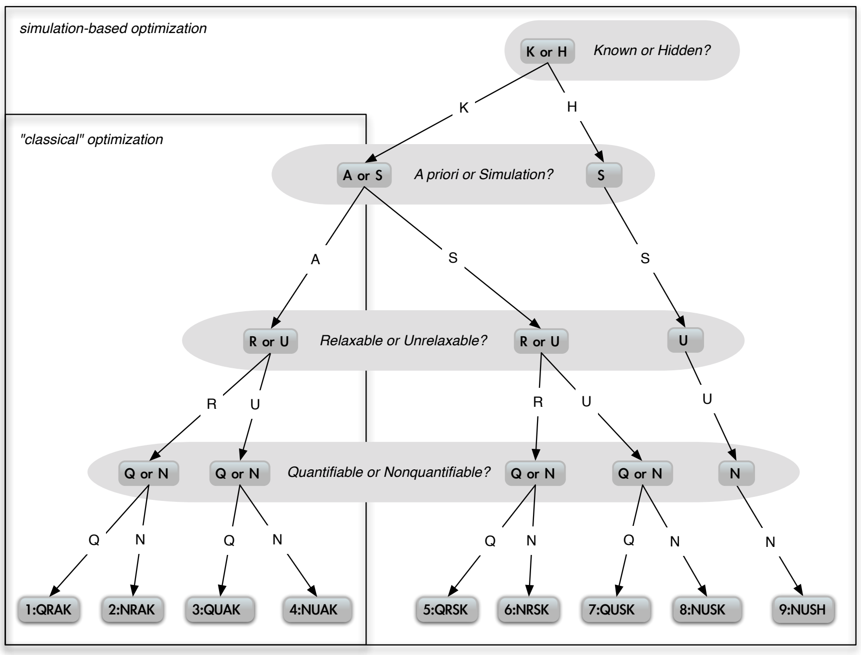

Fig. 1 Tree-based view of the KARQ taxonomy of constraints. Each leaf corresponds to a class of constraints. The leaves read from the bottom of the tree to the top of the tree.

A is for A priori, and K is for Known. The corresponding right branches are identified as N for Nonquantifiable, U for Unrelaxable, S for Simulation, and H for Hidden. We define each of these terms after an initial discussion of the structure of the taxonomy.

The two top levels of the tree are specific to SBO while the lower two are more general. In addition, most of constraints found in traditional nonlinear optimization (NLO) exist in the leftmost leaf. In fact, general difficulty grows from left to right, so that there is a preference for practitioners to specify constraints such that the constraints appear in the leftmost part of the tree possible. Further subdivisions (e.g., convexity, nonlinearity) are also important but more focused on the NLO case and hence not discussed here. Importantly, the taxonomy does not aim to capture whether the simulation output is a smooth function of the decision variables, nor does it imply the availability or non-availability of derivatives of the simulation output.

Each constraint in an SBO problem instance is assigned to one leaf of the tree. However, a constraint type from a classification scheme different from KARQ (e.g., bound constraint, nonlinear equality constraint) can correspond to several KARQ leaves at once. In this case, we use the generic wildcard notation "*." For example, simulation-based constraints can be written as **S*.

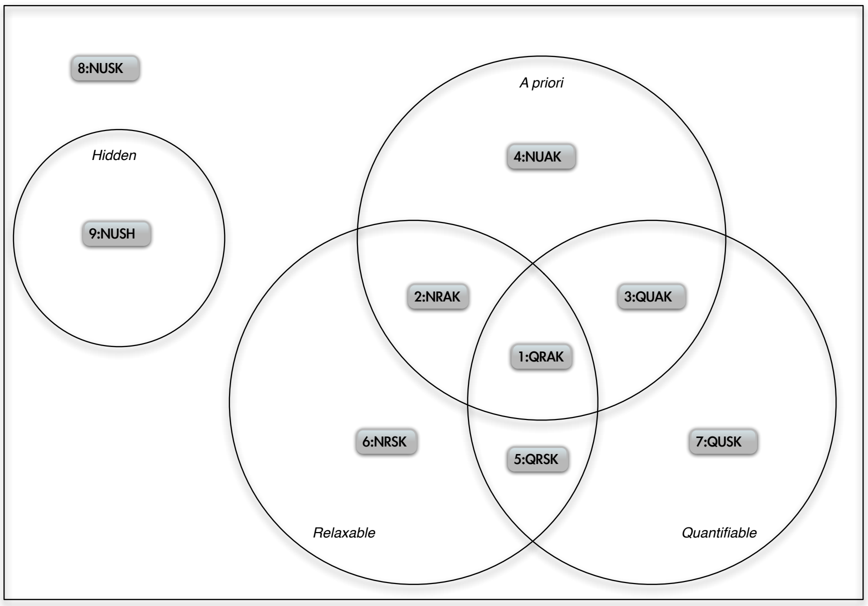

Fig. 2 Venn diagram of the taxonomy of constraints. Each region corresponds to a leaf in the tree of Figure 1.

Table 1 The taxonomy as a table where each row corresponds to a leaf in the tree of Figure 1 and to an intersection of regions in Figure 2.

Leaf Number in Figure 1	Name in the Taxonomy
1	QRAK
2	NRAK
3	QUAK
4	NUAK
5	QRSK
6	NRSK
7	QUSK
8	NUSK
9	NUSH (hidden)

These issues will appear natural as we proceed with examples and formal definitions of each level of the tree, starting from the bottom and moving to the top. In the next section, we provide examples from the literature for each leaf of the tree.

2.1 Quantifiable (Q) versus nonquantifiable (N)

In the case of a nonquantifiable constraint, one has only a binary indicator indicating whether the constraint has been satisfied or violated. Consequently, an alternative term for such constraint is a *binary* or *0-1* constraint, but this does not have a natural complementary term. Similarly, we avoid the terms *measurable/nonmeasurable* to prevent confusion with the term *measurable* in analysis.

Definition 1 A quantifiable constraint (Q) is a constraint for which the degree of feasibility and/or violation can be quantified. A nonquantifiable constraint (N) is a constraint for which the degrees of satisfaction or violation are both inaccessible.

The definition of a quantifiable constraint does not guarantee that measures of both feasibility and violation are available. In particular, both of the following examples fall under the category of quantifiable constraints.

Quantifiable feasibility: *The time required for the underlying simulation code to complete should be less than 10 seconds.*
Here, we have access to the time that it took for the code to complete (and hence we know how close we are to the 10-second limit), but the execution is interrupted if it fails to complete within 10 seconds (and hence we will never know the degree to which the constraint was violated).

Quantifiable violation: *A time-stepping simulation should run to completion (time T).*
If the simulation stops at time $\hat{t} < T$, then $T - \hat{t}$ quantifies the proximity to satisfying the constraint.

A constraint for which both the degrees of feasibility and of violation are available can be referred to as *fully quantifiable.*

From the perspective of a method or solver, the distinction between Q and N clearly is important. For example, if one wants to build a model of the constraint, Q might imply *interpolation*, while N might imply *classification.*

2.2 Relaxable (R) versus unrelaxable (U)

The next concept addressed by the taxonomy is that of relaxability.

Definition 2 A relaxable constraint (R) is a constraint that does not need to be satisfied in order to obtain *meaningful* outputs from the simulation for the computation of the objective and the constraints. An unrelaxable constraint (U) is one that must be satisfied to obtain meaningful outputs.

In this definition, *meaningful* simulation output(s) means that the values can be trusted as valid by an optimization algorithm and rightly interpreted when observed in a solution.

Typically, relaxable constraints are not part of a physical model but instead represent some customer specifications or some desired restrictions on the outputs of the simulation, such as a budget or a weight limit.

Within an optimization method, the consequence of this R versus U property is that all the iterates must satisfy unrelaxable constraints, whereas relaxable constraints are required to be satisfied only at the proposed solution. To put it differently, infeasible points may be considered as intermediary (approximate) solutions.

Alternative terms include *soft* versus *hard*, *open* versus *closed*, and *violable* versus *unviolable*; but these terms are often overloaded, as we note in Section 4.

2.3 A priori (A) versus simulation-based (S)

A simulation constraint is specific to BBO/SBO. The essence of a simulation constraint is such that it necessitates the initiation of a potentially resource-intensive computer or physical simulation to assess the constraint. We note, however, that this constraint evaluation may not ultimately prove to be costly. For instance, the simulation could encompass a constraint that is relatively inexpensive to evaluate and can serve as a flag to circumvent additional computations; even in this scenario, our taxonomy classifies such a constraint as an S constraint (more precisely, a *USK constraint).

Definition 3 An a priori constraint (A) is a constraint for which feasibility can be confirmed without running a simulation. A simulation-based constraint (S) (or simulation constraint) requires running a simulation to determine feasibility.

Basic examples of a priori constraints include one-sided bounds and linear equalities. Nevertheless, a priori constraints can encompass notably broader and specialized formulations, including semidefinite programming constraints, constraint programming constraints (e.g., *all different*, *ordered*), or constraints relative to the nature of variables, such as real, integer, binary, or categorical.

An alternative to "simulation" is *a posteriori* [8]; alternative terms for "a priori" include *algebraic* or *algebraically available*, *analytic*, *closed-form*, *expressible*, and *input-constraint.* An *algebraic function* is typically defined as a function that satisfies an equation that can be expressed as a polynomial of finite degree with rational coefficients. However, it is important to note that this definition excludes transcendental functions (e.g., e^x). Some modeling languages, such as GAMS [15], already use this terminology (GAMS is "generalized algebraic" to include available transcendentals). Formally, an *analytic function* is usually one that locally has a convergent power series; this rules out simple nonsmooth functions. The concept behind the term *input-constraint* lies in the fact that A constraints can be perceived as directly linked to the input variables $\mathbf{x}$, whereas S constraints are formulated as a function of the simulation outputs.

One can easily appreciate that a solver should ideally evaluate *UA* (unrelaxable, a priori) constraints first, avoiding a simulation execution if the candidate is infeasible–especially when it is expensive to obtain the simulation output. For *RA* (relaxable, a priori) constraints, it is not as clear whether an algorithm would benefit from a similar ordering of constraint evaluations. For example, in the case of noninteger input values, should these be transmitted to a simulator, which might subsequently round them to the nearest integer during simulation? The answer hinges on the specific context.

2.4 Known (K) versus hidden (H)

The final distinction within the taxonomy applies specifically to BBO/SBO.

Definition 4 A known constraint (K) is a constraint that is explicitly given in the problem formulation. A hidden constraint (H) is not explicitly known to the solver.

Most constraints encountered when solving SBO problems, particularly when an optimizer is involved in the early stages of modeling and problem formulation, are *known* to the optimizer. A hidden constraint typically (but not necessarily) arises when the simulation fails to converge or crashes, for various reasons such as incorrect usage conditions or simulator bugs.

For such constraints, we can detect only violations, typically when some error flag or exception is raised. However, a violation may go unnoticed. Alternative terms include *Unknown*, *Unspecified*, and *Forgotten*.

For example, consider the problem $\min\{f(\log x) : x \in \mathbb{R}\}$ with f being a simulation-based function from $\mathbb{R}$ to $\mathbb{R}$. If the constraint $x > 0$ is stated in the problem specification, then it is an a priori constraint. Otherwise, it is hidden and can only be observed for negative or null values of x. This constraint might have been explicitly stated within the simulator to prevent crashes and trigger specific flags; however, if this information is not communicated to the solver, the constraint remains categorized as H.

As depicted in Figure 1, the H branch of the tree uniquely leads to a terminal leaf. By definition, a hidden constraint cannot be a priori and quantifiable (as we lack the necessary information for quantification). Furthermore, it cannot be relaxable because the violation/satisfaction cannot be detected if the outputs are always meaningful.

Note that the demarcation between a hidden (NUSH) constraint and a NUSK constraint is subtle. In the NUSK case, however, the constraint is explicitly given, and its satisfaction can be checked. These subtle differences are emphasized in the presence of several different hidden constraints: When the simulation crashes, one has no way of knowing exactly what went wrong, a situation that would have been different if these constraints had been expressed with flags by the modeler.

3 Short case studies

The previous examples were related to the four levels of decision in the taxonomy. We now highlight that each of the nine leaves of the tree in Figure 1 is nonempty by providing constraint specifications, including examples from the literature, that belong to each leaf.

We note that many SBO problem specifications in the literature are composed of constraints of different types. The community groundwater problem [27] has QRSK constraints as well as bound and linear constraints (**AK), while the LOCKWOOD problem [47] has a linear objective and simulation constraints (**S*); different simulation-based instances of the LOCKWOOD constraints are considered in [36] alongside solution methodologies for the resulting formulations. The STYRENE problem from [6] has 11 simulation constraints corresponding to Leaves 5 and 8 of the tree of Figure 1: 7 quantifiable and relaxable constraints (QRSK), and 4 unrelaxable binary constraints (NUSK).

3.1 QRAK: Quantifiable Relaxable A priori Known

This constraint class captures the most common types of constraints encountered in classical nonlinear optimization.

Ex.- $\mathsf{location}_1 \notin \mathsf{forest}$

The GMON problem described in [4] involves placing gamma monitoring devices in a region represented by the image of a map. There are several locations on the map (corresponding to forests, lakes, etc.) that are considered unallowable for placement. However, the simulation, which analyzes the effects of placing devices at particular locations, is independent of the surface topology and thus can still evaluate such unallowable placements. The constraints associated with these unallowable locations are also quantifiable, since one can compute distances to the nearest (in)feasible location on the map.

Ex.- $0.15 \le x_1 \le 0.17$

In [39], a nonlinear-least-squares problem is solved for parameter values that calibrate a nuclear energy density functional to experimental data. Each residual involves running a simulation associated with self-consistency of the underlying system. Bounds are set in [39] on half of the parameters (those associated with nuclear matter properties) because the unconstrained solution yields values that are not transferable to other nuclear structure calculations. Because the simulation still yields meaningful output when these bounds are violated (as evidenced by the unconstrained problem being solved), these a priori and quantifiable constraints are also relaxable.

Ex.- $x_1 \in \{0, 1\}$

A violation measure is quantified by, for example, $\min\big\{|x_i|,\ |1 - x_i|\big\}$ and thus this constraint is QRAK provided the binary variable x_1 is relaxable.

3.2 NRAK: Nonquantifiable Relaxable A priori Known

A good example of an NRAK constraint is a categorical (e.g., nonordinal) variable constrained to a subset of its possible values. Such cases arise frequently in multifidelity optimization.

Ex.- simulator = Cart3D

For the purpose of designing a supersonic airfoil, [44] considers different available analysis simulators – such as a linearized panel method ("panel") and an Euler CFD solver ("Cart3D") – with varying fidelity levels. The final design must be evaluated by the more costly and accurate Cart3D simulator, but an optimization algorithm can use the less expensive, inaccurate panel simulator at intermediate points. The constraint "simulator = Cart3D" is relaxable because another option for the simulator exists.

Ex.- $x_2 = \mathsf{O2}$ if $x_1 = \mathsf{gcc}$

Consider a simulator that drives a C++ compilation, based on two categorical variables, each specified to take on two allowable values, $x_1 \in \{\mathsf{gcc}, \mathsf{icc}\}$ and $x_2 \in \{\mathsf{O2}, \mathsf{O3}\}$. We want the final solution to have $x_2 = \mathsf{O2}$ if $x_1 = \mathsf{gcc}$, and this constraint is of type NRAK.

We note that the two set constraints, $x_1 \in \{\mathsf{gcc}, \mathsf{icc}\}$ and $x_2 \in \{\mathsf{O2}, \mathsf{O3}\}$, may be NUAK; see below.

3.3 QUAK: Quantifiable Unrelaxable A priori Known

The occurrence of QUAK constraints in SBO is commonly related to algebraic constraints that play a (possibly secondary) role of ensuring that the input to a simulation satisfies the expectations of the simulator.

Ex.- $r_i \geq 0,\ i = 1, \ldots 6$

For the groundwater bioremediation problem considered in [36], pumping rates r need to be determined for six extraction wells. The simulator's design only permits extraction (not injection) at these specific sites, making these constraints unrelaxable. Furthermore, the quantity $\max\{0, r_i\}$ offers a quantifiable measure of feasibility.

3.4 NUAK: Nonquantifiable Unrelaxable A priori Known

The allowable domain of a categorical variable is a typical example of a NUAK constraint.

Ex.- compiler $\in \{\mathsf{gcc}, \mathsf{icc}\}$

Categorical decision variables are often necessary when optimizing code performance. For instance, the optimization of linear algebra kernels in [14] involves categorical variables like loop order (e.g., permuting groups of nested for loops) and compiler type.

3.5 QRSK: Quantifiable Relaxable Simulation Known

Any quantifiable and relaxable constraint that depends on a simulation output is a QRSK constraint.

Ex.- Minimal purity level of produced styrene of 0.99.
In the chemical engineering application for the production of styrene considered in [6], called STYRENE, one of the simulation outputs is the purity of the styrene produced. The problem requires a minimum purity level, but intermediate designs might not meet this requirement. In such cases, a quantifiable measure of violation represents how much the obtained purity level falls below 0.99.

Ex.- Percentage of test problems solved must be at least 90%.
In [7], the algorithmic parameters of an optimization solver are tuned in order to minimize the CPU time needed to solve a set of test problems. Here, the "simulator" is a call of the solver being tuned on a particular test problem. The authors consider constraints on correctness of the output, in particular that at least 90% of the runs satisfy a prescribed convergence criterion.

Ex.- A budget based on economical criteria, $S(\mathbf{x}) \leq b$.

3.6 NRSK: Nonquantifiable Relaxable Simulation Known

A relaxable constraint that relies on binary or nonordinal output from a simulation is a NRSK constraint.

Ex.- no compiler warnings
A common objective in code performance optimization is to minimize the run time of a generated code. Changes to the code may result in code that still compiles and runs, but for which undesirable warnings are produced during compilation. In [31], the run times of decision variable values that resulted in problems (errors or warnings) during compilation were discarded from the analysis. The constraint that there are no warnings requires one to generate, compile, and check the compilation output of the associated code is thus "simulation-based;" The study in [31] only considers the binary measures of this constraint, which is relaxable since run times can still be obtained (assuming that no compile errors were seen).

Ex.- toxicity value exceeded
A simulator displays a flag indicating whether a toxicity level has been reached during the simulation. However, neither the timing of this event nor the extent of the toxicity is known.

3.7 QUSK: Quantifiable Unrelaxable Simulation Known

Constraints that are both quantifiable and unrelaxable arise, for example, in multidisciplinary optimization (MDO), where one has a compatibility constraint between the simulations of two or more disciplines. Violating such a constraint can lead to unmeaningful output with respect to one of the disciplines.

Ex.- $y_1 \geq 0$
Compatibility constraints in MDO often arise when the output(s) of one discipline become inputs for another. Bounds on these quantities are typical QUSK constraints, such as the problem presented in [46, Section 5.1]. In this example, the first discipline output (y_1) appears with a squared root in the equation of the other discipline. The bound constraint $y_1 \geq 3.16$ is specified in [46] and is a QRSK constraint because it can be relaxed (albeit not arbitrarily so) and still result in meaningful output from the second discipline. The constraint $y_1 \geq 0$, however, is unrelaxable and thus a QUSK constraint (despite seeming to be redundant in the presence of the relaxable constraint $y_1 \geq 3.16$).

Ex.- $c_S(\mathbf{x}) \geq 0$
One of the outputs $c_S(\mathbf{x})$ of a simulation is a concentration level; if it is below zero, the simulation stops and displays NaN for all the outputs except c_S. Consequently, this constraint is both unrelaxable and quantifiable since there exists a measure indicating the proximity to violation.

3.8 NUSK: Nonquantifiable Unrelaxable Simulation Known

Constraints based on binary- and nonordinal-valued simulation outputs that must be satisfied are typical NUSK constraints.

Ex.- `SEP-STY` is structurally acceptable for subsequent simulations.
In the STYRENE problem of [6], four of the eleven simulation-based constraints correspond to binary flags related to the success of the convergence of internal numerical methods. In the case of the `SEP-STY` flag denoting failure, some of the other flags may still denote success; hence, one has richer information about the source of failure than one does for a hidden constraint.

Ex.- `exitflag` $= 0$
When a simulation generates an exit flag indicating success ("0") or multiple error code values accompanied by relevant documentation, it becomes possible to discern why the simulation did not succeed.

Each of these examples is not a hidden constraint since the reason for the violation can be identified. In contrast, a single binary flag indicating that the simulation failed qualifies as a hidden constraint. Similarly, an error message that cannot be interpreted is equivalent to such a flag and hence should be interpreted as hidden.

3.9 NUSH: Hidden

As defined in Section 2.4, hidden constraints provide no information about the cause for failure of a simulation.

Ex.- When sampled uniformly from a specified hyperrectangle, the STYRENE black-box of [6] terminates prematurely 60% of the time, resulting in a failure. These failures are not related to the four binary flags described in the NUSK examples and thus one has no means of targeting the cause of the failure.

Ex.- The simulation failed to complete and nothing is displayed, or a simple flag is raised or an undocumented error number indicated.

Ex.- The SOLAR simulator [42], available at `https://github.com/bbopt/solar`, is a realistic black-box in which, in some situations, some of the outputs are not computed. These situations reveal the presence of hidden constraints.

4 Literature review

In this section, we review the existing literature and gather terminology from the BBO, DFO, and SBO communities. Our goal is to unify and relate our taxonomy to past terms and formulations and to highlight inconsistencies among previous conventions. This context also underpins the naming conventions used in KARQ and the more formal definitions on which the taxonomy is built. We also survey early uses of various terms and illustrate the use of the taxonomy in the context of SBO software packages.

Before proceeding, we note that the proposed classification is not related to the field of constraint programming [51], where constraints are expressed as logical prepositions treated by specialized algorithms within a specific context.

4.1 Hidden constraints

The term *hidden constraint* corresponds to the NUSH leaf in the tree of Figure 1. This term is increasingly prevalent in DFO. In the modern literature, it is typically attributed to Choi and Kelley [19], who say that a hidden constraint is "the requirement that the objective be defined." This definition is used in Kelley's implicit filtering software [37] and has been used to solve several examples (see, e.g., [16, 18]) whereby a hidden constraint is said to be violated whenever flow conditions are found that prevent a simulation solution from existing.

In fact, the term had been previously used in the context of optimization. The earliest published instance that we are aware of is from 1967 [12] and involved optimizing the design of a condenser. In this case, after a design was numerically evaluated, one needed to verify that the Reynolds number obtained was large enough to justify the use of the equations in the calculations.

In our taxonomy, such a constraint is not labeled as hidden because one knows why the design "failed."

The term is also used by the authors of the SNOBFIT package [35] to capture when "a requested function value may turn out not to be obtainable." To handle such constraints, SNOBFIT assigns an artificial value, based on the values of nearby points, to the points where such a constraint was violated. Similarly, the authors of [22] define hidden constraints as those that

> "are not part of the problem specification/formulation, and their manifestation comes in the form of some indication that the objective function could not be evaluated."

The authors of [22] state that hidden constraints have historically been treated only by heuristic approaches or by the *extreme-barrier* approach, which uses extended-value functions in an attempt to establish feasibility.

A selection of recent works involving hidden constraints includes [2,13,48,52].

4.2 Unrelaxable and relaxable constraints

The terms *unrelaxable* and *relaxable* are widely used in the literature. For example, the book [22] states that unrelaxable constraints "have to be satisfied at all iterations" of an algorithm while "relaxable constraints need only be satisfied approximately or asymptotically." The related notions of *hard* and *soft* constraints appear with varying meanings. Here, we follow the convention of [34]:

> "To resolve this, the requirements are usually broken up into "hard" constraints for which any violation is prohibited, and "soft" constraints for which violations are allowed. Typically hard constraints are included in the formulation as explicit constraints, whereas soft constraints are incorporated into the objective function via some penalty that is imposed for their violation."

That is, we view soft constraints as being handled by either additional objectives or additional objective terms. A nice pre-1969 history of ways to move constraints into the objective can be found in [25]. Another example comes from SNOBFIT [35], where soft constraints are "constraints which need not be satisfied accurately." Other uses of "hard/soft constraints" can be found, for example, in [33]. There, the authors refer to soft constraints as those "that need not be satisfied at every iteration," a definition that is directly related to our term unrelaxable. A similar notion is used in [32]: "Relaxable constraints need only be satisfied approximately or asymptotically." But our definition requires that a solution satisfy relaxable constraints, and hence the degree of "approximate" satisfaction must be specified in the problem instance. In [45], constraints are divided into relaxable and unrelaxable constraints, where unrelaxable constraints

> "cannot be violated by any considered solution because they guarantee either the successful evaluation of the black-box function . . . or the physical/structural feasibility of the solution"

and relaxable constraints "may instead be violated as the objective function evaluation is still successful."

The authors of [24] employ a specific instance of our definition, and use both "nonrelaxable" and "unrelaxable" to describe constraints whose satisfaction is necessary for the successful evaluation of an objective function.

4.3 Other related work

Previous classifications have also been proposed, as for example the mixed-integer programming classification in [49] for linear inequalities, linear equations, continuous parameters, and discrete parameters.

The closest related work toward a more complete characterization of constraints is that of Alexandrov and Lewis [5], who examined different formulations for general problems arising in MDO. These authors considered constraint sets partitioned along three axes: open (closed) disciplinary analysis, open (closed) design constraints, and open (closed) interdisciplinary consistency constraints. They showed that out of the eight possible combinations, only four were possible in practice. They referred to *closed* constraints as those

> "assumed to be satisfied at every iteration of the optimization. If the formulation does not necessarily assume that a set of constraints is satisfied, we will say that that formulation is open with respect to the set of constraints."

This convention has subsequently been used by others in the MDO community (see, e.g., [53]).

The notion of unknown constraints appears in [30] but it differs from its use in our taxonomy; rather, it corresponds to constraints given by a black-box. Note that the same authors, along with others, discussed hidden constraints in [41].

Additional terms for describing general constraints are found in the literature. For example, *chance constraints* [17] are constraints whose satisfaction requirement depends on a probability. *Side constraint* is a generic term sometimes used to qualify constraints that are not lower or upper bounds or to distinguish new constraints added to a preexisting model; see [3] for an example. Other terms include the notions of *vanishing* constraints [1], *complementarity* constraints, or *variational inequalities* [43].

Conn et al. describe *easy* constraints and *difficult* constraints as follows [20]:

> "Easy constraints are the constraints whose values and derivatives can be easily computed,"
> and
> "Difficult constraints are constraints whose derivatives are not available

and whose values are at least as expensive to compute as that of the objective function."

The latter definition is similar to what [22] calls *derivative-free* constraints, that is, those for which derivatives are not available and which are typically given by a black-box. Such characterizations vary from our proposed taxonomy, which does not seek to guarantee a specific order based on the computational expense of constraint evaluation and/or feasibility determination.

Similarly, the authors of [20] refer to *virtual constraints* as "constraints that cannot explicitly be measured." Only the satisfiability of such constraints can be checked, and this is assumed to be a computationally expensive procedure. In our taxonomy such constraints are N**K.

Numerous modeling languages and collections of test problems such as GAMS [15], AMPL [26], CUTEst [29], or ZIMPL [38], use the following classic ways of categorizing constraints: fixed variables; bounds on the variables; adjacency matrix of a (linear) network; linear, quadratic, equilibrium, and conic constraints; logical constraints found in constraint programming; and equalities or inequalities. Usually, the remaining constraints are qualified as "general", a term frequently used in classical nonlinear optimization. All these constraints fit as **AK constraints in the "classical optimization" portion of the tree of Figure 1.

4.4 Software for constrained SBO problems

To highlight the potential benefits of utilizing the proposed taxonomy, we briefly outline how some algorithms and software handle various types of constraints, using the terminology of the taxonomy.

In general, most general-purpose software packages consider QR*K constraints, but some tend to use exclusively algebraic forms (e.g., box, linear, quadratic, convex). Furthermore, relaxable constraints often are also assumed to be quantifiable. Several packages allow for a priori constraints, but some assume that these cannot be relaxed, while others assume that they can.

The package SNOBFIT [35] treats *soft* and also NUSH (hidden) constraints. The software SID-PSM [23] handles constraints with derivatives and U (unrelaxable) constraints. The DFO code [21] (which we distinguish from the general class of optimization problems without derivatives) considers NUSH (hidden), NU*K, and Q*AK constraints. On the DFO solver page [21], the authors recommend moving S (simulation, *difficult*) constraints to the objective function, while keeping *easy* constraints (with derivatives) inside the trust-region subproblem; the authors also describe *virtual* constraints as N (non-quantifiable) constraints and recommend using an extreme-barrier approach. The HOPSPACK package [50] explicitly addresses integers; linear equalities and inequalities; and general inequalities and equalities. Depending on the type of constraint, HOPSPACK assumes that the constraint is relaxable (e.g., general equality constraints) or unrelaxable (e.g., integer sets). In NOMAD [11,40], the progressive-barrier technique [9] is used for the QRSK constraints, and special treatment (such as projection) is applied for some Q*AK constraints (i.e., bounds and integers). The extreme barrier is used for all other constraints, including hidden constraints.

In PDE-constrained optimization, solution approaches can be loosely classified into "Nested Analysis and Design" (NAND) and "Simultaneous Analysis and Design" (SAND) approaches [28]. In NAND approaches, the state variables of the PDE constraints are not treated as decision (optimization) variables and hence the solution of the PDE (for the state variables) is a simulation constraint. This situation exists even if the simulation is not just a black-box, but also returns additional information (e.g,. sensitivities, adjoints, tolerances). In the SAND approach, the state variables are included as decision variables and hence the PDE reduces to a set of algebraic equations (and therefore **AK constraints in our taxonomy).

5 Discussion

This work proposes a unification of past conventions and terms into a single taxonomy, denoted KARQ, which targets the constraints encountered in simulation-based optimization. The taxonomy has an intuitive representation as a tree where each leaf describes one of nine types of possible constraints. In addition, examples have been given for each constraint type and their possible treatment in applications and algorithms.

We propose that BBO, DFO, and SBO software and algorithms should adopt this taxonomy for two important reasons. The first is unification, so that researchers in the field use the same terms and practitioners and algorithm developers share the same language. The second reason is that the taxonomy is a tool to better identify constraint types and thereby achieve effective algorithmic treatment of more general types of constrained optimization problems.

Future work is related to extending the existing taxonomy. It is possible to refine the tree in Figure 1 by adding subcases to the leaves, depending on the specific context. These extensions within KARQ could encompass various constraint types, including stochastic, convex, linear, and smooth constraints that have available derivatives. Additionally, options for equality, inequality, or set membership constraints could be explored. For instance, while treating an equality N*SK constraint might be challenging (or even impossible), handling an equality Q*AK constraint could be more feasible. At a different level, we consider the addition of three branches from each Q node: quantifiable feasibility only, quantifiable violation only, and fully quantifiable. There is also a limit to being unrelaxable: So far we say that a constraint is unrelaxable if it is unrelaxable at some point, and we may want to specify such limits when they are known. Finally, some of the terms of the developed taxonomy could be extended to the objective function.

Acknowledgements This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research applied mathematics program under contract nos. DE-AC02-05CH11231 and DE-AC02-06CH11357; by the Air Force Office of Scientific Research Grant FA9550-12-1-0198; and by the Natural Sciences and Engineering Research Council of Canada Discovery Grant 418250. The authors thank the American Institute of Mathematics for the SQuaRE workshops that allowed them to initiate their discussions on the taxonomy; the authors thank the participants: Bobby Gramacy, Genetha Gray, Herbie Lee, and Garth Wells. Thanks also to Charles Audet, John Dennis, Warren Hare, Sven Leyffer, and Paul Patience, for their useful comments and suggestions.

References

1. Achtziger, W., Kanzow, C.: Mathematical programs with vanishing constraints: optimality conditions and constraint qualifications. Mathematical Programming **114**(1), 69–99 (2008). DOI 10.1007/s10107-006-0083-3
2. Adcock, B., Cardenas, J., Dexter, N.: An Adaptive Sampling and Domain Learning Strategy for Multivariate Function Approximation on Unknown Domains. SIAM Journal on Scientific Computing **45**(1), A200–A225 (2023). DOI 10.1137/22M1472693
3. Aggarwal, V., Aneja, Y., Nair, K.: Minimal spanning tree subject to a side constraint. Computers and Operations Research **9**(4), 287–296 (1982). DOI 10.1016/0305-0548(82)90026-0
4. Alarie, S., Audet, C., Garnier, V., Le Digabel, S., Leclaire, L.A.: Snow water equivalent estimation using blackbox optimization. Pacific Journal of Optimization **9**(1), 1–21 (2013). URL http://www.ybook.co.jp/online2/oppjo/vol9/p1.html
5. Alexandrov, N., Lewis, R.: Comparative properties of collaborative optimization and other approaches to MDO. ICASE Report 99–24, Institute for Computer Applications in Science and Engineering (1999). http://techreports.larc.nasa.gov/ltrs/PDF/1999/mtg/NASA-99-asmo-nma.pdf
6. Audet, C., Béchard, V., Le Digabel, S.: Nonsmooth optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search. Journal of Global Optimization **41**(2), 299–318 (2008). DOI 10.1007/s10898-007-9234-1
7. Audet, C., Dang, C.K., Orban, D.: Algorithmic parameter optimization of the DFO method with the OPAL framework. In: Software Automatic Tuning: From Concepts to State-of-the-Art Results, K. Naono and K. Teranishi and J. Cavazos and R. Suda edn., chap. 15, pp. 255–274. Springer (2010). DOI 10.1007/978-1-4419-6935-4_15
8. Audet, C., Dang, C.K., Orban, D.: Efficient use of parallelism in algorithmic parameter optimization applications. Optimization Letters **7**(3), 421–433 (2013). DOI 10.1007/s11590-011-0428-6
9. Audet, C., Dennis, Jr., J.E.: A Progressive Barrier for Derivative-Free Nonlinear Programming. SIAM Journal on Optimization **20**(1), 445–472 (2009). DOI 10.1137/070692662
10. Audet, C., Hare, W.: Derivative-Free and Blackbox Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Cham, Switzerland (2017). DOI 10.1007/978-3-319-68913-5
11. Audet, C., Le Digabel, S., Rochon Montplaisir, V., Tribes, C.: Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm. ACM Transactions on Mathematical Software **48**(3), 35:1–35:22 (2022). DOI 10.1145/3544489
12. Avriel, M., Wilde, D.: Optimal Condenser Design by Geometric Programming. Industrial & Engineering Chemistry Process Design and Development **6**(2), 256–263 (1967). DOI 10.1021/i260022a018
13. Bachoc, F., Helbert, C., Picheny, V.: Gaussian process optimization with failures: classification and convergence proof. Journal of Global Optimization **78**(3), 483–506 (2020). DOI 10.1007/s10898-020-00920-0
14. Balaprakash, P., Wild, S., Hovland, P.: Can search algorithms save large-scale automatic performance tuning? Procedia Computer Science (ICCS 2011) **4**, 2136–2145 (2011). DOI 10.1016/j.procs.2011.04.234

15. Brooke, A., Kendrick, D., Meeraus, A.: GAMS: A Users' Guide. The Scientific Press, Danvers, Massachusetts (1988)
16. Carter, R., Gablonsky, J., Patrick, A., Kelley, C., Eslinger, O.: Algorithms for Noisy Problems in Gas Transmission Pipeline Optimization. Optimization and Engineering **2**, 139–157 (2001). DOI 10.1023/A:1013123110266
17. Charnes, A., Cooper, W., Symonds, G.: Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. Management Science **4**(3), 235–263 (1958). DOI 10.1287/mnsc.4.3.235. URL http://www.jstor.org/stable/2627328
18. Choi, T., Eslinger, O., Kelley, C., David, J., Etheridge, M.: Optimization of Automotive Valve Train Components with Implicit Filtering. Optimization and Engineering **1**, 9–27 (2000). DOI 10.1023/A:1010071821464
19. Choi, T., Kelley, C.: Superlinear Convergence and Implicit Filtering. SIAM Journal on Optimization **10**(4), 1149–1162 (2000). DOI 10.1137/S1052623499354096
20. Conn, A., Scheinberg, K., Toint, P.L.: A Derivative Free Optimization Algorithm in Practice. In: Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization (1998). http://perso.fundp.ac.be/~phtoint/pubs/TR98-11.ps
21. Conn, A., Scheinberg, K., Toint, P.L.: DFO (Derivative Free Optimization Software). https://projects.coin-or.org/Dfo (2001)
22. Conn, A., Scheinberg, K., Vicente, L.: Introduction to Derivative-Free Optimization. MOS-SIAM Series on Optimization. SIAM, Philadelphia (2009). DOI 10.1137/1.9780898718768
23. Custódio, A., Vicente, L.: SID-PSM: A pattern search method guided by simplex derivatives for use in derivative-free optimization. http://www.mat.uc.pt/sid-psm (2005)
24. Diouane, Y., Gratton, S., Vicente, L.: Globally convergent evolution strategies for constrained optimization. Computational Optimization and Applications **62**(2), 323–346 (2015). DOI 10.1007/s10589-015-9747-3
25. Fiacco, A., McCormick, G.: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. Classics in Applied Mathematics 4. SIAM (1990). DOI 10.1137/1.9781611971316
26. Fourer, R., Gay, D., Kernighan, B.: AMPL: A Modeling Language for Mathematical Programming, second edn. Thomson/Brooks/Cole, Pacific Grove, California (2003)
27. Fowler, K., Reese, J., Kees, C., Dennis, Jr., J., Kelley, C., Miller, C., Audet, C., Booker, A., Couture, G., Darwin, R., Farthing, M., Finkel, D., Gablonsky, J., Gray, G., Kolda, T.: Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. Advances in Water Resources **31**(5), 743–757 (2008). DOI 10.1016/j.advwatres.2008.01.010
28. Ghattas, O., Biegler, L., Heinkenschloss, M., van Bloemen Wanders, B.: Large-Scale PDE-Constrained Optimization: An Introduction. In: L. Biegler, O. Ghattas, M. Heinkenschloss, B. van Bloemen Wanders (eds.) Large-Scale PDE-Constrained Optimization, pp. 3–13. Springer, New York (2003). URL http://www.cs.sandia.gov/~bartv/papers/overview.ps
29. Gould, N., Orban, D., Toint, P.: CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Computational Optimization and Applications **60**(3), 545–557 (2015). DOI 10.1007/s10589-014-9687-3. Code available at http://ccpforge.cse.rl.ac.uk/gf/project/cutest/wiki
30. Gramacy, R., Lee, H.: Optimization under unknown constraints. In: J. Bernardo, S. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, M. West (eds.) Proceedings of the Ninth Valencia International Meetings on Bayesian Statistics, pp. 229–256. Oxford University Press (2011). DOI 10.1093/acprof:oso/9780199694587.003.0008
31. Gramacy, R., Taddy, M., Wild, S.: Variable selection and sensitivity analysis via dynamic trees with an application to computer code performance tuning. Annals of Applied Statistics **7**(1), 51–80 (2013). DOI 10.1214/12-AOAS590
32. Gratton, S., Vicente, L.: A merit function approach for direct search. SIAM Journal on Optimization **24**(4), 1980–1998 (2014). DOI 10.1137/130917661
33. Griffin, J., Kolda, T.: Nonlinearly Constrained Optimization Using Heuristic Penalty Methods and Asynchronous Parallel Generating Set Search. Applied Mathematics Research eXpress **2010**(1), 36–62 (2010). DOI 10.1093/amrx/abq003

34. Griva, I., Nash, S., Sofer, A.: Linear and Nonlinear Optimization. SIAM (2009)
35. Huyer, W., Neumaier, A.: SNOBFIT – Stable Noisy Optimization by Branch and Fit. ACM Transactions on Mathematical Software **35**(2), 9:1–9:25 (2008). DOI 10.1145/1377612.1377613
36. Kannan, A., Wild, S.: Benefits of deeper analysis in simulation-based groundwater optimization problems. In: Proceedings of the XIX International Conference on Computational Methods in Water Resources (CMWR 2012) (2012). URL http://www.mcs.anl.gov/~wild/papers/2012/AKSW12.pdf
37. Kelley, C.: Implicit Filtering. SIAM, Philadephia, PA (2011). DOI 10.1137/1.9781611971903
38. Koch, T.: Rapid Mathematical Prototyping. Ph.D. thesis, Technische Universität Berlin (2004)
39. Kortelainen, M., Lesinski, T., Moré, J., Nazarewicz, W., Sarich, J., Schunck, N., Stoitsov, M.V., Wild, S.M.: Nuclear Energy Density Optimization. Physical Review C **82**(2), 024,313 (2010). DOI 10.1103/PhysRevC.82.024313
40. Le Digabel, S.: Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm. ACM Transactions on Mathematical Software **37**(4), 44:1–44:15 (2011). DOI 10.1145/1916461.1916468
41. Lee, H., Gramacy, R., Linkletter, C., Gray, G.: Optimization subject to hidden constraints via statistical emulation. Pacific Journal of Optimization **7**(3), 467–478 (2011). URL http://www.ybook.co.jp/online2/oppjo/vol7/p467.html
42. Lemyre Garneau, M.: Modelling of a solar thermal power plant for benchmarking blackbox optimization solvers. Master's thesis, Polytechnique Montréal (2015). URL https://publications.polymtl.ca/1996/
43. Luo, Z.Q., Pang, J.S., Ralph, D.: Mathematical Programs with Equilibrium Constraints. Cambridge University Press (1996). DOI 10.1017/CBO9780511983658
44. March, A., Willcox, K.: Constrained multifidelity optimization using model calibration. Structural and Multidisciplinary Optimization **46**(1), 93–109 (2012). DOI 10.1007/s00158-011-0749-1
45. Martelli, E., Amaldi, E.: PGS-COM: A hybrid method for constrained non-smooth black-box optimization problems: Brief review, novel algorithm and comparative evaluation. Computers and Chemical Engineering **63**, 108–139 (2014). DOI 10.1016/j.compchemeng.2013.12.014
46. Martins, J., Marriage, C., Tedford, N.: pyMDO: An object-oriented framework for multidisciplinary design optimization. ACM Transactions on Mathematical Software **36**(4), 20:1–20:25 (2009). DOI 10.1145/1555386.1555389
47. Matott, L., Leung, K., Sim, J.: Application of MATLAB and Python optimizers to two case studies involving groundwater flow and contaminant transport modeling. Computers & Geosciences **37**(11), 1894–1899 (2011). DOI 10.1016/j.cageo.2011.03.017
48. Müller, J., Day, M.: Surrogate optimization of computationally expensive black-box problems with hidden constraints. INFORMS Journal on Computing **31**(4), 689–702 (2019). DOI 10.1287/ijoc.2018.0864
49. Nemhauser, G., Savelsbergh, M., Sigismondi, G.: Constraint classification for mixed integer programming formulations. COAL Bulletin **20**, 8–12 (1992). http://alexandria.tue.nl/repository/books/365757.pdf
50. Plantenga, T.: HOPSPACK 2.0 User Manual. Tech. Rep. SAND2009-6265, Sandia National Laboratories, Livermore, California (2009). URL http://www.sandia.gov/hopspack/HopspackUserManual_2_0_2.pdf
51. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006). URL https://www.elsevier.com/books/handbook-of-constraint-programming/rossi/978-0-444-52726-4
52. Stripinis, L., Paulavičius, R.: A new DIRECT-GLh algorithm for global optimization with hidden constraints. Optimization Letters **15**(6), 1865–1884 (2021). DOI 10.1007/s11590-021-01726-z
53. Tosserams, S., Etman, L., Rooda, J.: A classification of methods for distributed system optimization based on formulation structure. Structural and Multidisciplinary Optimization **39**, 503–517 (2009). DOI 10.1007/s00158-008-0347-z