A Taxonomy of Constraints in Simulation-Based Optimization

Sébastien Le Digabel* **and Stefan M. Wild†**

Abstract. The types of constraints encountered in black-box and simulation-based optimization problems differ significantly from those treated 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 QRAK, 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.

Key words. Taxonomy of constraints, Black-box optimization, Simulation-based optimization.

AMS subject classifications. 90C30, 90C56, 65K05

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

$$\min_{x \in \Omega} f(x), \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ denotes an extended-value objective function. We propose a taxonomy of constraints, denoted QRAK, whose development is motivated by the field of *derivative-free optimization* (DFO), and more precisely *black-box optimization* (BBO) and *simulation-based optimization* (SBO). In BBO/SBO, the objective function f and/or some constraints defining an instance of Ω are, or can depend on, the outputs of one or more *black-box simulations*. We assume that SBO is the more general term; hence, we use it in the title of this work. In typical settings, evaluating the simulation(s) is the primary bottleneck for an optimization algorithm; the time required to evaluate algebraic terms associated with other constraints or the objective is inconsequential relative to the time required to evaluate the simulation components. In addition, simulations may sometimes fail to return a value, even for points inside Ω.

Our taxonomy addresses a specific instance (or "description") of Ω. This instance, rather than the mathematical problem (1.1), will be passed to an optimization solver (which may do some preprocessing of its own and then tackle a different instance).

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

$$\min_{x \in \mathbb{R}^2} \{x_1 + x_2 : x_1 \geq 0,\ x_2 \geq 0\}. \tag{1.2}$$

In fact, many instances of the feasible set Ω share a solution set with (1.2). For example, a different description can yield the same feasible set, either by chance,

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

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

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. http://www.mcs.anl.gov/~wild/.

or as a result of some redundancy,

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

Or, the feasible sets can differ from instance to instance, but the minimizers of f over the sets are the same, whether indirectly,

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

or explicitly,

$$\Omega_4 = \{x \in \mathbb{R}^2 : x_1 = 0,\ x_2 = 0\}.$$

In situations similar to these examples, one likely expects that a modern solver or modeling language—or even more classical techniques such as Fourier-Motzkin elimination—would perform preprocessing that would address redundancies, inefficiencies, and the like before invoking the heaviest machinery of a solver. However, when the problem involves some black-box or simulation component, the situation, and hence such preprocessing, can be considerably more difficult.

More generally, the proposed classification is not absolute: it depends on the entire set of constraint models specified in the instance and on the information that the problem/simulation designer gives. For example, a simple bound constraint may be indicated as the output of a black box rather than expressed algebraically, leading to two different classes in the taxonomy. Other examples of different constraints changing class will be described after the taxonomy has been introduced.

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

$$\Omega = \{x \in \mathbb{R}^n : c_i(x) = 0, \forall i \in \mathcal{I};\ \ c_j(x) \leq 0, \forall j \in \mathcal{J};\ \ c_k(x) \in \mathcal{A}_k, \forall k \in \mathcal{K}\}, \tag{1.3}$$

where $\mathcal{I}, \mathcal{J}, \mathcal{K}$ are finite and possibly empty. 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 (1.3) is general enough to include cases when a variable changes the total number of decision variables (such as when determining the number of bus stations to build as well as their locations).

As underscored in the recent book by Conn et al. [15], derivative-free optimization in the presence of general constraints has not yet been fully addressed in the algorithmic literature or in benchmark papers such as [35] or [38]. Even in broader SBO fields such as 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. 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 formulating problems.

The paper is structured as follows. Section 2 presents the taxonomy of constraints for SBO and describes the different classes. It also illustrates the taxonomy through practical examples and situations. Section 3 puts the taxonomy in perspective with the existing literature. Putting the literature review toward the end of the paper here is deliberate and eases the presentation. Section 4 summarizes our contributions and discusses extensions to the taxonomy.

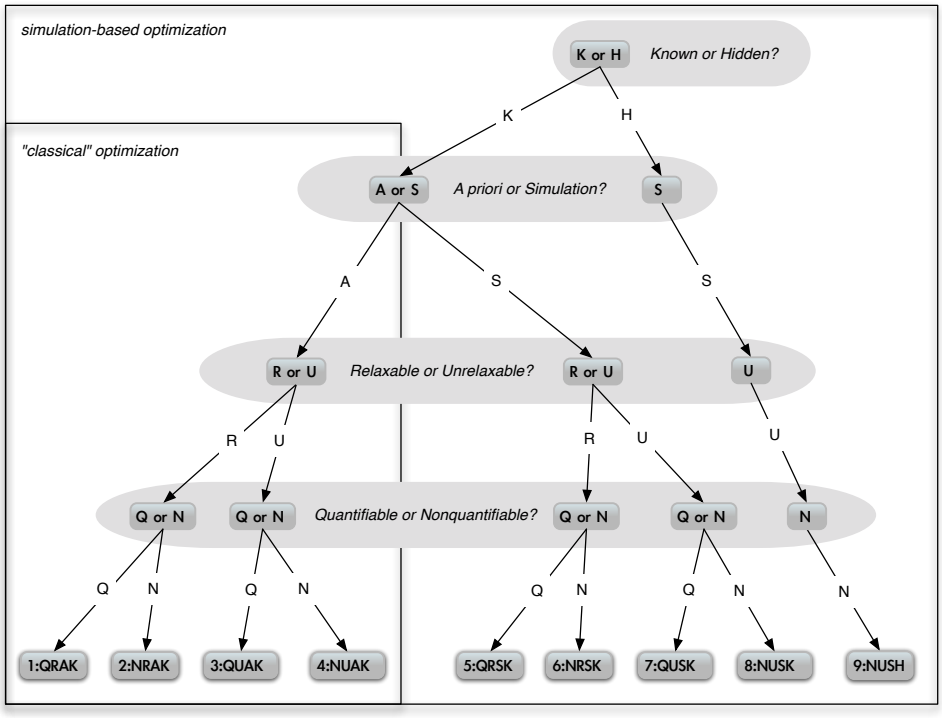

FIG. 1. *Tree-based view of the* QRAK *taxonomy of constraints. Each leaf corresponds to a class of constraints.*

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

The letters defining the acronym of the taxonomy correspond to four types of left branches in the tree: Q is for Quantifiable, R is for Relaxable, A is for A priori, and K is for Known. The corresponding right branches are identified with N for Nonquantifiable, U for Unrelaxable, S for Simulation, and H for Hidden.

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

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, which outlines a preference for practitioners to model constraints such that they appear in the most possible left part of the tree. Further subdivisions (convexity, nonlinearity, etc.) are also important but more focused on the NLO case and hence not discussed here.

Every constraint in an SBO problem instance fits in one leaf of the tree. However, a constraint type from a classification scheme different from QRAK (e.g., bound con-

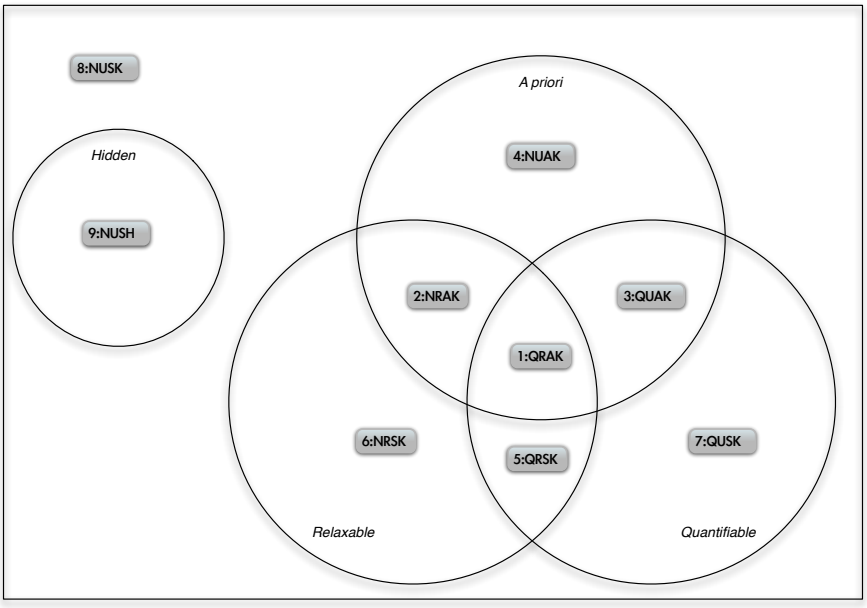

FIG. 2. *Venn diagram of the taxonomy of constraints. Each region corresponds to a leaf in the tree of Figure 1.*

TABLE 1

The taxonomy as a table where each column 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)

straint, nonlinear equality constraint) can correspond to several QRAK leaves at once. In this case, we use the generic wildcard notation "*." For example, depending on the context, a bound constraint can be relaxable or unrelaxable. It is clearly, however, a constraint that is known, a priori, and quantifiable. In this case, the bound constraint is identified by Q*AK. The wildcard is not systemically used when the sense is obvious: For example, we simply write S instead of **S*. These issues will appear as natural as we proceed with examples and formal definitions of each class/level of the tree, starting from the bottom and moving to the top.

2.1. Quantifiable (Q) versus nonquantifiable (N). For a nonquantifiable constraint, one has only a binary indicator saying 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* in order to avoid confusion with *measurable* in analysis.

DEFINITION 2.1. *A quantifiable constraint is a constraint for which the degree of feasibility and/or violation can be quantified. A nonquantifiable constraint is one for which the degrees of satisfying or violating the constraint are both unavailable.*

The definition of a quantifiable constraint does not guarantee that measures of both feasibility and violation are available. In particular, both of the following are examples 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 $T - \hat{t}$ measures how close one was 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 a method or solver point of view, the distinction between Q and U clearly is important. For example, if one wants to build a model of the constraint, Q might imply *interpolation* whereas U might imply *classification*.

2.2. Relaxable (R) versus unrelaxable (U). The next notion addressed by the taxonomy is that of relaxability.

DEFINITION 2.2. *A relaxable constraint is a constraint that does not need to be satisfied in order to obtain* meaningful *outputs from the simulations in order to compute the objective and the constraints. An unrelaxable constraint is one that must be satisfied for meaningful outputs to be obtained.*

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 implication regarding this R versus U property is that all the iterates must satisfy unrelaxable constraints, while relaxable constraints need be satisfied only at the proposed solution. Said differently, infeasible points may be considered as intermediate (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 3.

2.3. A Priori (A) versus simulation-based (S). A simulation constraint is specific to BBO/SBO. The nature of a simulation constraint is such that a potentially costly call to a computer simulation must be launched in order to evaluate the constraint. We note, however, that this constraint evaluation may not ultimately

prove to be costly. For example, the simulation could include a constraint that is cheap to evaluate and can be used as a flag to avoid any further computation; such a constraint is still defined by our taxonomy to be an S constraint (more specifically, a *USK constraint).

DEFINITION 2.3. *An a priori constraint is a constraint for which feasibility can be confirmed without running a simulation. A simulation-based constraint (or simulation constraint) requires running a simulation to verify feasibility.*

Simple examples of a priori constraints include one-sided bounds and linear equalities. A Priori constraints, however, can include very general and special formulations, such as semidefinite programming constraints, constraint programming constraints (e.g., *all different*, *ordered*), or some constraints relative to the nature of the variables: reals or integers or binary or categorical.

One can easily appreciate that a solver should want to evaluate *UA* (unrelaxable, a priori) constraints first and avoid a simulation execution if the candidate is infeasible—especially when the simulation is costly. 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, should noninteger input values be passed to a simulator that may then end up rounding to the nearest integer within the simulation? The answer depends on the context.

An alternative to "simulation" is *a posteriori* [5]; alternative terms for "a priori" include *algebraic* or *algebraically available*, *analytic*, *closed-form*, *expressible*, and *input-constraint*. An *algebraic function* is usually defined to be one that satisfies an equation that can be expressed as a finite-degree polynomial with rational coefficients. Unfortunately this definition does not include transcendental functions (e^x, etc.). Some modeling languages, such as GAMS [8], already use this term (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 idea behind the term *input-constraint* is that A constraints can be seen as simply related to the inputs x, whereas S constraints are somehow expressed as a function of the simulator.

2.4. Known (K) versus hidden (H). The final distinction in the taxonomy is specific to BBO/SBO.

DEFINITION 2.4. *A known constraint is a constraint that is explicitly given in the problem formulation. A hidden constraint is not explicitly known to the solver.*

The majority of constraints that one encounters when solving SBO problems—especially when an optimizer is involved early in the modeling and problem formulation process—are *known* to the optimizer. A hidden constraint typically (but not necessarily) appears when the simulations crashes. 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*.

A hidden constraint is not necessarily a bug in the simulator. For example, consider the problem $\min\{f(\log x) : x \in \mathbb{R}\}$ with f some simulation-based function from $\mathbb{R}$ to $\mathbb{R}$. If the constraint $x > 0$ is expressed in the description of the problem, then it is an a priori constraint. Otherwise, it is hidden and can be observed only for negative or null values of x. This constraint may have been stated explicitly inside the simulator in order to avoid a crash and raise some flag; but as far as it not indicated to the solver, it remains H.

As shown in Figure 1, the H branch of the tree is the only one that goes directly to

a terminal leaf. A hidden constraint cannot be a priori (by definition) and quantifiable (we do not know what to quantify). A hidden constraint also cannot be relaxable since the violation/satisfaction cannot be detected if the outputs are always meaningful.

Note that the boundary between a hidden (NUSH) constraint and a NUSK constraint is thin. 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.

2.5. Short case studies. The previous examples were related to the four levels of decision in the taxonomy. We now show that each of the nine leaves of the tree in Figure 1 (similarly, each row of Table 1) is nonempty, and we illustrate some situations that belong to each leaf.

1: QRAK (Quantifiable Relaxable A Priori Known): Probably the most common type of constraint found in classical nonlinear optimization.
   - $\sum_{i=1}^{n} x_i \leq 100$: If each x_i represents an amount of money, this constraint defines a budget.
   - Relaxable discrete variable: $x_i \in \{0, 1\}$ for some indices i. Then $\min\{|x_i|, |1 - x_i|\}$ provides the violation measure.

2: NRAK (Nonquantifiable Relaxable A Priori Known): A good example is a categorical variable constrained to a subset of its possible values:
   - A simulator can work in two modes depending on the value of a binary flag $x \in \{0, 1\}$. If $x = 1$, the simulation is costly but more accurate. If $x = 0$, it is cheap but imprecise. We want a solution that has been validated with $x = 1$, but an optimization algorithm can set $x = 0$ at intermediate points. The NRAK constraint is "$x = 1$".
   - Consider a simulator that drives a C++ compilation, with the two categorical variables $x_1 \in \{\mathsf{gcc}, \mathsf{icc}\}$ 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 these two constraints are of type NRAK. (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: QUAK (Quantifiable Unrelaxable A Priori Known):
   - Well rates in groundwater problems: If we can simulate only extraction (and not injection), then the constraints $r_i \geq 0$ for all well indexes i are of type QUAK.
   - Decision variables must be ordered or be all different.

4: NUAK (Nonquantifiable Unrelaxable A Priori Known): Categorical variables are a typical example: $\mathsf{compiler} \in \{\mathsf{gcc}, \mathsf{icc}\}$.

5: QRSK (Quantifiable Relaxable Simulation Known): Simply consider a requirement on one of the simulation's output, such as the following:
   - A budget based on economical criteria, $S(x) \leq b$.
   - In the context of optimization of algorithm parameters, the percentage of problems solved by the algorithm under consideration must be 100%.

6: NRSK (Nonquantifiable Relaxable Simulation Known):
   - A simulator displays a flag indicating whether a toxicity level has been reached during the simulation, but we know neither when this occurred nor the level of toxicity.
   - The simulator indicates whether the power consumption remained under

100W, but we have access only to the notification.

7: QUSK (Quantifiable Unrelaxable Simulation Known): One of the outputs $c_S(x)$ of the simulation is a concentration level; if it is below zero, the simulation stops and displays NaN for all the outputs except c_S.

8: NUSK (Nonquantifiable Unrelaxable Simulation Known):
- A flag indicates that the convergence of some specific and identified numerical method inside the simulation could not converge.
- An error number/code with associated documentation is obtained.

These are not hidden constraints since the reason for the violation can be identified. However, a single binary flag indicating that the simulation failed is considered as a hidden constraint. In the same way, an error message that cannot be interpreted is equivalent to such a flag and hence should be interpreted as hidden.

9: NUSH (Hidden):
The simulation failed to complete and nothing is displayed, or a simple flag is raised or an undocumented error number indicated.

3. Literature review. In this section, we review the existing literature and collect terminology from the BBO, DFO, and SBO communities in order 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 QRAK and the more formal definitions on which the taxonomy is built. Some of the terms from the literature may have been used to define an alternative classification of the constraints, and some of them have already been mentioned in our presentation, such as *soft* versus *hard* in Section 2.2. We also survey early uses of various terms (e.g., hidden constraints) for a historical perspective, and we illustrate the use of the taxonomy in the context of modeling languages, algorithms, and some specific applications.

Before proceeding, we note that the proposed classification is not related to the constraint programming field [39], where, within a specific context, constraints can be expressed as logical prepositions treated by specialized algorithms.

3.1. Hidden constraints. The term *hidden constraint* corresponds to the NUSH leaf in the tree of Figure 1. It often appears in the literature on derivative-free optimization. In the modern literature, this term is typically attributed to Choi and Kelley [12], who say that a hidden constraint is "the requirement that the objective be defined." This definition is used in Kelley's implicit filtering software [28] and has been used to solve several examples (see, e.g., [9, 11]) whereby a hidden constraint is said to be violated whenever flow conditions are found that prevent a simulation solution from existing. The term is also used by the authors of the SNOBFIT package [26] 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. A more recent reference to the term is in [23], where hidden constraints are "constraints which 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."

In fact, the term had been previously used in the context of optimization. The earliest published instance of hidden constraint that we are aware of is from 1967 [7] 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 high enough to justify use of the equations in the calculations.

3.2. *Introduction to Derivative-Free Optimization* textbook. Although the DFO book [15] focuses on unconstrained optimization, it uses definitions of relaxable and hidden constraints similar to those used in QRAK. The book mentions that unrelaxable constraints "have to be satisfied at all iterations" of an algorithm while "relaxable constraints need only be satisfied approximately or asymptotically." Moreover, constraints for which derivatives are not available and which are typically given by a black box, are denoted as *derivative-free constraints*. Although in general these constraints can be treated as relaxable, some situations require them to be unrelaxable. Doing so demands a feasible starting point, which may be difficult to obtain in practice. Moreover, hidden constraints are seen as an extreme case of such unrelaxable constraints. They are defined as constraints 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 [15] 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.

Scheinberg et al. [13] 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.

3.3. Unrelaxable and relaxable constraints. The terms *unrelaxable* and *relaxable* are widely used in the literature (see [15]). The related notions of *hard* and *soft* constraints appear almost as frequently but with several different meanings. Here, we follow the convention of [25]:

> "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 [17]. Another example comes from SNOBFIT [26], where soft constraints are "constraints which need not be satisfied accurately." Other uses of hard and soft constraint can be found, for example, in [24]. 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 [23]: "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 [33], 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."

3.4. Modeling languages and applications. Several modeling languages and collections of test problems, such as AMPL [18], CUTEst [21], GAMS [8], or ZIMPL [29], 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.

The QRAK taxonomy can be illustrated on the following examples of SBO problems from the recent literature. The community groundwater problem [19] has only bound constraints (**AK), while the LOCKWOOD problem [34] has a linear objective and simulation constraints (S); different simulation-based instances of the LOCKWOOD constraints are considered in [27] alongside solution methodologies for the resulting formulations. The STYRENE problem from [4] 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.

3.5. Algorithms and software for constrained problems. To motivate the opportunities that such a constraint taxonomy affords, we briefly describe how some algorithms and software address different types of constraints, using the taxonomy syntax.

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 [26] treats *soft* and also NUSH (hidden) constraints. The software SID-PSM [16] handles constraints with derivatives and U (unrelaxable) constraints. The DFO code [14] (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 [14], 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 (nonquantifiable) constraints and recommend using an extreme-barrier approach. The HOPSPACK package [37] 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 [30], the progressive-barrier technique [6] 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 [20]. 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 NAND approach, the state variables are included as decision variables and hence the PDE reduces to a set of algebraic equations (and therefore *A*K constraints in our taxonomy).

3.6. Other related work. Previous classifications also have been proposed. An example is the mixed-integer programming classification [36] for linear inequalities, linear equations, continuous parameters, and discrete parameters.

The closest related work toward a more compete characterization of constraints is that of Alexandrov and Lewis [3], who examined different formulations for general problems arising in multidisciplinary optimization (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 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., [40]).

The notion of unknown constraints appears in [22] but is not equivalent to its use in our taxonomy; rather, it corresponds to constraints given by a black box. Note that the same authors and others addressed hidden constraints in [31].

Additional terms for describing general constraints are found in the literature. For example, *chance constraints* [10] 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 [2] for an example. Other terms include the notions of *vanishing* constraints [1], *complementarity* constraints, or *variational inequalities* [32]. Conn et al. describe *easy* constraints and *difficult* constraints as follows [13]:

> "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 the *derivative-free* constraints described in [15]. This characterization differs from our proposed taxonomy, which does not seek to guarantee an ordering with regard to the computational expense of evaluating a constraint and/or establishing feasibility with respect to the constraint.

4. Discussion. This work proposes a unification of past conventions and terms into a single taxonomy, denoted QRAK, 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 extensions to the taxonomy. One can refine the tree in

Figure 1, depending on the context, by adding subcases to the leaves. Such extensions within QRAK could include stochastic, convex, linear, and smooth constraints (i.e., constraints for which derivatives are available). Equality, inequality, or set membership is also an option: For example, an equality N*SK constraint is difficult (impossible?) to treat, whereas an equality Q*AK constraint may be easy. 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.

Acknowledgments. This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Contract 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 for his useful comments and suggestions.

REFERENCES

[1] W. Achtziger and C. Kanzow, *Mathematical programs with vanishing constraints: optimality conditions and constraint qualifications*, Mathematical Programming, 114 (2008), pp. 69–99.

[2] V. Aggarwal, Y. P. Aneja, and K. P. K. Nair, *Minimal spanning tree subject to a side constraint*, Computers and Operations Research, 9 (1982), pp. 287–296.

[3] N. M. Alexandrov and R. M. Lewis, *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.

[4] C. Audet, V. Béchard, and S. Le Digabel, *Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search*, Journal of Global Optimization, 41 (2008), pp. 299–318.

[5] C. Audet, C.-K. Dang, and D. Orban, *Efficient use of parallelism in algorithmic parameter optimization applications*, Optimization Letters, 7 (2013), pp. 421–433.

[6] C. Audet and J. E. Dennis, Jr., *A progressive barrier for derivative-free nonlinear programming*, SIAM Journal on Optimization, 20 (2009), pp. 445–472.

[7] Mordecai Avriel and D. J. Wilde, *Optimal condenser design by geometric programming*, Industrial & Engineering Chemistry Process Design and Development, 6 (1967), pp. 256–263.

[8] A. Brooke, D. Kendrick, and A. Meeraus, *GAMS: A Users' Guide*, The Scientific Press, Danvers, Massachusetts, 1988.

[9] R. G. Carter, J. M. Gablonsky, A. Patrick, C. T. Kelley, and O. J. Eslinger, *Algorithms for noisy problems in gas transmission pipeline optimization*, Optimization and Engineering, 2 (2001), pp. 139–157.

[10] A. Charnes, W. W. Cooper, and G. H. Symonds, *Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil*, Management Science, 4 (1958), pp. 235–263.

[11] T. D. Choi, O. J. Eslinger, C. T. Kelley, J. W. David, and M. Etheridge, *Optimization of automotive valve train components with implicit filtering*, Optimization and Engineering, 1 (2000), pp. 9–27.

[12] T. D. Choi and C. T. Kelley, *Superlinear convergence and implicit filtering*, SIAM Journal on Optimization, 10 (2000), pp. 1149–1162.

[13] A. R. Conn, K. Scheinberg, and Ph. L. Toint, *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.

[14] A. R. Conn, K. Scheinberg, and Ph. L. Toint, *DFO (derivative free optimization software)*. https://projects.coin-or.org/Dfo, 2001.

[15] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*, MOS-SIAM Series on Optimization, SIAM, Philadelphia, 2009.

[16] A. L. Custódio and L. N. Vicente, *SID-PSM: A pattern search method guided by simplex derivatives for use in derivative-free optimization*. http://www.mat.uc.pt/sid-psm, 2005.

[17] A. V Fiacco and G. P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, Classics in Applied Mathematics 4, SIAM, 1990.

[18] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Thomson/Brooks/Cole, Pacific Grove, California, second ed., 2003.

[19] K. R. Fowler, J. P. Reese, C. E. Kees, J. E. Dennis, Jr., C. T. Kelley, C. T. Miller, C. Audet, A. J. Booker, G. Couture, R. W. Darwin, M. W. Farthing, D. E. Finkel, J. M. Gablonsky, G. Gray, and T. G. Kolda, *Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems*, Advances in Water Resources, 31 (2008), pp. 743–757.

[20] O. Ghattas, L.T. Biegler, M. Heinkenschloss, and B. van Bloemen Wanders, *Large-scale pde-constrained optimization: An introduction*, in Large-Scale PDE-Constrained Optimization, L.T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Wanders, eds., Springer, New York, 2003, pp. 3–13.

[21] N. I. M. Gould, D. Orban, and Ph. L. Toint, *CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization*, Computational Optimization and Applications, (2014), pp. 1–13. Code available at http://ccpforge.cse.rl.ac.uk/gf/project/cutest/wiki.

[22] R. B. Gramacy and H. K. H. Lee, *Optimization under unknown constraints*, in Proceedings of the ninth Valencia International Meetings on Bayesian Statistics, J. Bernardo, S. Bayarri, J.O. Berger, A.P. Dawid, D. Heckerman, A.F.M. Smith, and M. West, eds., Oxford University Press, 2011, pp. 229–256.

[23] S. Gratton and L. N. Vicente, *A merit function approach for direct search*, SIAM Journal on Optimization, 24 (2014), pp. 1980–1998.

[24] J. D. Griffin and T. G. Kolda, *Nonlinearly constrained optimization using heuristic penalty methods and asynchronous parallel generating set search*, Applied Mathematics Research eXpress, 2010 (2010), pp. 36–62.

[25] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, SIAM, 2009.

[26] W. Huyer and A. Neumaier, *SNOBFIT – Stable noisy optimization by branch and fit*, ACM Transactions on Mathematical Software, 35 (2008), pp. 9:1–9:25.

[27] Aswin Kannan and Stefan M. Wild, *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), June 2012.

[28] C. T. Kelley, *Implicit Filtering*, SIAM, Philadephia, PA, 2011.

[29] T. Koch, *Rapid Mathematical Prototyping*, PhD thesis, Technische Universität Berlin, 2004.

[30] S. Le Digabel, *Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm*, ACM Transactions on Mathematical Software, 37 (2011), pp. 44:1–44:15.

[31] H. K. H. Lee, R. B. Gramacy, C. Linkletter, and G. A. Gray, *Optimization subject to hidden constraints via statistical emulation*, Pacific Journal of Optimization, 7 (2011), pp. 467–478.

[32] Z.-Q. Luo, J.-S. Pang, and D. Ralph, *Mathematical Programs with Equilibrium Constraints*, Cambridge University Press, 1996.

[33] E. Martelli and E. Amaldi, *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 (2014), pp. 108–139.

[34] L. S. Matott, K. Leung, and J. Sim, *Application of MATLAB and Python optimizers to two case studies involving groundwater flow and contaminant transport modeling*, Computers & Geosciences, 37 (2011), pp. 1894–1899.

[35] J. J. Moré and S. M. Wild, *Benchmarking derivative-free optimization algorithms*, SIAM Journal on Optimization, 20 (2009), pp. 172–191.

[36] G. L. Nemhauser, M. W. P. Savelsbergh, and G. S. Sigismondi, *Constraint classification for mixed integer programming formulations*, COAL Bulletin, 20 (1992), pp. 8–12. http://alexandria.tue.nl/repository/books/365757.pdf.

[37] T. D. Plantenga, *HOPSPACK 2.0 user manual*, Tech. Report SAND2009-6265, Sandia National Laboratories, Livermore, California, October 2009.

[38] L. M. Rios and N. V. Sahinidis, *Derivative-free optimization: a review of algorithms and comparison of software implementations*, Journal of Global Optimization, 56 (2013), pp. 1247–1293.

[39] F. Rossi, P. van Beek, and T. Walsh, eds., *Handbook of Constraint Programming*, Elsevier, 2006.

[40] S. Tosserams, L. Etman, and J. Rooda, *A classification of methods for distributed system optimization based on formulation structure*, Structural and Multidisciplinary Optimization, 39 (2009), pp. 503–517.