SIAM J. OPTIM.
Vol. 29, No. 2, pp. 1164–1189

THE MESH ADAPTIVE DIRECT SEARCH ALGORITHM FOR GRANULAR AND DISCRETE VARIABLES*

CHARLES AUDET†, SÉBASTIEN LE DIGABEL†, AND CHRISTOPHE TRIBES†

Abstract. The mesh adaptive direct search (MADS) algorithm is designed for blackbox optimization problems for which the functions defining the objective and the constraints are typically the outputs of a simulation seen as a blackbox. It is a derivative-free optimization method designed for continuous variables and is supported by a convergence analysis based on the Clarke calculus. This work introduces a modification to the MADS algorithm so that it handles granular variables, i.e., variables with a controlled number of decimals. This modification involves a new way of updating the underlying mesh so that the precision is progressively increased. A corollary of this new approach is the ability to treat discrete variables. Computational results are presented using the NOMAD software, the free C++ distribution of the MADS algorithm.

Key words. blackbox optimization, derivative-free optimization, mesh adaptive direct search, granular variables, discrete variables

AMS subject classifications. 90C11, 90C30, 90C56

DOI. 10.1137/18M1175872

1. Introduction and context.

1.1. Motivation. This work studies derivative-free optimization (DFO) [12, 26], and more precisely blackbox optimization, where the functions defining the problem are obtained from the execution of a simulation seen as a blackbox. We focus on the case where a minimal granularity is imposed on some or all variables, which we call *granular variables*. Integers are an example in which the granularity equals one. The situation in which a variable has a fixed number of decimals is another. For example, a variable may represent a setting on a machine that takes values from the set $\{0, 0.05, 0.10, 0.15, \ldots\}$.

The mesh adaptive direct search (MADS) algorithm [9] is designed for blackbox optimization problems, but the algorithm was conceived with continuous variables in mind. In order to explore the space of variables $\mathbb{R}^n$, MADS uses a discretized mesh whose coarseness is parameterized by a scalar that is typically an integer power of four. In [16], the scalar is replaced by a vector in $\mathbb{R}^n$ to allow an anisotropic mesh and the ability to achieve automatic scaling. The coarseness or fineness of the mesh is adjusted at the end of each iteration.

In the present work, we propose a new way to update the mesh size vector from one iteration to another. The new strategy harmonizes the minimal granularity of variables with the finest mesh containing all trial points. Another advantage of our proposed approach is that the number of decimals in the trial points is controlled. The motivation for this feature is made clearer by the following context: in [17], the authors study the question of finding values of the four standard parameters of a

*Received by the editors March 16, 2018; accepted for publication (in revised form) January 18, 2019; published electronically April 23, 2019.
http://www.siam.org/journals/siopt/29-2/M117587.html
Funding: This work is supported by the NSERC CRD RDCPJ 490744-15 grant and by an InnovÉÉ grant, in collaboration with both Hydro-Québec and Rio Tinto.
†GERAD and Département de mathématiques et génie industriel, Polytechnique Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Québec H3C 3A7, Canada (Charles.Audet@gerad.ca, https://www.gerad.ca/Charles.Audet, sebastien.le-digabel@polymtl.ca, https://www.gerad.ca/Sebastien.Le.Digabel, christophe.tribes@polymtl.ca).

trust-region algorithm to minimize the computational time for solving a collection of problems. The values proposed in many textbooks are

$$p^C = \left(\frac{1}{4},\ \frac{3}{4},\ \frac{1}{2},\ 2\right).$$

The authors of [17] suggest the values

$$\overline{p} = (0.22125,\ 0.94457031,\ 0.37933594,\ 2.3042969),$$

which reduce the computational time by 25% on their test bed. These proposed values might be interesting for the theoretician, but are difficult to popularize because of the large number of decimals. A solution limited to two or three significant figures would possibly have been used more often.

Another motivating example of granularity is the application described in [67], on the optimization of an aircraft departure procedure. In practice, this kind of optimization is performed within the cockpit, on a laptop that is not connected to the aircraft systems. Controlling the number of decimals of the solution reduces the risk of error when the pilot manually enters it. More generally, granular variables are of interest whenever a solution must be implemented in practice, on systems for which precision is an issue.

The present work considers the following optimization problem:

$$\begin{array}{ll} \min\limits_{x\in\mathcal{X}\subseteq\mathbb{R}^n} & f(x) \\ \text{s.t.} & c(x) \le 0, \end{array} \tag{1.1}$$

where $\mathcal{X} = \{x \in \mathbb{R}^n : \ell \le x \le u \text{ and } x_i/\delta_i^{\min} \in \mathbb{Z} \text{ for all } i \in \mathcal{G}\}$. The functions $f : \mathbb{R}^n \mapsto \mathbb{R}\cup\{\infty\}$ and $c : \mathbb{R}^n \mapsto \{\mathbb{R}\cup\{\infty\}\}^m$ are typically the outputs (or functions of the outputs) of a simulation, a *blackbox* whose explicit nature cannot be exploited by the optimization method. For example, the computation of these functions may require the launch of a time-consuming computer simulation or a laboratory experiment. A survey of derivative-free and blackbox optimization can be found in [12].

The vectors $\ell, u \in \{\mathbb{R} \cup \{\pm\infty\}\}^n$ represent bounds for the n variables. We introduce $\mathcal{G}$ to be the set of indices in $N = \{1, 2, \ldots, n\}$ for which the user provides a minimal granularity, $\delta_i^{\min} > 0$ for $i \in \mathcal{G}$. This means that the ith component of any trial point must be an integer multiple of the granularity $\delta_i^{\min}$, as imposed by the requirement $x_i/\delta_i^{\min} \in \mathbb{Z}$ in the definition of the set $\mathcal{X}$. Variables whose indices are in $\mathcal{G}$ are called *granular variables*. The other variables of the problem do not have a minimal granularity; they are treated as and called *continuous*. With the above notation, the following different cases are illustrated below.

- The variable x_i is an integer if $\delta_i^{\min} = 1$.
- The variable x_i is Boolean if $\delta_i^{\min} = 1$, $\ell_i = 0$, and $u_i = 1$.
- A precision of two decimals is required for the variable x_i when $\delta_i^{\min} = 0.01$.
- The variable x_i represents an angle multiple of 30° when $\delta_i^{\min} = \pi/6$.

Using the taxonomy of constraints of [41], the bounds and granularity constraints defining $\mathcal{X}$ are a priori: $x \in \mathcal{X}$ can be verified without executing the blackbox. In addition, these constraints are defined as *unrelaxable*, meaning that all intermediate solutions must remain in $\mathcal{X}$. We assume that $f(x)$ and $c(x)$ return the value ∞ whenever $x \in \mathbb{R}^n \setminus \mathcal{X}$. The other constraints $c(x) \le 0$ are considered *relaxable*, i.e., they can be violated at intermediate solutions, and *quantifiable* (a distance to feasibility and infeasibility is available). In practice, this means that the constraints $c(x) \le 0$ can be treated by the *progressive barrier* approach [10].

Of course, with appropriate scaling, any mixed-integer solver can treat granular variables. For example, the NOMAD [3, 40] implementation of MADS previously treated integer variables by rounding and imposing a minimal mesh size parameter of one, as proposed in [2]. Hence, with appropriate scaling, it may be applied to problems with granular variables. The approach adopted in the present work is not based on the scaling of discrete variables. Rather, granular variables are treated natively and iterates are generated with more and more precision until the desired number of decimals is reached for all variables. This new strategy has become available since release 3.9.0 of NOMAD.

1.2. Literature on DFO with mixed variables. To the best of our knowledge, no work has been done in DFO regarding granular variables. The following literature review focuses on handling discrete variables.

A special case of discrete optimization considers the presence of categorical variables. These variables have no ordering property and their discrete nature is unrelaxable. The presence of such variables is particularly common when simulations are involved. For example, they can represent types of materials. This special structure is not assumed in the present work, but algorithms designed to solve problem (1.1) can be applied, as long as these algorithms generate only valid discrete values for the categorical variables. Several DFO methods are specialized for categorical variables [2, 4, 7, 47, 48, 64]. Applications of these algorithms can be found in [1, 38, 76].

In the DFO literature, several papers consider mixed-integer problems with problems containing both discrete and continuous variables. In particular, line searches are used in [43, 44, 45], where several algorithmic variations are introduced, and for which global convergence based on a sufficient decrease is proved. The most recent algorithms handle general constraints with a sequential penalty approach. These methods are publicly available at the Derivative-Free Library (DFL) [60], except for the more recent technical report [45]. An example of a realistic application solved using this library is available in [46]. Other approaches based on surrogate models are given in [24, 27, 55, 56, 57]. One of them, surrogate optimization-integer (SO-I), considers constraints with a two-phase framework, where feasibility is the priority of the first phase. The mixed-integer surrogate optimization (MISO) framework implementation for the box-constrained case, in MATLAB, is available at J. Mueller's homepage.[1] Surrogate models are also used for a specific application in [34], search directions for the box-constrained case in [30], and projection in [70]. Other DFO methods have also been adapted to the mixed-integer case: the brute force optimizer (BFO) algorithm [61] is a mesh-based direct-search method for bound-constrained problems using polling directions. The associated MATLAB code is freely available. The derivative-free trust-region approach is also considered in [58] for box-constrained problems, and the Nelder–Mead algorithm is the subject of [20, 73]. Articles in engineering journals, such as [71, 74], also consider the solution of mixed-integer blackbox problems. Finally, several heuristics exist, such as [21, 22, 32, 39, 42, 63, 69], but there is no theoretical background to support their use.

The objective of this work is to introduce a new way to update the mesh parameters within MADS for problems with any mixture of continuous, integer, and granular variables. This method is presented as Algorithm 5 in section 3.3. It is the result of several steps detailed throughout sections 2 and 3. While section 2 presents a simple polling algorithm as well as an extension to control the number of decimals,

[1] See https://ccse.lbl.gov/people/julianem/ (last accessed on 12 October 2018).

section 3 recalls the dynamic scaling of individual variables and then introduces a mechanism to handle the minimal granularity of variables—and implicitly, integers. In addition to gradually introducing the new method, these two sections provide a novel and improved description of MADS that includes many updates introduced since 2006. Computational experiments are conducted in section 4, both on analytical problems from the literature and on a realistic blackbox application. The behavior of the proposed algorithm is tested for problems with continuous variables only, then on problems with granular variables only, and finally on mixed-integer problems. The new algorithm is also tested with the RobustMADS algorithm [15] on the motivating stochastic optimization problem outlined in section 1.1.

2. Polling algorithms. One of the two main steps of the MADS algorithm is the poll. This section is divided into four subsections. The first one illustrates the simplest strictly polling algorithm. Section 2.2 introduces a new mechanism that allows us to control the number of decimals for that simple algorithm. Then, section 2.3 applies the mechanism to a broader algorithmic class. The last subsection details how to easily generate a dense set of polling directions.

2.1. The coordinate search algorithm. The simplest derivative-free algorithm for the unconstrained minimization of a function f from $\mathbb{R}^n$ to $\mathbb{R}$ is the coordinate search (CS) algorithm [29]. To launch this algorithm, one simply needs to supply an initial point $x^0 \in \mathbb{R}^n$.

CS is an iterative algorithm. At iteration k, CS attempts to find another trial point whose objective function value is strictly less than $f(x^k)$ by testing trial points in the positive and negative coordinate directions with respect to the current incumbent solution x^k, using a step size of $\delta^k > 0$. If one of these trial points improves the objective function value, then that trial point becomes x^{k+1}, and the iteration ends and is labelled as successful. If none of these trial points improves f, the iteration is said to be unsuccessful, x^{k+1} is set to x^k, and the step-size parameter is cut in half: $\delta^{k+1} = \frac{1}{2}\delta^k$.

We use the following terminology. The set of tentative trial points at which f may be computed during the kth iteration is called the poll set and is denoted by P^k. In the context of CS, $P^k = \{x^k \pm \delta^k e_i : i \in N\}$ has exactly $2n$ elements ($e_i \in \mathbb{R}^n$ denotes the ith coordinate direction). We write "test P^k" to indicate that f is evaluated at trial points in P^k. The test process terminates opportunistically as soon as a trial point $t \in P^k$ produces an objective function value $f(t)$ strictly less than the incumbent value $f(x^k)$. Therefore, it is likely that at successful iterations the number of calls to f will be strictly less than $2n$, the cardinality of P^k. Later, when applying the algorithm to constrained optimization, the term "test P^k" will be used in the sense described in [10]: the iteration terminates if either a better feasible point or an infeasible point that dominates an incumbent is found.

Algorithm 1 details a slightly more evolved version of CS that doubles the step size at the end of successful iterations rather than keeping it constant (as proposed in [66]). This allows the step-size parameter to adapt if $\delta^0 = 1$ was initially chosen too small.

There are two possible behaviors for this simple algorithm. Either the sequence of trial points $\{x^k\}_{k=0}^{\infty}$ belongs to a bounded set, or it does not. If it is unbounded, then there is a subsequence $\{x^k\}_{k\in K}$ of trial points whose corresponding subsequence of objective function values $\{f(x^k)\}_{k\in K}$ is strictly decreasing, with $\{\|x^k\|\}_{k\in K} \to \infty$. Nothing more can be said in that case.

Algorithm 1. The coordinate search algorithm (CS).

Input: $x^0 \in \mathbb{R}^n$, the initial point
$\delta^0 = 1$, the initial step-size parameter

for iteration $k = 0, 1, 2, \ldots$ **do**
  poll step: test $P^k := \{x^k \pm \delta^k e_i \ : \ i \in N\}$
  if the poll step was successful at a trial point $t \in P^k$, **then**
    set $x^{k+1} = t$ and $\delta^{k+1} = 2\delta^k$
  else
    set $x^{k+1} = x^k$ and $\delta^{k+1} = \frac{1}{2}\delta^k$

However, the more frequent and more interesting situation occurs when the sequence of trial points is bounded. The following result holds.

THEOREM 2.1. *If the sequence of trial points $\{x^k\}_{k=0}^{\infty}$ produced by* CS *belongs to a bounded set, then there exists a subsequence K of indices for which the subsequence $\{x^k\}_{k\in K}$ converges and for which*

$$\lim_{k\in K} \delta^k = 0$$

and furthermore, if f is strictly differentiable near $x^* := \lim_{k\in K} x^k$, then

$$\|\nabla f(x^*)\| = 0.$$

The above theorem was shown by Torczon [66] in the context of the more general pattern search algorithm for a twice continuously differentiable function f, and in [8] when the function is only strictly differentiable.

2.2. The coordinate search algorithm with controlled decimals. A consequence of updating the step-size parameter δ^k by multiplying or dividing it by a factor 2 is that, quite rapidly, the algorithm may modify many decimals during an iteration. For example, if the algorithm starts at the origin $x^0 = 0 \in \mathbb{R}^n$ with $\delta^0 = 1$ but the three first iterations are unsuccessful, then the next poll step will make modifications of magnitude $\delta^3 = \frac{1}{8} = 0.125$ and will modify up to the third decimal. For some users, this might be an undesired behavior, as they would prefer a step size of 0.1. In such a case, we propose to address this issue by making the step-size parameter an integer multiple of a power of 10 rather than a power of 2.

At each iteration the step-size parameter δ^k will be constructed so that it belongs to the discrete set

$$\mathcal{P} := \{a \times (10)^b : a \in \{1, 2, 5\},\ b \in \mathbb{Z}\}.$$

At each iteration, the step-size parameter is the product of either 1, 2, or 5 and a positive or negative integer power of 10. At successful iterations we increase the step-size parameter, and refine it at unsuccessful iterations. For $a \times (10)^b \in \mathcal{P}$, we introduce the following operators from $\mathcal{P}$ to $\mathcal{P}$:

$$\texttt{increase}(a \times (10)^b) = \begin{cases} 2 \times (10)^b & \text{if } a = 1, \\ 5 \times (10)^b & \text{if } a = 2, \\ 1 \times (10)^{b+1} & \text{if } a = 5, \end{cases}$$

$$\texttt{decrease}(a \times (10)^b) = \begin{cases} 5 \times (10)^{b-1} & \text{if } a = 1, \\ 1 \times (10)^b & \text{if } a = 2, \\ 2 \times (10)^b & \text{if } a = 5. \end{cases}$$

The `increase` and `decrease` operators are the inverse of each other. This allows us to formulate Algorithm 2, a variant of Cs in which the number of decimals is controlled.

Algorithm 2. The coordinate search algorithm (Cs) with controlled decimals.

Input: $x^0 \in \mathbb{R}^n$, the initial point
$\delta^0 = 1 \times (10)^0 \in \mathcal{P}$, the initial step-size parameter

for iteration $k = 0, 1, 2, \ldots$ **do**
  poll step: test $P^k := \{x^k \pm \delta^k e_i : i \in N\}$
  if the poll step was successful at a trial point $t \in P^k$, **then**
    set $x^{k+1} = t$ and $\delta^{k+1} = \texttt{increase}(\delta^k)$
  else
    set $x^{k+1} = x^k$ and $\delta^{k+1} = \texttt{decrease}(\delta^k)$

This algorithm benefits from the same convergence results as in Theorem 2.1. We postpone the proof until section 3.3, where all the following algorithmic improvements are presented. We next introduce a way to use more polling directions than only the coordinate directions.

2.3. Polling with a rich set of directions. The objective function of our target problem is usually nonsmooth. The previous convergence result, Theorem 2.1, involving the limit of the step-size parameter being zero, remains valid, but the result on the gradient of f is not applicable because the gradient does not exist. We propose to modify the way that the poll set P^k is constructed so that it explores other directions than the positive and negative coordinate directions $\pm e_i$.

In order to achieve this, we split the role of the step-size parameter into two: $\Delta^k = a^k \times (10)^{b^k} \in \mathcal{P}$ is called the poll size parameter and will be used to delimit the region in which the poll points are selected; $\delta^k > 0$, called the mesh size parameter, is obtained from the integer b^k used to construct the poll size parameter,

$$\delta^k := (10)^{b^k - |b^k|}. \tag{2.1}$$

An equivalent way of writing (2.1) would be to set δ^k to 1 when $\Delta^k \geq 1$ and to the square of $(10)^{b^k}$ when $0 < \Delta^k < 1$. This means that as the poll size parameter Δ^k goes to zero the mesh size parameter will be of the order of its square: $\delta^k = \mathcal{O}((\Delta^k)^2)$. By construction, the poll size parameter Δ^k is an integer multiple of the mesh size parameter. The ratio of these two parameters is denoted by the integer ρ^k,

$$\rho^k := \frac{\Delta^k}{\delta^k} = \frac{a^k \times 10^{b^k}}{(10)^{b^k - |b^k|}} = a^k \times 10^{|b^k|} \in \mathbb{N},$$

and will play a useful role in what follows.

Now, instead of setting the poll set to $\{x^k \pm \delta^k e_i : i \in N\}$ we first construct D^k as a positive basis[2] composed of integer vectors in $\mathbb{Z}^n$ such that the infinity norm of each of its vector is less than or equal to the ratio of the poll and mesh size parameters,

[2] A positive basis in $\mathbb{R}^n$ is a minimal set of vectors whose nonnegative linear combination spans the entire space $\mathbb{R}^n$ [28].

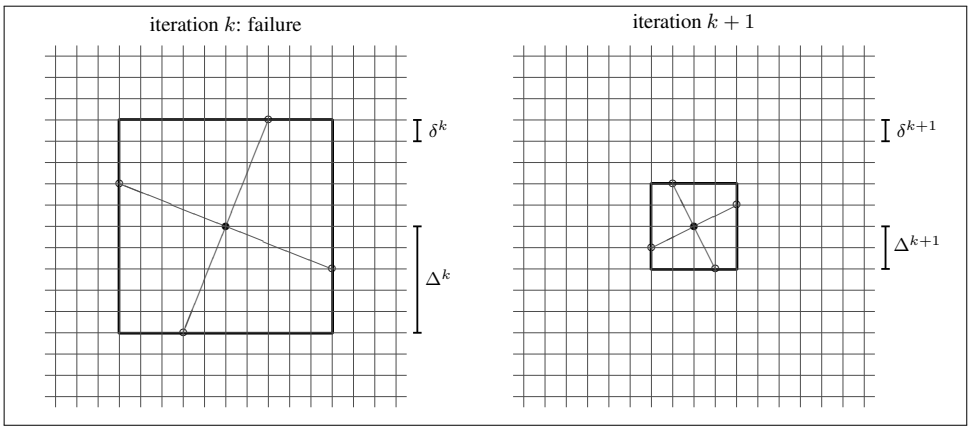


FIG. 2.1. *The central bullet represents the incumbent solution. All trial points are generated at the intersections of horizontal and vertical lines. The thick lines delimit the region containing the poll points.*

i.e., $D^k \in \mathbb{Z}^{n\times n}$ is a positive basis satisfying $\|d\|_\infty \leq \rho^k$ for each $d \in D^k$. The poll set is then redefined as

$$P^k = \{x^k + \delta^k d : d \in D^k\}. \tag{2.2}$$

With this construction, it can be seen that the distance from any poll point to the incumbent solution x^k satisfies

$$\|\delta^k d\|_\infty \leq \delta^k \times \rho^k = (10)^{b^k - |b^k|} \times a^k \times (10)^{|b^k|} = \Delta^k. \tag{2.3}$$

The left part of Figure 2.1 illustrates an example in which $\Delta^k = 5$, $\delta^k = 1$, and their ratio is $\rho^k = 5$. The matrix D^k is a positive basis whose entries are integers between -5 and 5. In the figure, the positive basis is

$$D^k = \left\{ \begin{bmatrix} 2 \\ 5 \end{bmatrix}, \begin{bmatrix} 5 \\ -2 \end{bmatrix}, \begin{bmatrix} -2 \\ -5 \end{bmatrix}, \begin{bmatrix} -5 \\ 2 \end{bmatrix} \right\}$$

and the four columns are depicted by the slanted lines. The dark circle corresponds to the incumbent solution x^k, and the four open circles represent the four poll points of P^k. The right part of the same figure illustrates the next iteration, once iteration k has failed to improve the incumbent solution. The poll size parameter is decreased to $\Delta^{k+1} = 2$ and the mesh size parameter remains the same: $\delta^{k+1} = 1$. The ratio ρ^k plays a fundamental role in the generation of the poll points. In the figure, the ratio may be interpreted as the infinity-norm radius of the poll set, i.e., the number of squares from the incumbent to the boundary of the poll region. The radius equals 5 in the left part of the figure, and is 2 in the right part.

Algorithm 3 summarizes the main steps of the rich polling algorithm. It differs from the CS algorithm with controlled decimals by the way in which the poll set P^k is constructed and by the fact that the step-size parameter is replaced by the mesh and poll size parameters.

Once again, if the sequence of trial points $\{x^k\}_{k=0}^\infty$ produced by the rich polling algorithm belongs to a bounded set, then there exists a subsequence of unsuccessful iteration indices for which the mesh and poll size parameters converge to zero, and

Algorithm 3. The rich polling algorithm.

Input: $x^0 \in \mathbb{R}^n$, the initial point
$\Delta^0 = 1 \times (10)^0 \in \mathcal{P}$, the initial poll size parameter

for iteration $k = 0, 1, 2, \ldots$ **do**
- set $\delta^k := (10)^{b^k - |b^k|}$, the mesh size parameter
- set $\rho^k := a^k \times 10^{|b^k|}$, the ratio parameter
- poll step: test $P^k := \{x^k + \delta^k d : d \in D^k\}$, where $D^k \in \mathbb{Z}^{n \times n}$ is a positive spanning set satisfying $\|d\|_\infty \leq \rho^k$
- **if** the poll step was successful at a trial point $t \in P^k$, **then**
  - set $x^{k+1} = t$ and $\Delta^{k+1} = \texttt{increase}(\Delta^k)$
- **else**
  - set $x^{k+1} = x^k$ and $\Delta^{k+1} = \texttt{decrease}(\Delta^k)$

for which the corresponding subsequence of incumbent solutions converges. Such a subsequence of incumbent solutions is called a *refining sequence*, and its limit point is called a *refined point*.

The next issue that needs to be discussed is ensuring that the union of normalized directions over all iterations grows dense in the unit sphere. There are different ways to achieve this, and we present a technique that constructs a maximal positive basis using the Householder transformation.

At iteration k of the algorithm, we first generate a vector $v \in \mathbb{R}^n$, normalized with respect to the Euclidean norm (details on how these normalized vectors are generated are found in [16, Page 347]). For conciseness, we will not attach the superscript k to the vector. Second, let $H = I - 2vv^\top$ be the orthogonal $n \times n$ Householder matrix and let $\{h_j : j \in N\}$ denote its columns. Finally, in order to make sure that each polling direction belongs to $\mathbb{Z}^n$ and has an infinity norm bounded above by the ratio $\rho^k \in \mathbb{N}$, we normalize the columns, multiply by the ratio, and round them as follows:

$$D^k = \left\{ \pm \text{round}\left(\rho^k \times \frac{h_j}{\|h_j\|_\infty}\right) : j \in N \right\} \subset \mathbb{Z}^n.$$

Here the function round : $\mathbb{R}^n \to \mathbb{Z}^n$ rounds all entries of the input vector in $\mathbb{R}^n$ upward to the nearest integer. This construction method ensures that D^k forms an integer maximal positive basis of $\mathbb{R}^n$ [16, Proposition 9.1]. Equation (2.3) is verified since every vector $d \in D^k$ satisfies $\|d\|_\infty = \rho^k = a^k \times (10)^{|b^k|}$, and consequently

$$\|\delta^k d\|_\infty = (10)^{b^k - |b^k|} \times (a^k \times (10)^{|b^k|}) = \Delta^k.$$

A minimal positive basis may be obtained by setting D^k to

$$\left\{ \text{round}\left(\rho^k \times \frac{h_j}{\|h_j\|_\infty}\right) : j \in N \right\} \cup \left\{ -\sum_{j \in N} \text{round}\left(\rho^k \times \frac{h_j}{\|h_j\|_\infty}\right) \right\}.$$

With this method of generating the polling directions, Algorithm 3 is entirely parameter-free. The only input the user needs to supply is the mechanism that computes the objective function value f (most often, this is a computer code), together with an initial point $x^0 \in \mathbb{R}^n$ where $f(x^0)$ may be computed.

3. The MADS algorithm. The previous section described polling algorithms in which the tentative points are always generated in the vicinity of the current incumbent solution x^k. In practice, however, it is usually preferable to explore the space of variables more globally. In order to do so, we redefine the discretization of the space of variables so that each variable is scaled individually. We define the mesh as

$$M^k := \{x + \text{diag}(\boldsymbol{\delta}^k)z : x \in V^k,\ z \in \mathbb{Z}^n\}, \tag{3.1}$$

where V^k is the set of all trial points at which the simulation was evaluated by the start of iteration k, and where $\boldsymbol{\delta}^k = (\delta_1^k, \delta_2^k, \ldots, \delta_n^k) \in \mathcal{P}^n$ is not a scalar but the mesh size vector composed of mesh size parameters. Going back to Figure 2.1, the mesh is represented by the intersection of all horizontal and vertical lines, not just those delimited by the dark lines.

At each iteration, MADS first enters what is called the search step. During that step, the simulation is launched at finitely many mesh points in the hope of improving the incumbent solution x^k. There are two possible outcomes of the search step: either it succeeds in improving the incumbent solution or it fails. When it is unsuccessful, the poll step is invoked, during which the simulation is launched at the trial points from the set

$$P^k := \{x^k + \text{diag}(\boldsymbol{\delta}^k)d : d \in D^k\} \subset M^k, \tag{3.2}$$

where $D^k \subset \mathbb{Z}^n$ is a positive spanning set satisfying $-\boldsymbol{\Delta}^k \leq \text{diag}(\boldsymbol{\delta}^k)d \leq \boldsymbol{\Delta}^k$ for every $d \in D^k$, and where $\boldsymbol{\Delta}^k = (\Delta_1^k, \Delta_2^k, \ldots, \Delta_n^k) \geq \boldsymbol{\delta}^k$ is the poll size vector in $\mathcal{P}^n$. In constrained optimization using the progressive barrier [10], the incumbent solution x^k is either the best feasible solution found so far, or the infeasible one unfiltered by the progressive barrier with the best objective function value.

Equation (3.2) generalizes (2.2) by allowing individual scaling for each variable. As for the search, the same two outcomes are possible for the poll. The entire iteration is said to be successful if a new incumbent is found during either the search step or the poll step; otherwise it is said to be unsuccessful.

The reason for separating each iteration into two steps is to allow the use of efficient strategies to explore the space of variables. For example, a surrogate of the true optimization problem may be used to generate promising trial points. Surrogates may be static low-fidelity models or dynamically constructed with techniques such as Gaussian processes, polynomial interpolation, or radial basis functions (see [13] for a survey on DFO model-based methods). The search step allows MADS to be coupled with other optimization techniques. Strategies such as Latin hypercube sampling, variable neighborhood searches [6], or Nelder–Mead searches [19] may be used in an attempt to escape local solutions or to accelerate convergence.

In addition, to ensure convergence, the poll step can be viewed as a safeguard to ensure that the mesh and poll size parameters are not reduced too aggressively. Before declaring an iteration unsuccessful, the true simulation must be launched at each of the trial points in the poll set P^k, formed by a positive spanning set of directions. Concretely, this implies that the central point x^k satisfies some kind of crude mesh optimality conditions: no better solution was found in its discrete neighborhood P^k parameterized by $\boldsymbol{\Delta}^k$. At the end of an unsuccessful iteration, the mesh and poll size parameters are all reduced. If the iteration is successful, they are either increased or kept constant.

3.1. Initial scaling of variables. The first step of MADS consists in estimating the approximate magnitude of each poll size parameter. This is done by taking into

account the starting point $x^0 \in \mathbb{R}^n$ as well as the lower and upper bounds vectors ℓ and u (which are possibly infinite). For each $i \in N$, set $\alpha_i^0 > 0$ to the value

(3.3)
$$\alpha_i^0 = \begin{cases} \frac{u_i - \ell_i}{10} & \text{if } x_i^0 \text{ is bounded by the finite values } \ell_i < u_i, \\ \frac{|x_i^0 - w_i|}{10} & \text{if } x_i^0 \text{ has only one finite bound } w_i \text{ which differs from } x_i^0, \\ \frac{|x_i^0|}{10} & \text{if } x_i^0 \text{ has only one finite bound which equals } x_i^0 \text{ and } x_i^0 \neq 0, \\ \frac{|x_i^0|}{10} & \text{if no bound is provided for } x_i^0 \text{ but } x_i^0 \neq 0, \\ 1 & \text{otherwise.} \end{cases}$$

This way of computing the magnitudes α_i^0 was proposed in [16]. The first line of (3.3) has the most information on the domain (it has two distinct and finite bounds) and information deteriorates as one goes down the list. The last line includes the situation where x_i is unbounded and the initial point is the origin: this means that no information about the magnitude of the variable is available.

In the previous implementations of MADS [16], this value α_i^0 served as the initial poll size parameter for the ith variable. In our context, we simply set Δ_i^0 to the value $a_i^0 \times (10)^{b_i^0}$ in $\mathcal{P}$ that is the closest to α_i^0 by using a simple rounding operation. The initial poll size vector is $\boldsymbol{\Delta}^0 \in \mathcal{P}^n$.

Iteration k of the MADS algorithm is initiated with the poll size vector $\boldsymbol{\Delta}^k \in \mathcal{P}^n$. For each $i \in N$, the mesh size parameter δ_i^k is obtained from the poll size parameter $\Delta_i^k = a_i^k \times (10)^{b_i^k}$ as follows:

$$\delta_i^k := (10)^{b_i^k - |b_i^k - b_i^0|}. \tag{3.4}$$

Equation (3.4) generalizes (2.1) by taking into account the initial scaling imposed on the variables. The mesh size parameter δ_i^k equals $(10)^{b_i^0}$ when b_i^k is greater than or equal to b_i^0, and equals $(10)^{2b_i^k - b_i^0}$ otherwise. This way of setting the mesh size vector can be written compactly in vector notation as $\boldsymbol{\delta}^k := (10)^{\mathbf{b}^k - |\mathbf{b}^k - \mathbf{b}^0|}$.

Each trial point generated during the iteration lies on the mesh M^k from (3.1). Notice that, by construction, the poll size parameter Δ_i^k is an integer multiple of the mesh size parameter δ_i^k. The ratio of these two parameters is defined by

$$\rho_i^k := \frac{\Delta_i^k}{\delta_i^k} = a_i^k \times 10^{|b_i^k - b_i^0|} \in \mathbb{N}.$$

An equivalent way of writing this last equation is to define the ratio vector as $\boldsymbol{\rho}^k = (\rho_1^k, \rho_2^k, \ldots, \rho_n^k) := \operatorname{diag}(\boldsymbol{\delta}^k)^{-1} \boldsymbol{\Delta}^k$.

Finally, the set of directions constituting D^k can be constructed as follows. Set

$$D^k = \left\{ \pm \operatorname{round}\left(\operatorname{diag}(\boldsymbol{\rho}^k) \times \frac{h_j}{\|h_j\|_\infty} \right) : j \in N \right\} \subset \mathbb{Z}^n,$$

where $\{h_j : j \in N\}$ contains the columns of the orthogonal $n \times n$ Householder matrix constructed from a unit vector $v \in \mathbb{R}^n$.

MADS, with initial scaling and controlled decimals, is summarized in Algorithm 4.

3.2. Dynamic scaling of variables. In Algorithm 4, once the initial scaling vector $\boldsymbol{\Delta}^0$ has been determined, the relative scaling between variables does not change much. This is because at successful iterations all entries of $\boldsymbol{\Delta}^k$ are increased, and at unsuccessful ones they are all decreased.

Algorithm 4. MADS with initial scaling.

Input: $x^0 \in \mathbb{R}^n$, the initial point
$\boldsymbol{\Delta}^0$, the initial poll size vector where Δ_i^0 is the value $a_i^0 \times (10)^{b_i^0}$ in $\mathcal{P}$ which is the closest to α_i^0 from (3.3)

for iteration $k = 0, 1, 2, \ldots$ **do**
- set $\boldsymbol{\delta}^k := (10)^{\mathbf{b}^k - |\mathbf{b}^k - \mathbf{b}^0|} \in \mathcal{P}^n$, the mesh size vector
- set $\boldsymbol{\rho}^k := \text{diag}(\boldsymbol{\delta}^k)^{-1}\boldsymbol{\Delta}^k \in \mathbb{N}^n$, the ratio vector
- search step: test finitely many trial points on the mesh M^k
- poll step: test $P^k := \{x^k + \text{diag}(\boldsymbol{\delta}^k)d : d \in D^k\} \subset M^k$, where $D^k \in \mathbb{Z}^{n\times n}$ is a positive spanning set satisfying $-\boldsymbol{\rho}^k \le d \le \boldsymbol{\rho}^k$ for every $d \in D^k$
- **if** the search or poll step was successful at a trial point $t \in M^k$, **then**
  - set $x^{k+1} = t$ and $\boldsymbol{\Delta}^{k+1} = \texttt{increase}(\boldsymbol{\Delta}^k)$ $\quad (*)$
- **else**
  - set $x^{k+1} = x^k$ and $\boldsymbol{\Delta}^{k+1} = \texttt{decrease}(\boldsymbol{\Delta}^k)$

(the star $(*)$ is referred to in section 3.2)

In previous work [16], we proposed increasing the mesh size parameters at successful iterations only for the variables whose values changed significantly. An additional mechanism needs to be introduced to make sure that if one of these parameters converges to zero, then all others also converge to zero.

Recall that, for the trial points generated during the poll step, the value $|d_i|$ is bounded above by the ratio ρ_i^k. We will consider a modification to be important enough to justify increasing the corresponding mesh size parameter if $|d_i|/\rho_i^k > a_t$, where $a_t > 0$ is the anisotropy trigger parameter to be selected. In addition, recall that we need to control the rate at which the mesh size parameters converge to zero. This is done by imposing a condition on the ratio parameter. More precisely, the mesh size parameter Δ_i^{k+1} is also increased when the mesh size parameter δ_i^k is finer than the original one, δ_i^0, and the ratio ρ_i^k exceeds the square of another ratio parameter ρ_l^k for some index $l \in N$. Formally, we replace the line marked $(*)$ in Algorithm 4 by the following: set $x^{k+1} = t$ and

$$\Delta_i^{k+1} = \begin{cases} \texttt{increase}(\Delta_i^k) & \begin{cases} \text{if} \quad |d_i|/\rho_i^k > a_t, \text{ or if} \\ \quad\ \ \delta_i^k < \delta_i^0 \text{ and } \rho_i^k > (\rho_l^k)^2 \text{ for some } l \in N, \end{cases} \\ \Delta_i^k & \text{otherwise,} \end{cases} \tag{3.5}$$

for every index $i \in N$, and where $d \in \mathbb{Z}^n$ is the direction that led to success: $x^{k+1} = x^k + \text{diag}(\boldsymbol{\delta}^k)d$.

3.3. MADS for granular variables. The last algorithmic contribution of this work concerns situations in which a minimal granularity is assigned to some variables. The most natural example is when some variables are integers. Another situation is when only two decimals are desired, or when a variable needs be rounded to the nearest multiple of 0.005, for example. The granularity of these cases is 1, 0.01, and 0.005, respectively. Of course, each of these situations could be handled inside the blackbox by making the variable integer and then multiplying it by the corresponding granularity. But this requires altering the blackbox, the bounds, and the starting point, and it masks the actual numerical values from the user. Instead, we propose to handle the granularities within the MADS algorithm.

Recall that $\mathcal{G}$ is the set of indices of variables with an explicit minimal granularity. The value $\delta_i^{\min}$ represents the granularity and the corresponding poll parameters satisfy

$$\Delta_i^k \in \mathcal{P}(\delta_i^{\min}) = \{a \times (10)^b \times \delta_i^{\min} : a \in \{1, 2, 5\},\ b \in \mathbb{N}\} \text{ for } i \in \mathcal{G}.$$

Notice that the value of b in this set is required to be a nonnegative integer. This means that Δ_i^k will necessarily be an integer multiple of the granularity $\delta_i^{\min}$, as required. The smallest value that Δ_i^k may take is exactly $\delta_i^{\min}$, and occurs when $a = 1$ and $b = 0$.

The poll size parameters corresponding to the continuous variables are handled as follows:

$$\Delta_i^k \in \mathcal{P} = \{a \times (10)^b : a \in \{1, 2, 5\},\ b \in \mathbb{Z}\} \text{ for } i \in N \setminus \mathcal{G}.$$

The mesh size parameters may take values arbitrarily close to zero. For convenience, we set $\delta_i^{\min} := 0$ for every $i \in N \setminus \mathcal{G}$.

On unsuccessful iterations, the operator that decreases the mesh size is redefined as follows: set $x^{k+1} = x^k$ and

$$\boldsymbol{\Delta}_i^{k+1} = \begin{cases} \texttt{decrease}(\Delta_i^k) & \text{if } i \in N \setminus \mathcal{G}, \\ \delta_i^{\min} \times \max\left\{1,\ \texttt{decrease}\left(\frac{\Delta_i^k}{\delta_i^{\min}}\right)\right\} & \text{if } i \in \mathcal{G}. \end{cases} \tag{3.6}$$

This implies that the variables that do not have a minimal granularity are handled as previously, and those that have a minimal granularity are reduced, up to the minimal value $\delta_i^{\min}$. The corresponding mesh size parameter is obtained from the poll size parameter as follows:

$$\delta_i^k := \begin{cases} (10)^{b_i^k - |b_i^k - b_i^0|} & \text{if } i \in N \setminus \mathcal{G}, \\ \delta_i^{\min} \times \max\{1, (10)^{b_i^k - |b_i^k - b_i^0|}\} & \text{if } i \in \mathcal{G}. \end{cases} \tag{3.7}$$

This last equation ensures that the mesh size parameters are bounded below by their respective minimal granularity. The ratio of poll and mesh sizes for $i \in \mathcal{G}$ is given by

$$\rho_i^k := a_i^k \times \min\{10^{b_i^k}, 10^{|b_i^k - b_i^0|}\} \in \mathbb{N}.$$

On successful iterations, the mesh size vector is updated as in the previous subsection, except that (3.5) becomes

$$\Delta_i^{k+1} = \begin{cases} \texttt{increase}(\Delta_i^k) & \begin{cases} \text{if} & |d_i|/\rho_i^k > a_t, \text{ or if} \\ & \delta_i^k < \delta_i^0 \text{ and } \rho_i^k > (\rho_l^k)^2 \text{ for some } l \in N \setminus \mathcal{G}, \end{cases} \\ \Delta_i^k & \text{otherwise} \end{cases} \tag{3.8}$$

for every index $i \in N$. The difference from (3.5) is that the comparison on the ratio parameters $\rho_i^k > (\rho_l^k)^2$ is only made with the continuous variables $l \in N \setminus \mathcal{G}$, because the variables whose indices are in $\mathcal{G}$ are granular and the mesh size parameter associated with a granular variable cannot converge to zero. Algorithm 5 contains all the functionalities introduced in the present paper.

Algorithm 5. MADS with minimal granularity and controlled decimals.

Input: $\mathcal{G} \subseteq N$, the indices of variables with minimal granularity
$x^0 \in \mathbb{R}^n$, the initial point, with $\frac{x_i^0}{\delta_i^{\min}} \in \mathbb{Z}$ for each $i \in \mathcal{G}$
$\boldsymbol{\Delta}^0$, the initial poll size vector, where Δ_i^0 is the value $a_i^0 \times (10)^{b_i^0}$ in $\mathcal{P}$ which is the closest to α_i^0 from (3.3)

for iteration $k = 0, 1, 2, \ldots$ **do**
- let $\boldsymbol{\delta}^k$ be the mesh size vector from (3.7)
- set $\boldsymbol{\rho}^k := \text{diag}(\boldsymbol{\delta}^k)^{-1}\boldsymbol{\Delta}^k \in \mathbb{N}^n$, the ratio vector
- search step: test finitely many trial points on the mesh M^k (3.1)
- poll step: test $P^k := \{x^k + \text{diag}(\boldsymbol{\delta}^k)d : d \in D^k\} \subset M^k$, where $D^k \in \mathbb{Z}^{n\times n}$ is a positive spanning set satisfying $-\boldsymbol{\rho}^k \le d \le \boldsymbol{\rho}^k$ for every $d \in D^k$
- **if** the search or poll step was successful at a trial point $t \in M^k$, **then**
  - set $x^{k+1} = t$ and $\boldsymbol{\Delta}^{k+1}$ as in (3.8)
- **else**
  - set $x^{k+1} = x^k$ and $\boldsymbol{\Delta}^{k+1}$ as in (3.6)

3.4. Convergence of MADS with minimal granularity and controlled decimals. The algorithmic modifications of Algorithm 5 versus the MADS algorithm from [16] correspond to the way in which the poll and mesh size parameters are updated from one iteration to another. The update rules in [16] are similar to those proposed for the generalized pattern search (GPS) algorithm of Torczon [66], in which both the mesh and poll size parameters are identical (in fact, there is a single parameter for both roles). In MADS, the mesh size parameter was less than or equal to the poll size, and typically $\delta^k = \min\{\Delta^k, (\Delta^k)^2\}$. In both algorithms, these parameters were updated by multiplying them by an integer power of a rational number τ. In [16], the parameters δ^k and Δ^k are replaced by the vectors $\boldsymbol{\delta}^k$ and $\boldsymbol{\Delta}^k$, as in the present work, but again each component of these vectors is an integer power of τ. The key element in the convergence analysis is to show that, at any iteration number p, all tentative trial points generated by these algorithms belong to a fine mesh. This was first shown in [66, Theorem 3.2].[3] The proof of this result is far from trivial, as it uses the hypothesis that the number τ is rational. The necessity of this requirement is illustrated in [5].

The equivalent of Torczon's key theorem is written in the present work as follows.

THEOREM 3.1. *The iterate x^p generated by the* MADS *algorithm with minimal granularity and controlled decimals (Algorithm* 5*) can be expressed in the form*

$$x^p = x^0 + \delta^{\min} \sum_{k=0}^{p-1} w_k,$$

where

- $x^0 \in V^0$ *is an initial point provided by the user,*
- $\delta^{\min}$ *is an integer power of* 10 *that depends on* p*,*
- $w^k \in \mathbb{Z}^n$ *for each* $k = 0, 1, \ldots, p-1$*.*

[3] This paper won the 1999 SIAM outstanding paper prize.

Proof. At iteration p, the point x^p belongs to the mesh M^k and therefore may be decomposed as $x^p = x^{p-1} + \text{diag}(\boldsymbol{\delta}^{p-1})z^{p-1}$ for some $z^{p-1} \in \mathbb{Z}^n$. Applying the same decomposition to $x^{p-1}, x^{p-2}, \ldots$ yields

$$x^p = x^0 + \sum_{k=0}^{p-1} \text{diag}(\boldsymbol{\delta}^k) z^k$$

for some vectors $z^k \in \mathbb{Z}^n$.

Each component of the vector $\boldsymbol{\delta}^k$ is an integer power of 10. By defining

$$\delta^{\min} = \min\{\delta_i^k : k \in \{0, 1, \ldots, p-1\},\ i \in N\} > 0$$

and setting $w^k = \frac{1}{\delta^{\min}} \text{diag}(\boldsymbol{\delta}^k) z^k$ we get

$$x^p = x^0 + \delta^{\min} \sum_{k=0}^{p-1} w^k,$$

and $w_i^k = (\delta_i^k / \delta^{\min}) z_i^k$ is integer because the value $\delta_i^k / \delta^{\min}$ is a nonnegative integer power of 10 and because $z_i^k \in \mathbb{Z}$. ∎

The above proof is extremely simple, because the components of the mesh size vectors are always an integer power of 10. The theorem implies that at any given iteration all trial points are generated on a mesh scaled by $\delta^{\min}$ and translated by the initial point x^0. The immediate consequence of this result is that if all trial points produced by the algorithm belong to a bounded set, then, for any variable whose index belongs to $N \setminus \mathcal{G}$, the limit inferior of the mesh size parameter is zero.

It follows that the main convergence results remain valid, and unaltered in essence. They are similar to the results shown in [2], in which those with respect to the continuous variables are restricted to the subspace in which the granular variables are fixed. The first result does not involve any assumptions on the functions, and is called the zeroth-order result.

THEOREM 3.2 (zeroth-order result). *Let $\{x^k\}_{k \in K}$ be a refining subsequence converging to the refined point x^*. Then x^* is the limit of a sequence of mesh local optimizers $\{x^k\}_{k \in K}$ on meshes that get infinitely fine with respect to the continuous variables and on the finest allowable mesh with respect to the granular variables:*

$$\begin{cases} \delta_i^k \to 0 & \text{if } i \in N \setminus \mathcal{G}, \\ \delta_i^k = \delta_i^{\min} & \text{if } i \in \mathcal{G}. \end{cases}$$

The above result may be strengthened by assuming Lipschitz continuity with respect to the continuous variables. For a given x^*, we define $f_{x^*}(y) = f(x)$, where $x_i = x_i^*$ for every $i \in \mathcal{G}$ and $x_i = y_i$ for every $i \in N \setminus \mathcal{G}$. With this notation f_{x^*} is a function of $|N \setminus \mathcal{G}|$ variables, defined on the space $S_{x^*} = \{x \in \mathbb{R}^n : x_i = x_i^*\ \forall i \in \mathcal{G}\}$ of the same dimension.

THEOREM 3.3 (unconstrained optimization). *Let $\{x^k\}_{k \in K}$ be a refining subsequence converging to the refined point $x^* \in \mathbb{R}^n$, and let $\mathcal{D} = \{\frac{d}{\|d\|} : d \in D^k,\ k \in K\}$ be the set of normalized polling directions used. If f_{x^*} is Lipschitz with respect to the continuous variables near x^*, and if the set of directions $\mathcal{D}$ is dense in the unit sphere in the subspace of continuous variables S_{x^*}, then*

$$f_{x^*}^\circ(x^*; d) \geq 0 \quad \forall d \in S_{x^*}.$$

The next result involves constraints. Let us define $\Omega_{x^*} = \{x \in \Omega : x_i = x_i^* \ \forall i \in \mathcal{G}\}$ to be the restriction of Ω to the subspace S_{x^*} in which the granular variables are fixed. As in [2], we consider the hypertangent cone $T^H_{\Omega_{x^*}}$ to the continuous variables. The term *refining sequence* still refers to unsuccessful iterations, but is extended to take constraints into account, as in [10].

THEOREM 3.4 (constrained optimization). *Let $\{x^k\}_{k\in K}$ be a refining subsequence converging to the refined point $x^* \in \Omega$, and let $\mathcal{D} = \{\frac{d}{\|d\|} : d \in D^k,\ k \in K\}$ be the set of normalized polling directions used. If f_{x^*} is Lipschitz near x^*, and if the set of directions $\mathcal{D}$ is dense in the unit sphere in the subspace of continuous variables S_{x^*}, then*

$$f^\circ_{x^*}(x^*; d) \geq 0 \quad \forall d \in T^H_{\Omega_{x^*}}(x^*).$$

Theorem 3.4 is the fundamental result of the convergence analysis. A more elaborate hierarchical analysis may be derived as in [9] by studying regular or strictly differentiable functions, as well as more general types of tangent cones.

4. Computational results. Computational results for different algorithms are obtained on a series of test cases relying on the following definitions.

DEFINITION 4.1. *For algorithms requiring an initial point x_0 as input, a* computational problem *is characterized by a unique expression of the objective and constraint functions, the set $\mathcal{X}$ used for the definition of problem* (1.1)*, and a single initial point.*

Hence, it is possible to increase the number of computational problems by changing the initial points. This can be an option to obtain more computational results when the number of tests cases is limited (but is actually somewhat restrictive in scope).

DEFINITION 4.2. *Replicating optimization runs on the same computational problem constitutes different* run instances.

For a given computational problem, different run instances of a given algorithm can be obtained by varying the pseudorandom-number-generator seed influencing the behavior of this algorithm. This type of variation, when available, can increase the number of computational results to better assess the performance of algorithms influenced by a seed (but again is actually somewhat restrictive in scope).

In this work, the relative performance of algorithms is assessed by data profiles [54], which requires one to define a convergence test for a given computational problem.

Let x^b denote the best feasible point obtained by all tested algorithms on all run instances and let x_e be the best feasible iterate obtained after e evaluations of a run instance performed by a given algorithm. The computational problem is said to be successfully solved by the algorithm within the convergence tolerance $\tau > 0$ when

$$\bar{f}_{\text{fea}} - f(x_e) \geq (1 - \tau)(\bar{f}_{\text{fea}} - f(x^b)),$$

where the reference value $\bar{f}_{\text{fea}}$ is obtained by taking the average of the first feasible objective function values over all run instances of that computational problem for all algorithms. If no feasible iterate is obtained, the convergence test fails. For computational problems without constraints, $\bar{f}_{\text{fea}} = f(x_0)$, where x_0 is the initial point.

The data profiles show the proportion of computational problems solved by all run instances of a given algorithm, within a given tolerance τ, versus the number of groups of $(n+1)$ evaluations.[4]

This section is organized as follows: after describing the different solvers in section 4.1, we first compare two variants of MADS on purely continuous problems (4.2) and on discrete problems (4.3). We then compare MADS with other solvers on bound-constrained discrete problems in section 4.4, and conclude in section 4.5 with the specific application outlined in the introduction, which involves granular variables.

4.1. Algorithms and solvers. Computational results using the MADS algorithm are generated using version 3.9.0 of the NOMAD [40] software package, freely available at www.gerad.ca/nomad.

The MADS algorithm with $n+1$ poll directions [14] with or without the use of quadratic models, as in [25], is considered.

Two variants are used. First, "GMesh," corresponds to MADS with the new granular mesh. Second, MADS in which the mesh size parameter is updated by multiplying by an integer power of a fixed scalar is called "XMesh." The latter corresponds to the anisotropic MADS $(n+1)$ variant (each variable is dynamically scaled) of [16], where the mesh size is multiplied or divided by an integer power of a rational number τ, and where rounding is performed to obtain integer values when needed.

In section 4.4, NOMAD is compared with the three solvers mentioned in the introduction: first, DFL [60]; then its DFL gen version for constrained problems; then the MATLAB implementations of the MISO and BFO solvers for bound-constrained optimization. Deployment details for these solvers are given in section 4.4.

4.2. GMesh and XMesh on continuous analytical problems. Preliminary experiments on 87 continuous analytical computational problems from the optimization literature are conducted to set the default values of the anisotropy trigger parameter a_t (see (3.8)). The characteristics and sources of these problems are summarized in Table 4.1, ignoring the "Mod. Int." column. The number of variables ranges from $n = 2$ to 20, with 19 problems having constraints ($m > 0$) other than bound constraints.

In all the figures and tables in the remainder of this section, the new algorithm introduced in this paper is called GMesh and the previous MADS algorithm is called XMesh. We use 10 run instances for each problem, with different random seeds. Results with different values of the parameter a_t are presented in Figure 4.1, with precision $\tau \in \{10^{-3}, 10^{-5}, 10^{-7}\}$, with and without the use of quadratic models. Unsurprisingly, the algorithmic performance is improved with the use of models on these analytical test problems [25].

The data profiles reveal that the effect of parameter a_t is more important when quadratic models are used. Without the models, the new method GMesh outperforms XMesh, and the value $a_t = 2$ is slightly preferable to the other values tested. When models are enabled, only the variants with a small value of a_t outperform XMesh. For the remaining tests, $a_t = 0.1$ is considered.

4.3. GMesh and XMesh on problems with discrete variables. Among the 87 continuous analytical computational problems listed in the previous subsection, 51 are modified as mixed integer variable problems (the column "Mod. Int." in Table 4.1 indicates problems that can be modified to include integers). The modification

[4] $n+1$ is the number of evaluations required to construct a simplex gradient or a linear interpolant in $\mathbb{R}^n$.

TABLE 4.1
Description of the set of 87 *continuous analytical problems from the literature.*

#	Name	Source	n	m	Bnds	Mod. Int.
1	ARWHEAD10	[31]	10	0	no	yes
2	ARWHEAD20	[31]	20	0	no	yes
3	BARD	[53]	3	0	no	yes
4	BDQRTIC10	[31]	10	0	no	yes
5	BDQRTIC20	[31]	20	0	no	yes
6	BEALE	[53]	2	0	no	yes
7	BIGGS	[31]	6	0	no	yes
8	BOX	[53]	3	0	no	yes
9	BRANIN	[33]	2	0	yes	yes
10	BROWNAL5	[31]	5	0	no	yes
11	BROWNAL7	[31]	7	0	no	yes
12	BROWNAL10	[31]	10	0	no	yes
13	BROWNAL20	[31]	20	0	no	yes
14	BROWNDENNIS	[53]	4	0	no	yes
15	BROWN_BS	[53]	2	0	no	yes
16	CHENWANG_F2	[23]	8	6	yes	no
17	CHENWANG_F3	[23]	10	8	yes	no
18	CRESCENT	[10]	10	2	no	yes
19	DISK	[10]	10	1	no	yes
20	ELATTAR	[49]	6	0	no	yes
21	EVD61	[49]	6	0	no	yes
22	FILTER	[49]	9	0	no	yes
23	FREUDENSTEINROTH	[53]	2	0	no	yes
24	GAUSSIAN	[53]	3	0	no	yes
25	G210	[11]	10	2	yes	yes
26	G220	[11]	20	2	yes	yes
27	GRIEWANK	[33]	10	0	yes	yes
28	GULFRD	[53]	3	0	no	yes
29	HELICALVALLEY	[53]	3	0	no	yes
30	HS19	[35]	2	2	yes	no
31	HS78	[49]	5	0	no	no
32	HS83	[35]	5	6	yes	no
33	HS114	[49]	9	6	yes	no
34	JENNRICHSAMPSON	[53]	2	0	no	no
35	KOWALIKOSBORNE	[53]	4	0	no	no
36	MAD6	[49]	5	7	no	yes
37	MCKINNON	[50]	2	0	no	no
38	MDO	[68]	10	10	yes	yes
39	MEZMONTES	[51]	2	2	yes	no
40	MEYER	[53]	3	0	no	no
41	OPTENG_RBF	[37]	3	4	yes	no
42	OSBORNE1	[53]	5	0	no	yes
43	OSBORNE2	[49]	11	0	no	yes
44	PBC1	[49]	5	0	no	yes
45	PENALTY1_4	[31]	4	0	no	no
46	PENALTY1_10	[31]	10	0	no	no
47	PENALTY1_20	[31]	20	0	no	no
48	PENALTY2_4	[31]	4	0	no	no
49	PENALTY2_10	[31]	10	0	no	no
50	PENALTY2_20	[31]	20	0	no	no
51	PENTAGON	[49]	6	15	no	yes
52	PIGACHE	[59]	4	11	yes	no
53	POLAK2	[49]	10	0	no	yes
54	POWELL_BS	[53]	2	0	no	no
55	POWELLSG4	[31]	4	0	no	yes
56	POWELLSG8	[31]	8	0	no	yes
57	POWELLSG12	[31]	12	0	no	yes
58	POWELLSG20	[31]	20	0	no	yes
59	RADAR	[52]	7	0	yes	no
60	RANA	[36]	2	0	yes	yes
61	RASTRIGIN	[33]	2	0	yes	no
62	ROSENBROCK	[53]	2	0	yes	no
63	SHOR	[49]	5	0	no	yes
64	SNAKE	[10]	2	2	no	yes
65	SPRING	[62]	3	4	yes	no
66	SROSENBR6	[31]	6	0	no	yes
67	SROSENBR8	[31]	8	0	no	yes
68	SROSENBR10	[31]	10	0	no	yes
69	SROSENBR20	[31]	20	0	no	yes
70	TAOWANG_F2	[65]	7	4	yes	no
71	TREFETHEN	[36]	2	0	yes	no
72	TRIDIA10	[31]	10	0	no	no
73	TRIDIA20	[31]	20	0	no	no
74	TRIGONOMETRIC	[53]	10	0	no	no
75	VARDIM8	[31]	8	0	no	no
76	VARDIM10	[31]	10	0	no	no
77	VARDIM20	[31]	20	0	no	no
78	WANGWANG_F3	[72]	2	0	yes	no
79	WANGWANG_F5	[72]	2	0	yes	no
80	WATSON9	[53]	9	0	no	yes
81	WATSON12	[53]	12	0	yes	no
82	WONG1	[49]	7	0	no	yes
83	WONG2	[49]	10	0	no	yes
84	WOODS4	[31]	4	0	no	yes
85	WOODS12	[31]	12	0	no	yes
86	WOODS20	[31]	20	0	no	yes
87	ZHAOWANG_F5	[75]	13	9	yes	no

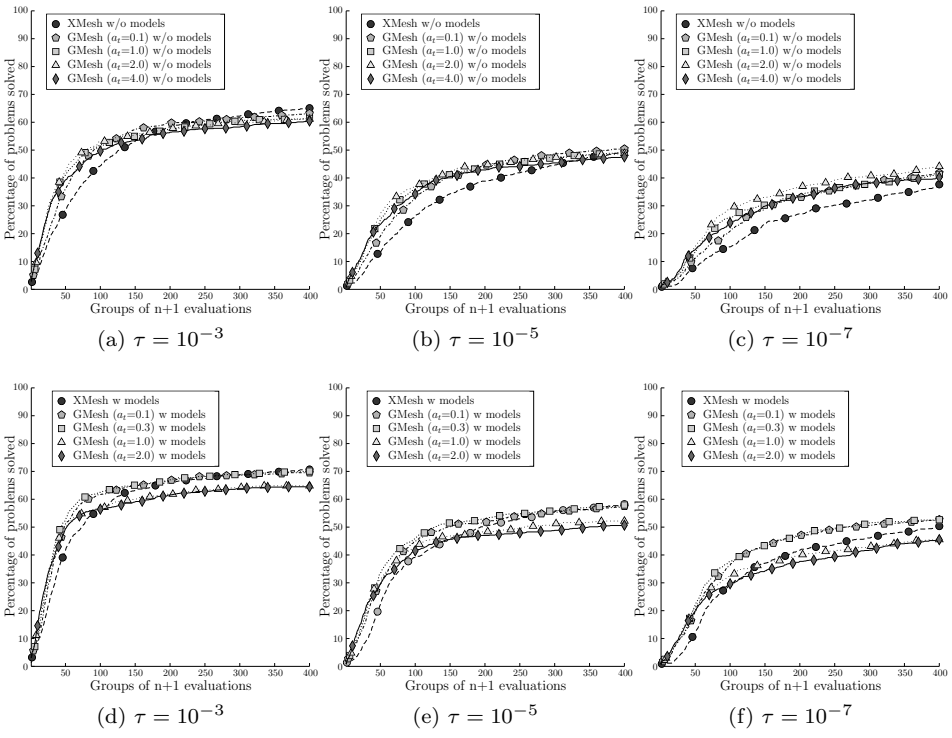
consists in changing the type of even-numbered index variables from real to integer whenever the corresponding initial point value and bounds are integers.

In addition, we also consider problems with a mix of continuous, integer, and binary variables from the literature. The MISO analytical problems from [55] have no constraints other than bounds. A subset of the SO-I analytical problems from [57] and the SO-MI set from [56] have constraints other than bounds. Duplicated problems in MISO, SO-I, and SO-MI are not considered. Table 4.2 summarizes the set of MISO, SO-I, and SO-MI discrete problems considered in this work. In total, we have a set of 94 discrete problems in order to compare the performance of GMesh and XMesh on 10 run instances per algorithm and problem.

Figure 4.2 shows the data profiles comparing the two MADS variants. The new version GMesh always dominates the previous version XMesh, and this domination amplifies when the requested precision grows. The domination is explained by the fact that the anisotropy trigger a_t was tuned for the GMesh version.

4.4. Comparison with other solvers on discrete problems. A subset of MISO, SO-I, and SO-MI problems from Table 4.2 is also used to compare the performance of GMesh and XMesh with the DFL, MISO, and BFO solvers.

The MISO code cannot directly solve problems with only integer or binary variables and problems with constraints other than bounds. The BFO solver considers only bound-constrained problems with mixed real and integer variables. Hence, only 12 problems on 10 initial points from the MISO and SO-MI sets are available.



(a) $\tau = 10^{-3}$ (b) $\tau = 10^{-5}$ (c) $\tau = 10^{-7}$

(d) $\tau = 10^{-3}$ (e) $\tau = 10^{-5}$ (f) $\tau = 10^{-7}$

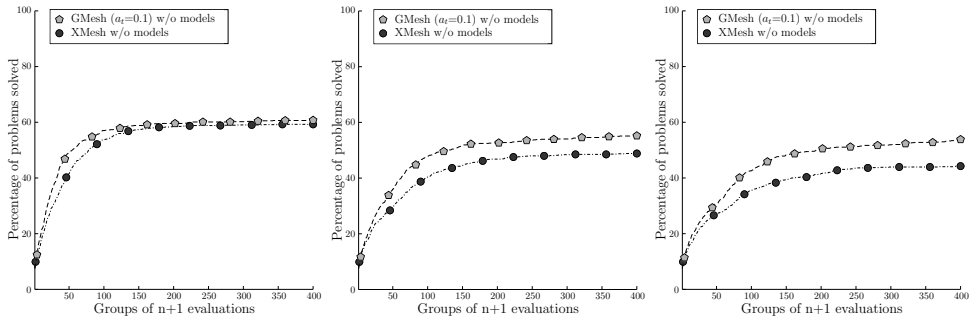
FIG. 4.1. *Data profiles obtained with convergence tolerance* $\tau = 10^{-3}$, $\tau = 10^{-5}$, *and* $\tau = 10^{-7}$ *on* 10 *run instances of* 87 *continuous analytical problems.*

TABLE 4.2
Description of selected discrete problems from [55, 56, 57].

Name	Source	n	Cont.	Int.	Bin.	m	Bnds
MISO1	[55]	12	7	5	0	0	yes
MISO2	[55]	8	4	4	0	0	yes
MISO4	[55]	5	2	3	0	0	yes
MISO6	[55]	15	9	6	0	0	yes
MISO7	[55]	2	1	1	0	0	yes
MISO8	[55]	15	5	10	0	0	yes
MISO9	[55]	3	2	1	0	0	yes
SO-I1	[57]	2	0	2	0	2	yes
SO-I2	[57]	5	0	5	0	0	yes
SO-I3	[57]	5	0	3	2	5	yes
SO-I4	[57]	5	0	5	0	6	yes
SO-I5	[57]	7	0	7	0	4	yes
SO-I6	[57]	13	0	4	9	9	yes
SO-I7	[57]	10	0	10	0	0	yes
SO-I8	[57]	16	0	0	16	8	no
SO-I9	[57]	12	0	12	0	0	yes
SO-I10	[57]	30	0	30	0	0	yes
SO-I11	[57]	25	0	25	0	2	yes
SO-I12	[57]	20	0	0	25	0	yes
SO-I13	[57]	10	0	10	0	0	yes
SO-I14	[57]	13	0	13	0	9	yes
SO-I15	[57]	12	0	12	0	0	yes

Name	Source	n	Cont.	Int.	Bin.	m	Bnds
SO-I16	[57]	8	0	8	0	0	yes
SO-I17	[57]	5	0	3	2	3	yes
SO-MI1	[56]	11	7	0	4	7	yes
SO-MI2	[56]	8	4	4	0	0	yes
SO-MI3	[56]	5	3	2	0	5	yes
SO-MI4	[56]	3	2	0	1	3	yes
SO-MI5	[56]	25	19	6	0	3	yes
SO-MI6	[56]	5	3	2	0	6	yes
SO-MI7	[56]	2	1	1	0	2	yes
SO-MI8	[56]	7	4	3	0	4	yes
SO-MI9	[56]	5	2	3	0	3	yes
SO-MI10	[56]	5	2	3	0	0	yes
SO-MI11	[56]	10	5	5	0	0	yes
SO-MI12	[56]	10	5	5	0	0	yes
SO-MI13	[56]	12	7	5	0	0	yes
SO-MI14	[56]	12	7	5	0	0	yes
SO-MI15	[56]	30	20	10	0	0	yes
SO-MI16	[56]	11	7	0	4	13	yes
SO-MI19	[56]	10	5	5	0	3	yes
SO-MI20	[56]	8	4	4	0	3	yes
SO-MI21	[56]	10	5	5	0	3	yes

For both DFL and MISO solvers there is no algorithmic parameter available to perform more than one run instance on a given problem (for NOMAD and BFO this can be achieved by changing the seed of the pseudorandom number generator). Hence, a single run instance is performed per problem, but 10 different starting points are considered, giving a total of 120 computational problems.



(a) $\tau = 10^{-3}$ (b) $\tau = 10^{-5}$ (c) $\tau = 10^{-7}$

FIG. 4.2. *Data profiles obtained with convergence tolerance* $\tau = 10^{-3}$, $\tau = 10^{-5}$, *and* $\tau = 10^{-7}$ *on* 10 *run instances of* 94 *mixed continuous, integer, and binary analytical problems (see Tables* 4.1 *and* 4.2*).*

In addition to the problem definition, the input arguments for MISO are the maximum number of allowable blackbox evaluations, the type of radial basis function used for the surrogate model, and the sampling strategies. We enabled the option of MISO to provide a unique initial point to define a problem.

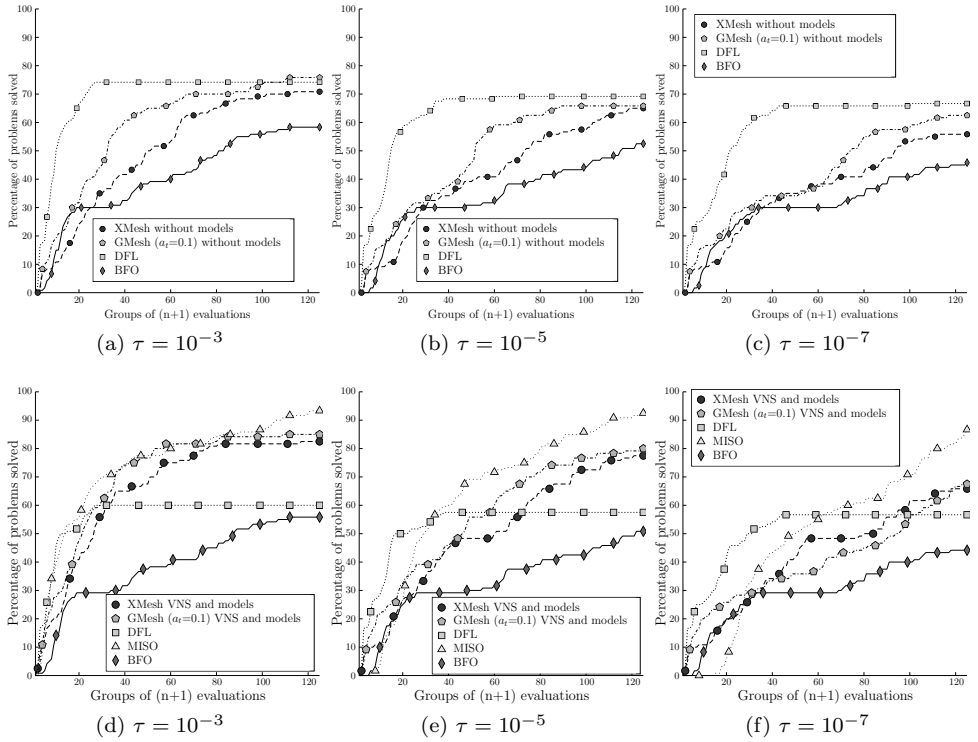
The MISO solver default settings for the sampling strategy (`cptvl`) and the surrogate model radial basis function (`rbf_c`) were considered. Preliminary experiments showed that the sampling strategy in the MISO solver allows one to perform a better exploration of the design space than that obtained with the default settings of NOMAD. For a fair comparison, the optional variable neighborhood search [6] available for XMesh and GMesh to escape local minimums was enabled on these problems. The DFL and BFO solvers have no equivalent option available.

For a fair comparison of solvers using (or not) a cache, multiple evaluations of the same points are not counted in the statistics. For the MISO solver, as reported in [55], the execution time to create new points tends to increase significantly with the number of points already evaluated (see Table 4.3). To obtain a practical overall execution time for MISO, we considered a maximum of $150 \times n$ blackbox evaluations, similar to the number used in [55], with a limit of 2,000 evaluations. From our observations, the important computational cost of the MISO solver is possibly due to the cost of constructing surrogates, which grows rapidly with the number of points evaluated. It highlights the fact that MISO should rather be used, as intended by its author, only with computationally costly blackboxes. The cost of the model construction in NOMAD is much smaller.

TABLE 4.3

Optimization times for three selected problems with $n \in \{5, 10, 15\}$ *from a single starting point with an evaluation budget of* $150 \times n$ *and a limit of* 2,000 *evaluations for* MISO*. Solvers with quadratic models disabled are tagged with a "†" symbol. Runs stopping before reaching the maximum evaluation budget are tagged with a "∗" symbol.*

		Optimization time in seconds						
		Without models				With models		
n	Name	BFO	DFL	GMesh†	XMesh†	GMesh	XMesh	MISO
5	`MISO4`	0.1*	1*	1*	1*	6	5	69
10	`SOMI11`	0.3	3*	3*	4*	16	16	298
15	`MISO6`	0.2	8*	12	12	58	57	2,168



(a) $\tau = 10^{-3}$ (b) $\tau = 10^{-5}$ (c) $\tau = 10^{-7}$

(d) $\tau = 10^{-3}$ (e) $\tau = 10^{-5}$ (f) $\tau = 10^{-7}$

FIG. 4.3. *Data profiles obtained with convergence tolerances* $\tau = 10^{-3}$, $\tau = 10^{-5}$, *and* $\tau = 10^{-7}$ *on* 12 *continuous, integer, and binary analytical problems without constraints (see Table* 4.2*) from* 10 *starting points, with* MADS, DFL, MISO, *and* BFO. *The upper data profiles are obtained with quadratic models disabled in* NOMAD.

The BFO solver default algorithmic settings have been considered. Figure 4.3 presents the data profiles comparing the two versions of MADS with DFL, MISO, and BFO. The top plots compare solvers that do not use models. DFL finds the best solutions for a fixed number of function evaluations. Both XMesh and GMesh are comparable, with a slight advantage for GMesh. The solutions produced by BFO are dominated by the solutions of the other methods, but the computational times are much lower. The three plots in the bottom part of the figure compare all methods with their default parameters (i.e., the model constructions in XMesh and GMesh are not disabled). DFL usually finds a good solution fast, but then stops without using the whole evaluation budget. This indicates the lack of a mechanism to escape local optima. MISO performs well in terms of the number of function evaluations, but not in terms of the elapsed time. GMesh is slightly better than XMesh, and both methods perform well. Comparison of the top and bottom plots shows that the MADS utilization of models improves the quality of the solutions.

4.5. Tuning trust-region parameters. This section presents computational results on the parameter tuning optimization problem outlined in the motivation of this work in section 1.1. This blackbox problem is taken from [17], and consists in finding values of a set of four continuous parameters of a trust-region (TR) optimization algorithm:

$$p = (\eta_1, \eta_2, \alpha_1, \alpha_2) \in \Omega = \{p \in \mathbb{R}^4 : 0 \leq \eta_1 < \eta_2 < 1 \text{ and } 0 < \alpha_1 < 1 < \alpha_2 \leq 10\}.$$

The objective function value $f^{2006}(p)$ for the parameter tuning optimization problem is the sum of CPU times to solve 55 problems using a TR with parameter values $p \in \Omega$ (the superscript denotes the year in which the paper was published). If $p \notin \Omega$, then $f^{2006}(p) = +\infty$.

Each TR problem has its own objective function f_{pb}. The TR algorithm stops as soon as an iterate x_k satisfies $\|\nabla f_{pb}(x_k)\|_2 \leq 10^{-5}$. The algorithm also stops when this criterion is not met after a given number of iterations. In the computational experiments of [17], the maximum number of iterations was set to 1,000. For the classical TR parameters $p^c = (1/4, 3/4, 1/2, 2)$, the objective function value was $f^{2006}(p^c) = 13{,}461$ seconds on a 500 MHz Sun Blade 100 computer, which means that it took almost 4 hours for TR to solve the collection of problems. The best solution reported in [17] is $p^* = (0.22125, 0.94457031, 0.37933594, 2.3042969)$ with $f^{2006}(p^*) = 10{,}193$ seconds, an improvement of approximately 30%. With p^c, 10 problems reached the maximum number of iterations (9 problems for p^*).

TABLE 4.4

Detailed computational times of the objective functions f^{2006} and f^{2018} indicated in seconds for the TR parameter tuning problem. The problems are sorted in decreasing time based on the values obtained when evaluating $f^{2018}(p^c)$. The eight problems taking more than 50 seconds are listed individually and the remaining problems are grouped. The problems with a time in bold font reached the execution time limit or the maximum number of iterations.

Problems	$f^{2018}(p^c)$	$f^{2006}(p^c)$	$f^{2006}(p^*)$	$f^{2018}(p^*)$
1 – NONCVXUN	**180.0**	**708.8**	**555.8**	**180.0**
2 – GENHUMPS	**180.0**	**615.3**	**445.0**	**180.0**
3 – FLETCBV3	96.8	**314.5**	**420.2**	121.8
4 – FLETCHBV	95.1	**315.0**	**408.2**	115.5
5 – NONCVXU2	80.9	**647.5**	**539.3**	76.8
6 – NCB20B	69.5	1,444.5	1,152.8	52.0
7 – CHAINWOO	68.8	**1,224.3**	**1,224.1**	72.9
8 – EIGENALS	52.0	1,768.3	1,177.2	41.8
Total pbs 1 to 8	823.1	7,038.0	5,922.5	841.1
Total pbs 9 to 55	184.9	6,423.0	4,270.6	183.3
Total pbs 1 to 55	1,008.0	13,461.0	10,193.1	1,024.4

The computational experiments of [17] were conducted more than 10 years ago, with a maximum number of iterations of 1,000. Now, with the same maximum number of iterations, on an Intel(R) Core(TM) i7 CPU 3.40 GHz computer, the sum of CPU times to solve 55 problems with p^c is 375 seconds, and the user time when solving in parallel on four processors (of the six available) is around 90 seconds.

In order to reduce the number of problems reaching the maximum number of iterations that do not satisfy the TR convergence criterion, we increased the limit to 9,999 iterations and set a CPU time limit of 180 seconds to solve each problem. This results in a new optimization problem, in which the new objective function $f^{2018}(p)$ is still the sum of CPU times to solve 55 problems for a set of continuous parameters p of a TR algorithm but the measure is more suited to the current computer's capability. Again, if $x \notin \Omega$, then $f^{2018}(p) = +\infty$. We now obtain $f^{2018}(p^c) = 1{,}008$ seconds and there are two problems that do not converge within the time limit (none reached the iteration limit) on an Intel(R) Core(TM) i7 CPU@3.40 GHz computer. With p^*, two problems (NONCVXUN and GENHUMPS) reach the maximum CPU time limit without reaching the convergence criterion on the gradient of f and $f^{2018}(p^*) =$

1,024 seconds. Table 4.4 breaks down the computational times of the 2006 and 2018 versions, for both point p^c and point p^*.

An evaluation of $f^{2018}(p)$ depends on the workload of the computer, and to limit the impact of this factor on the optimization a single optimization is conducted at a time with no other significant workload. Even with this precaution, the tuning TR parameters optimization problem has a stochastic nature. The range of values for $f^{2018}(p^c)$ on several consecutive runs ranges between 1,001 and 1,099 seconds. Hence, the RobustMads algorithm [15] is used to optimize on a smoothed version of f^{2018} and the quadratic models are disabled.

A maximum budget of 500 evaluations is considered during optimization in addition to the default minimum mesh size convergence criterion of NOMAD. Setting a maximum budget allows the optimization time to be limited to one day. Preliminary tests showed that solutions are not significantly improved with respect to the variability on evaluations when removing this budget criterion.

Table 4.5

Solutions obtained for the TR parameter tuning problem with different algorithms. The last column indicates the relative improvement over p^c.

Algo.	Solution	f^{2018}	Improv. (%)
Classical	p^c = (0.25, 0.75, 0.5, 2)	1,008.0	0
XMesh	p^x = (0.2939819787, 0.979406601, 0.4716387306, 1.474147761)	733.6	27
GMesh no $\delta^{\min}$	p^{no} = (0.672010424, 0.685829734, 0.061485394, 1.34816385)	727.0	28
GMesh $\delta^{\min} = 0.005$	p^{005} = (0.845, 0.99, 0.485, 1.575)	697.6	31
GMesh $\delta^{\min} = 0.01$	p^{01} = (0.74, 0.99, 0.17, 1.34)	688.7	32
GMesh $\delta^{\min} = 0.05$	p^{05} = (0.2, 0.9, 0.2, 1.3)	768.4	24

Table 4.6

Detailed computational time of the objective function value $f^{2018}(p)$ given in seconds for the TR parameter tuning problem. The problems are sorted in decreasing time based on the values obtained when evaluating $f^{2018}(p^c)$. The eight problems accounting taking more than 50 seconds are listed individually and the remaining problems are grouped. The problems with a time in bold font reached the execution time limit.

Problems	$f^{2018}(p^c)$	$f^{2018}(p^x)$	$f^{2018}(p^{no})$	$f^{2018}(p^{005})$	$f^{2018}(p^{01})$	$f^{2018}(p^{05})$
1 – NONCVXUN	**180.0**	**180.0**	**180.0**	**180.0**	**180.0**	**180.0**
2 – GENHUMPS	**180.0**	**180.0**	86.7	83.8	96.1	**180.0**
3 – FLETCBV3	96.8	18.1	17.5	17.6	17.4	18.0
4 – FLETCHBV	95.1	17.3	17.5	16.0	19.6	17.5
5 – NONCVXU2	80.9	55.3	58.1	55.4	51.0	55.2
6 – NCB20B	69.5	38.4	60.0	59.6	48.8	34.6
7 – CHAINWOO	68.8	47.1	55.0	77.4	56.6	46.5
8 – EIGENALS	52.0	31.0	58.0	40.0	43.2	34.2
Total pbs 1 to 8	823.1	567.1	532.8	529.7	512.9	566.1
Total pbs 9 to 55	184.9	166.5	194.2	167.9	175.8	202.4
Total pbs 1 to 55	1,008.0	733.6	727.0	697.6	688.7	768.4

Controlling the mesh granularity is straightforward using Mads with GMesh, as stated in section 3, with the $\delta^{\min}$ parameter.

Table 4.5 compares the classical parameter p^c with the solutions obtained by XMesh and GMesh, using (or not) a minimal granularity. Detailed computational times are displayed in Table 4.6. Note that the optimized solutions are rather equivalent in terms of quality, but that the ones with a controlled granularity are more

elegant and simpler to report, thereby increasing the chances that these values are actually used in practice.

5. Discussion. This work introduces a novel way to update the mesh and poll size vectors in the MADS algorithm for derivative-free and blackbox optimization. An advantage of this new approach is that the number of decimals in the solutions explored by the algorithm is controlled. This allows a new treatment of integer variables, and of variables that have a minimal granularity.

Computational experiments confirm the new MADS variant (GMesh) is slightly better than the previous variant (XMesh) on both continuous and discrete problems, and that it is competitive with, if not better than, the state-of-the-art DFO solvers DFL, MISO, and BFO.

The new strategy is compatible with all the extensions of MADS: the biobjective version of the algorithm [18], the extreme [9] and progressive barrier [10] approaches for handling constraints, the parallel space decomposition technique [11], the recent dynamic scaling of [16], and the robust version of the algorithm [15], for instance.

REFERENCES

[1] M. ABRAMSON, *Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm*, Optim. Eng., 5 (2004), pp. 157–177, https://doi.org/10.1023/B:OPTE.0000033373.79886.54.

[2] M. ABRAMSON, C. AUDET, J. CHRISSIS, AND J. WALSTON, *Mesh adaptive direct search algorithms for mixed variable optimization*, Optim. Lett., 3 (2009), pp. 35–47, https://doi.org/10.1007/s11590-008-0089-2.

[3] C. AUDET, S. LE DIGABEL, V. ROCHON MONTPLAISIR, AND C. TRIBES, *The NOMAD Project*; software available at https://www.gerad.ca/nomad, 2015.

[4] M. ABRAMSON, C. AUDET, AND J. DENNIS, JR., *Filter pattern search algorithms for mixed variable constrained optimization problems*, Pac. J. Optim., 3 (2007), pp. 477–500, http://www.ybook.co.jp/online/pjoe/vol3/pjov3n3p477.html.

[5] C. AUDET, *Convergence results for generalized pattern search algorithms are tight*, Optim. Eng., 5 (2004), pp. 101–122, https://doi.org/10.1023/B:OPTE.0000033370.66768.a9.

[6] C. AUDET, V. BÉCHARD, AND S. LE DIGABEL, *Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search*, J. Global Optim., 41 (2008), pp. 299–318, https://doi.org/10.1007/s10898-007-9234-1.

[7] C. AUDET AND J. DENNIS, JR., *Pattern search algorithms for mixed variable programming*, SIAM J. Optim., 11 (2001), pp. 573–594, https://doi.org/10.1137/S1052623499352024.

[8] C. AUDET AND J. DENNIS, JR., *Analysis of generalized pattern searches*, SIAM J. Optim., 13 (2003), pp. 889–903, https://doi.org/doi:10.1137/S1052623400378742.

[9] C. AUDET AND J. DENNIS, JR., *Mesh adaptive direct search algorithms for constrained optimization*, SIAM J. Optim., 17 (2006), pp. 188–217, https://doi.org/10.1137/040603371.

[10] C. AUDET AND J. DENNIS, JR., *A progressive barrier for derivative-free nonlinear programming*, SIAM J. Optim., 20 (2009), pp. 445–472, https://doi.org/10.1137/070692662.

[11] C. AUDET, J. DENNIS, JR., AND S. LE DIGABEL, *Parallel space decomposition of the mesh adaptive direct search algorithm*, SIAM J. Optim., 19 (2008), pp. 1150–1170, https://doi.org/10.1137/070707518.

[12] C. AUDET AND W. HARE, *Derivative-Free and Blackbox Optimization*, Springer Ser. Oper. Res. Financ. Eng., Springer, Cham, 2017, https://doi.org/10.1007/978-3-319-68913-5.

[13] C. AUDET AND W. HARE, *Model-based methods in derivative-free nonsmooth optimization*, in Numerical Nonsmooth Optimization, A. Bagirov, M. Gaudioso, N. Karmitsa, M. M. Mäkelä, and S.Taheri, eds., Springer, New York, 2019.

[14] C. AUDET, A. IANNI, S. LE DIGABEL, AND C. TRIBES, *Reducing the number of function evaluations in mesh adaptive direct search algorithms*, SIAM J. Optim., 24 (2014), pp. 621–642, https://doi.org/10.1137/120895056.

[15] C. AUDET, A. IHADDADENE, S. LE DIGABEL, AND C. TRIBES, *Robust optimization of noisy blackbox problems using the mesh adaptive direct search algorithm*, Optim. Lett., 12 (2018), pp. 675–689, https://doi.org/10.1007/s11590-017-1226-6.

[16] C. AUDET, S. LE DIGABEL, AND C. TRIBES, *Dynamic scaling in the mesh adaptive direct*

search algorithm for blackbox optimization, Optim. Eng., 17 (2016), pp. 333–358, https://doi.org/10.1007/s11081-015-9283-0.

[17] C. Audet and D. Orban, *Finding optimal algorithmic parameters using derivative-free optimization*, SIAM J. Optim., 17 (2006), pp. 642–664, http://dx.doi.org/doi:10.1137/040620886.

[18] C. Audet, G. Savard, and W. Zghal, *Multiobjective optimization through a series of single-objective formulations*, SIAM J. Optim., 19 (2008), pp. 188–210, https://doi.org/10.1137/060677513.

[19] C. Audet and C. Tribes, *Mesh-based Nelder–Mead algorithm for inequality constrained optimization*, Comput. Optim. Appl., 71 (2018), pp. 331–352, https://doi.org/10.1007/s10589-018-0016-0.

[20] E. Brea, *An extension of Nelder–Mead method to nonlinear mixed-integer optimization problems*, Rev. Int. Métod. Numér. Cálc. Diseño Ing., 29 (2013), pp. 163–174, https://doi.org/10.1016/j.rimni.2013.06.005 (in Spanish).

[21] Y. Cao, L. Jiang, and Q. Wu, *An evolutionary programming approach to mixed-variable optimization problems*, Appl. Math. Model., 24 (2000), pp. 931–942, https://doi.org/10.1016/S0307-904X(00)00026-3.

[22] M. Cardoso, R. Salcedo, S. de Azevedo, and D. Barbosa, *A simulated annealing approach to the solution of minlp problems*, Comput. Chem. Eng., 21 (1997), pp. 1349–1364, https://doi.org/10.1016/S0098-1354(97)00015-X.

[23] X. Chen and N. Wang, *Optimization of short-time gasoline blending scheduling problem with a DNA based hybrid genetic algorithm*, Chem. Eng. Process., 49 (2010), pp. 1076–1083, https://doi.org/10.1016/j.cep.2010.07.014.

[24] R. Coelho, *Metamodels for mixed variables based on moving least squares*, Optim. Eng., 15 (2013), pp. 311–329, https://doi.org/10.1007/s11081-013-9216-8.

[25] A. Conn and S. Le Digabel, *Use of quadratic models with mesh-adaptive direct search for constrained black box optimization*, Optim. Methods Softw., 28 (2013), pp. 139–158, https://doi.org/10.1080/10556788.2011.623162.

[26] A. Conn, K. Scheinberg, and L. Vicente, *Introduction to Derivative-Free Optimization*, MOS-SIAM Ser. Optim., SIAM, Philadelphia, PA, 2009, https://doi.org/10.1137/1.9780898718768.

[27] A.-S. Crélot, C. Beauthier, D. Orban, C. Sainvitu, and A. Sartenaer, *Combining Surrogate Strategies with MADS for Mixed-Variable Derivative-Free Optimization*, Technical report G-2017-70, Les cahiers du GERAD, Montreal, 2017; available at https://doi.org/10.13140/RG.2.2.25690.24008.

[28] C. Davis, *Theory of positive linear dependence*, Amer. J. Math., 76 (1954), pp. 733–746, https://www.jstor.org/stable/2372648.

[29] E. Fermi and N. Metropolis, *Numerical Solution of a Minimum Problem*, Los Alamos Unclassified Report LA-1492, Los Alamos National Laboratory, Los Alamos, NM, 1952.

[30] U. García-Palomares, E. Costa-Montenegro, R. Asorey-Cacheda, and F. González-Castaño, *Adapting derivative free optimization methods to engineering models with discrete variables*, Optim. Eng., 13 (2012), pp. 579–594, https://doi.org/10.1007/s11081-011-9168-9.

[31] N. Gould, D. Orban, and P. Toint, *CUTEr (and SifDec): A constrained and unconstrained testing environment, revisited*, ACM Trans. Math. Software, 29 (2003), pp. 373–394, https://doi.org/10.1145/962437.962439.

[32] Y. He, J. Zhou, N. Lu, H. Qin, and Y. Lu, *Differential evolution algorithm combined with chaotic pattern search*, Kybernetika (Prague), 46 (2010), pp. 684–696, http://eudml.org/doc/196547.

[33] A.-R. Hedar, *Global Optimization Test Problems*, http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm (20 October, 2017).

[34] T. Hemker, K. Fowler, M. Farthing, and O. Stryk, *A mixed-integer simulation-based optimization approach with surrogate functions in water resources management*, Optim. Eng., 9 (2008), pp. 341–360, https://doi.org/10.1007/s11081-008-9048-0.

[35] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Econom. and Math. Systems 187, Springer, Cham, 1981.

[36] M. Jamil and X.-S. Yang, *A literature survey of benchmark functions for global optimisation problems*, Internat. J. Math. Model. Numer. Optim., 4 (2013), pp. 150–194, https://doi.org/10.1504/IJMMNO.2013.055204.

[37] S. Kitayama, M. Arakawa, and K. Yamazaki, *Sequential approximate optimization using radial basis function network for engineering optimization*, Optim. Eng., 12 (2011), pp. 535–557, https://doi.org/10.1007/s11081-010-9118-y.

[38] M. Kokkolaras, C. Audet, and J. Dennis, Jr., *Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system*, Optim. Eng., 2 (2001), pp. 5–29, https://doi.org/10.1023/A:1011860702585.

[39] M. Laguna, F. Gortázar, M. Gallego, A. Duarte, and R. Martí, *A black-box scatter search for optimization problems with integer variables*, J. Global Optim., 58 (2014), pp. 497–516, https://doi.org/10.1007/s10898-013-0061-2.

[40] S. Le Digabel, *Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm*, ACM Trans. Math. Software, 37 (2011), pp. 44:1–44:15, https://doi.org/10.1145/1916461.1916468.

[41] S. Le Digabel and S. Wild, *A Taxonomy of Constraints in Simulation-Based Optimization*, Technical report G-2015-57, Les cahiers du GERAD, Montreal, 2015; available at http://www.optimization-online.org/DB_HTML/2015/05/4931.html.

[42] T. Liao, K. Socha, M. M. de Oca, T. Stützle, and M. Dorigo, *Ant colony optimization for mixed-variable optimization problems*, IEEE Trans. Evol. Comput., 18 (2014), pp. 503–518, https://doi.org/10.1109/TEVC.2013.2281531.

[43] G. Liuzzi, S. Lucidi, and F. Rinaldi, *Derivative-free methods for bound constrained mixed-integer optimization*, Comput. Optim. Appl., 53 (2012), pp. 505–526, https://doi.org/10.1007/s10589-011-9405-3.

[44] G. Liuzzi, S. Lucidi, and F. Rinaldi, *derivative-free methods for mixed-integer constrained optimization problems*, J. Optim. Theory Appl., 164 (2015), pp. 933–965, https://doi.org/10.1007/s10957-014-0617-4.

[45] G. Liuzzi, S. Lucidi, and F. Rinaldi, *An Algorithmic Framework Based on Primitive Directions and Nonmonotone Line Searches for Black Box Problems with Integer Variables*, Technical report 2018-02-6471, http://www.optimization-online.org/DB_HTML/2018/02/6471.html, 2018.

[46] S. Lucidi, M. Maurici, L. Paulon, F. Rinaldi, and M. Roma, *A derivative-free approach for a simulation-based optimization problem in healthcare*, Optim. Lett., 10 (2016), pp. 219–235, https://doi.org/10.1007/s11590-015-0905-4.

[47] S. Lucidi and V. Piccialli, *A derivative-based algorithm for a particular class of mixed variable optimization problems*, Optim. Methods Softw., 19 (2004), pp. 371–387, https://doi.org/10.1080/10556780410001654197.

[48] S. Lucidi, V. Piccialli, and M. Sciandrone, *An algorithm model for mixed variable programming*, SIAM J. Optim., 15 (2005), pp. 1057–1084, https://doi.org/10.1137/S1052623403429573.

[49] L. Lukšan and J. Vlček, *Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization*, Technical report V-798, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2000; available at http://hdl.handle.net/11104/0124190.

[50] K. McKinnon, *Convergence of the Nelder–Mead simplex method to a nonstationary point*, SIAM J. Optim., 9 (1998), pp. 148–158, https://doi.org/10.1137/S1052623496303482.

[51] E. Mezura-Montes and C. Coello, *Useful infeasible solutions in engineering optimization with evolutionary algorithms*, in Proceedings of the 4th Mexican International Conference on Advances in Artificial Intelligence, Lecture Notes in Comput. Sci. 3789, Springer, Berlin, 2005, pp. 652–662, https://doi.org/10.1007/11579427_66.

[52] N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović, *Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search*, European J. Oper. Res., 151 (2003), pp. 389–399, https://doi.org/10.1016/S0377-2217(02)00833-0.

[53] J. Moré, B. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41, https://doi.org/10.1145/355934.355936.

[54] J. Moré and S. Wild, *Benchmarking derivative-free optimization algorithms*, SIAM J. Optim., 20 (2009), pp. 172–191, https://doi.org/10.1137/080724083.

[55] J. Müller, *MISO: Mixed-integer surrogate optimization framework*, Optim. Eng., 17 (2016), pp. 177–203, https://doi.org/10.1007/s11081-015-9281-2.

[56] J. Müller, C. Shoemaker, and R. Piché, *SO-MI: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems*, Comput. Oper. Res., 40 (2013), pp. 1383–1400, https://doi.org/10.1016/j.cor.2012.08.022.

[57] J. Müller, C. Shoemaker, and R. Piché, *SO-I: A surrogate model algorithm for expensive nonlinear integer programming problems including global optimization applications*, J. Global Optim., 59 (2014), pp. 865–889, https://doi.org/10.1007/s10898-013-0101-y.

[58] E. Newby and M. Ali, *A trust-region-based derivative free algorithm for mixed integer programming*, Comput. Optim. Appl., 60 (2015), pp. 199–229, https://doi.org/10.1007/s10589-014-9660-1.

[59] F. Pigache, F. Messine, and B. Nogarede, *Optimal design of piezoelectric transformers: A rational approach based on an analytical model and a deterministic global optimization*, IEEE Trans. Ultrason., Ferroelectr., Freq. Control, 54 (2007), pp. 1293–1302, https://doi.org/10.1109/TUFFC.2007.390.

[60] G. Pillo, G. Fasano, G. Liuzzi, S. Lucidi, V. Piccialli, F. Rinaldi, and M. Sciandrone, *DFL – A derivative-free library: A software library for derivative-free optimization*; software available at http://www.dis.uniroma1.it/~lucidi/DFL/, 2015.

[61] M. Porcelli and P. Toint, *BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables*, ACM Trans. Math. Software, 44 (2017), pp. 6:1–6:25, https://doi.org/10.1145/3085592.

[62] J. Rodríguez, J. Renaud, and L. Watson, *Trust region augmented lagrangian methods for sequential response surface approximation and optimization*, J. Mech. Design, 120 (1998), pp. 58–66, https://doi.org/10.1115/1.2826677.

[63] K. Socha, *ACO for continuous and mixed-variable optimization*, in Ant Colony Optimization and Swarm Intelligence, M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, and T. Stützle, eds., Lecture Notes in Comput. Sci. 3172, Springer, Cham, 2004, pp. 25–36, https://doi.org/10.1007/978-3-540-28646-2_3.

[64] T. Sriver, J. Chrissis, and M. Abramson, *Pattern search ranking and selection algorithms for mixed variable simulation-based optimization*, European J. Oper. Res., 198 (2009), pp. 878–890, https://doi.org/10.1016/j.ejor.2008.10.020.

[65] J. Tao and N. Wang, *DNA double helix based hybrid GA for the gasoline blending recipe optimization problem*, Chem. Eng. Tech., 31 (2008), pp. 440–451, https://doi.org/10.1002/ceat.200700322.

[66] V. Torczon, *On the convergence of pattern search algorithms*, SIAM J. Optim., 7 (1997), pp. 1–25, https://doi.org/10.1137/S1052623493250780.

[67] R. Torres, C. Bès, J. Chaptal, and J.-B. Hiriart-Urruty, *Optimal, environmentally-friendly departure procedures for civil aircraft*, J. Aircraft, 48 (2011), pp. 11–22, https://doi.org/10.2514/1.C031012.

[68] C. Tribes, J.-F. Dubé, and J.-Y. Trépanier, *Decomposition of multidisciplinary optimization problems: Formulations and application to a simplified wing design*, Eng. Optim., 37 (2005), pp. 775–796, https://doi.org/10.1080/03052150500289305.

[69] S. Tsai and S. Fu, *Genetic-algorithm-based simulation optimization considering a single stochastic constraint*, European J. Oper. Res., 236 (2014), pp. 113–125, https://doi.org/10.1016/j.ejor.2013.11.034.

[70] L. Vicente, *Implicitly and densely discrete black-box optimization problems*, Optim. Lett., 3 (2009), pp. 475–482, https://doi.org/10.1007/s11590-009-0120-2.

[71] H. Wang, *Subspace dynamic-simplex linear interpolation search for mixed-integer black-box optimization problems*, Naval Res. Logist., 64 (2017), pp. 305–322, https://doi.org/10.1002/nav.21747.

[72] K. Wang and N. Wang, *A novel RNA genetic algorithm for parameter estimation of dynamic systems*, Chem. Eng. Res. Design, 88 (2010), pp. 1485–1493, https://doi.org/10.1016/j.cherd.2010.03.005.

[73] E. Wyers, M. Steer, C. Kelley, and P. Franzon, *A bounded and discretized Nelder–Mead algorithm suitable for RFIC calibration*, IEEE Trans. Circuits Syst. I, Reg. Papers, 60 (2013), pp. 1787–1799, https://doi.org/10.1109/TCSI.2012.2230496.

[74] H. Yang, J. Kim, and J. Choe, *Field development optimization in mature oil reservoirs using a hybrid algorithm*, J. Petrol. Sci. Eng., 156 (2017), pp. 41–50, https://doi.org/10.1016/j.petrol.2017.05.009.

[75] J. Zhao and N. Wang, *A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling*, Comput. Chem. Eng., 35 (2011), pp. 272–283, https://doi.org/10.1016/j.compchemeng.2010.01.008.

[76] Z. Zhao, J. Meza, and M. V. Hove, *Using pattern search methods for surface structure determination of nanomaterials*, J. Phys. Cond. Matter, 18 (2006), pp. 8693–8706, https://doi.org/10.1088/0953-8984/18/39/002.