The mesh adaptive direct search algorithm for granular and discrete variables *

Charles Audet† Sébastien Le Digabel‡ Christophe Tribes§

March 16, 2018

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.

Keywords: Blackbox optimization, Derivative-free optimization, Mesh adaptive direct search, Granular variables, Discrete variables.

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

1 Introduction and context

1.1 Motivation

This work studies derivative-free optimization (DFO) [12, 25], and more precisely black-box 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, that 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 which is typically an integer power of four. In [15], 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 [16], the authors study the question of finding values of the four standard parameters of a 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 [16] suggest the values \(\bar{p} = (0.22125, 0.94457031, 0.37933594, 2.3042969)\) which reduce the computational time by 25% on their testbed. These proposed values might be interesting for the theoretician, but are difficult to democratize because of the number of decimals. A solution controlling the number of decimals to the hundredth or thousandth would have been easier to popularize.

The present work considers the optimization problem:

\[
\min_{x \in \mathcal{X} \subseteq \mathbb{R}^n} f(x) \quad \text{s.t.} \quad c(x) \leq 0
\]

where \(\mathcal{X} = \{x \in \mathbb{R}^n : \ell \leq x \leq u \text{ and } x_i/\delta_{i}^{\min} \in \mathbb{Z} \text{ for all } i \in G\}\). 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 to launch 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 \(G\) 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 \(i \in G\). This means that the \(i\)-th 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 \(G\) 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 introduced notation, 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 2 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 [40], 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 $\infty$ whenever $x \in \mathbb{R}^n \setminus \mathcal{X}$. The other constraints $c(x) \leq 0$ 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 $c(x) \leq 0$ 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, 39] 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 is available since the 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) 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, 46, 47, 64]. Applications of these algorithms can be found in [1, 37, 75].

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 [42, 43, 44], where several algorithmic variations are introduced, and for which global convergence based on a sufficient decrease is proved. The most recent of these algorithms handle general constraints with a sequential penalty approach. These methods are publicly
available at the DFL library webpage [60], except for the more recent technical report [44]. An example of a realistic application solved using this library is available in [45]. Other approaches based on surrogate models are given in [23, 26, 55, 56, 57]. One of them (SO-I) considers constraints with a two-phase framework where feasibility is the priority of the first phase. The MISO implementation for the box-constrained case, in MATLAB, is available at J. Mueller’s homepage [54]. Surrogate models are also used in [33] for a specific application, search directions for the box-constrained case in [29], and projection in [69]. Other DFO methods have also been adapted to the mixed-integer case: The 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 [19, 72]. Articles in Engineering journals also consider the solution of mixed-integer blackbox problems, such as [70, 73]. Finally, several heuristics exist, such as [20, 21, 31, 38, 41, 63, 68], 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, 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 [14] 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. The present section is divided into four subsections. The first one illustrates the simplest strictly polling algorithm. Section 2.2 introduces a new mechanism that allows to control the number of decimals for that simple algorithm. Then, Section 2.3 applies the mechanism to a broader algorithmic class. Finally, 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 [28]. 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 $x^{k+1}$, the iteration ends and is labelled as being successful. If none of these trial points improves $f$, the iteration is said to be unsuccessful and $x^{k+1}$ 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 $k$-th 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 $i$-th 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 is found, or if 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.

**Algorithm 1: The coordinate search algorithm (Cs)**

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

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 $\delta^{k+1} = \frac{1}{2}\delta^k$

There are two possible behaviours for this simple algorithm. Either the sequence of trial points $\{x^k\}_{k=0}^\infty$ belongs to a bounded set, or 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.

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 $\delta^k$ by multiplying or dividing it by the 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 if the three first iterations are unsuccessful, then the next poll step will make modifications of magnitude of $\delta^3 = \frac{1}{8} = 0.125$ and will modify up to the third decimal. For some users, this might be an undesired behaviour, as they would prefer a step size of 0.1. We propose to address this issue by making the step size parameter an integer multiple of a power of 10, rather than being a power of 2.

At each iteration the step size parameter $\delta^k$ will be constructed so that it belongs to the discrete set
\[
P := \{a \times (10)^b : a \in \{1, 2, 5\}, b \in \mathbb{Z}\}.
\]

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

\[
i_{\text{increase}}(a \times (10)^b) = \begin{cases} 
2 \times (10)^b & \text{if } a = 1 \\
5 \times (10)^b & \text{if } a = 2 \\
1 \times (10)^{b+1} & \text{if } a = 5 
\end{cases}
\]

\[
i_{\text{decrease}}(a \times (10)^b) = \begin{cases} 
5 \times (10)^{b-1} & \text{if } a = 1 \\
1 \times (10)^b & \text{if } a = 2 \\
2 \times (10)^b & \text{if } a = 5.
\end{cases}
\]

Each operator is the inverse of the other. This allows us to formulate Algorithm 2, a variant of CS in which the number of decimals is controlled.

This algorithm benefits from the same convergence results as in Theorem 2.1. We postpone the proof to Section 3.3 after that all algorithmic improvements are presented. We next introduce a way to use more polling directions than only the coordinate directions.
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 \( P^k := \{ x^k \pm \delta^k e_i : i \in \mathbb{N} \} \) |
|           | if the poll step was successful at a trial point \( t \in P^k \), then |
|           | set \( x^{k+1} = t \) and \( \delta^{k+1} = \text{increase}(\delta^k) \) |
|           | else |
|           | set \( x^{k+1} = x^k \) and \( \delta^{k+1} = \text{decrease}(\delta^k) \) |

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 exists. 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 in 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. A second parameter \( \delta^k > 0 \), called the mesh size parameter, is obtained from the integer \( b^k \) used to construct the poll size parameter:

\[
\delta^k := (10)^{b^k - \lfloor b^k \rfloor}. \quad (2)
\]

An equivalent way of writing Equation (2) would be to set \( \delta^k \) 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 \( \Delta^k \) 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 \( \Delta^k \) is an integer multiple of the mesh size parameter. The ratio of these two parameters is denoted by the integer \( \rho^k \), and will play a useful role in what follows:

\[
\rho^k := \frac{\Delta^k}{\delta^k} = \frac{a^k \times 10^{b^k}}{(10)^{b^k - \lfloor b^k \rfloor}} = a^k \times 10^{\lfloor b^k \rfloor} \in \mathbb{N}. \quad (2)
\]

Now, instead of setting the poll set to \( \{ x^k \pm \delta^k e_i : i \in \mathbb{N} \} \) we first construct \( D^k \) as a positive basis\(^1\) 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, 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

\[
P^k = \{ x^k + \delta^k d : d \in D^k \}. \quad (3)
\]

---

\(^1\) A positive basis in \( \mathbb{R}^n \) is a minimal set of vectors whose nonnegative linear combination span the entire space \( \mathbb{R}^n \) [27].
With this construction, it can be seen that the distance from any poll point to the incumbent solution \( x^k \) satisfies
\[
\|\delta^k d\|_\infty \leq \delta^k \times \rho^k = (10)^{|b^k|} \times a^k \times (10)^{|b^k|} = \Delta^k.
\] (4)

The left part of Figure 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, when iteration \( k \) 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 \( \rho^k \) 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. In the left part of the figure, the radius equals \( 5 \) 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 \( \{x^k\}^\infty_{k=0} \) produced by the rich polling algorithm belongs to a bounded set, then there exists a subsequence of unsuccessful iteration.
Algorithm 3: The rich polling algorithm

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

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 $\Delta^{k+1} = \text{increase}(\Delta^k)$
  else
    set $x^{k+1} = x^k$ and $\Delta^{k+1} = \text{decrease}(\Delta^k)$

indices for which the mesh and poll size parameters converge to zero, and 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 to devise a way to make sure 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 [15, 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

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

where the function $\text{round} : \mathbb{R}^n \to \mathbb{Z}^n$ rounds upward all entries of the input vector in $\mathbb{R}^n$ to the nearest integer. This way of constructing ensures that $D^k$ forms an integer maximal positive basis of $\mathbb{R}^n$ [15, Proposition 9.1]. Equation (4) 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 way of generating the polling directions, Algorithm 3 is entirely parameter-free. The only input that 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 more globally the space of variables. 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

$$M^k := \{ x + \text{diag}(\delta^k)z : x \in V^k, z \in \mathbb{Z}^n \}$$  \hspace{1cm} (5)

where $V^k$ is the set of all trial points at which the simulation was evaluated by the start of iteration $k$, and where $\delta^k = (\delta^k_1, \delta^k_2, \ldots, \delta^k_n) \in \mathcal{P}^n$ is not a scalar, it is the mesh size vector composed of mesh size parameters. Going back to Figure 1, the mesh is represented by the intersection of all horizontal and vertical lines, not only 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 hopes of improving the incumbent solution $x^k$. There are two possible outcomes of the search step: either it succeeds in doing so or it fails. When it is unsuccessful, the poll step is invoked. During this step the simulation is launched at the trial points from the set

$$P^k := \{ x^k + \text{diag}(\delta^k)d : d \in D^k \} \subset M^k$$  \hspace{1cm} (6)

where $D^k \subset \mathbb{Z}^n$ is a positive spanning set satisfying $-\Delta^k \leq \text{diag}(\delta^k)d \leq \Delta^k$ for every $d \in D^k$, and where $\Delta^k = (\Delta^k_1, \Delta^k_2, \ldots, \Delta^k_n) \geq \delta^k$ is the poll size vector in $\mathcal{P}^n$. In constrained optimization [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 (6) generalizes (3) by allowing individual scaling for each variable. As for the search, the two same outcomes are possible for the poll. The entire iteration is said to be successful if a new incumbent is found during either the search 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, radial basis functions, etc. The search step allows to couple MADS with other optimization techniques. Strategies such as latin hypercube sampling or VNS searches [6] or Nelder-Mead searches [18] may be used in an attempt to escape local solutions or to accelerate convergence.
The role of 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 neighbourhood \( P_k \) parameterized by \( \Delta_k \). At the end of an unsuccessful iteration, the mesh and poll size parameters are all reduced. Otherwise, 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 \( \ell \) and \( u \) (which are possibly infinite). For each \( i \in \mathbb{N} \), set \( \alpha^0_i > 0 \) to the value

\[
\alpha^0_i := \begin{cases} 
\frac{|x^0_i|}{10} & \text{if no bound is provided for } x^0_i \text{ but } x^0_i \neq 0 \\
\frac{u_i - \ell_i}{10} & \text{if } x^0_i \text{ is bounded by the finite values } \ell_i < u_i \\
\frac{|x^0_i - w_i|}{10} & \text{if } x^0_i \text{ has only one bound } w_i \text{ which differs from } x^0_i \\
\frac{|x^0_i|}{10} & \text{if } x^0_i \text{ has only one bound which equals } x^0_i \text{ and } x^0_i \neq 0 \\
1 & \text{otherwise.}
\end{cases}
\]  

(7)

In the previous implementations of MADS [15], this value \( \alpha^0_i \) served as the initial poll size parameter for the \( i \)-th variable. In our context, we simply set \( \Delta^0_i \) to the value \( \alpha^0_i \times (10)^{b^0_i} \) in \( \mathcal{P} \) which is the closest to \( \alpha^0_i \) by a simple rounding operation. The initial poll size vector is \( \Delta^0 \in \mathcal{P}^n \).

Iteration \( k \) of the MADS algorithm is initiated with the poll size vector \( \Delta^k \in \mathcal{P}^n \). For each \( i \in \mathbb{N} \), the mesh size parameter \( \delta^k_i \) is obtained from the poll size parameter \( \Delta^k_i = a^k_i \times (10)^{b^k_i} \) as follows:

\[
\delta^k_i := (10)^{b^k_i - |b^k_i - b^0_i|}.
\]  

(8)

Equation (8) generalizes (2) by taking into account the initial scaling imposed on the variables. The mesh size parameter \( \delta^k_i \) equals \( (10)^{b^0_i} \) when \( b^k_i \) is greater than or equal to \( b^0_i \), and equals \( (10)^{2b^k_i - b^0_i} \) otherwise. This way of setting the mesh size vector will be written compactly in vector notation as \( \delta^k := (10)^{b^k - |b^k - b^0|} \).

Each trial point generated during the iteration lies on the mesh \( M^k \) from Equation (5). Notice that by construction, the poll size parameter \( \Delta^k_i \) is an integer multiple of the mesh size parameter \( \delta^k_i \). The ratio of these two parameters is denoted by

\[
\frac{\Delta^k_i}{\delta^k_i} = a^k_i \times 10^{|b^k_i - b^0_i|} \in \mathbb{N}.
\]
An equivalent way of writing this last equation is to define the ratio vector as \( \rho^k = (\rho_1^k, \rho_2^k, \ldots, \rho_n^k) := \text{diag}(\delta^k)^{-1} \Delta^k \).

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

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

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.

**Algorithm 4: MADS with initial scaling**

Input: \( x^0 \in \mathbb{R}^n \), the initial point
\( \Delta^0 \), the initial poll size vector where \( \Delta^0_i \) is the value \( a_i^0 \times (10)^{b_i^0} \) in \( \mathcal{P} \) which is the closest to \( \alpha_i^0 \) from Equation (7)

for iteration \( k = 0, 1, 2, \ldots \) do

- Set \( \delta^k := (10)^{b^k-|b^k-b^0|} \in \mathcal{P}^n \), the mesh size vector
- Set \( \rho^k := \text{diag}(\delta^k)^{-1} \Delta^k \in \mathbb{N}^n \), the ratio vector
- Search step: Test finitely many trial points on the mesh \( M^k \)
- Poll step: Test \( P^k := \{x^k + \text{diag}(\delta^k)d : d \in D^k \} \subset M^k \) where \( D^k \subset \mathbb{Z}^{n \times n} \) is a positive spanning set satisfying \(-\rho^k \leq d \leq \rho^k \) 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 \( \Delta^{k+1} = \text{increase}(\Delta^k) \) (*)

else

- set \( x^{k+1} = x^k \) and \( \Delta^{k+1} = \text{decrease}(\Delta^k) \)

(the star (*) will be referred to in Section 3.2)

### 3.2 Dynamic scaling of variables

In Algorithm 4, once that the initial scaling vector \( \Delta^0 \) is determined, the relative scaling between variables does not change much. This is because at successful iterations all entries of \( \Delta^k \) are increased, and at unsuccessful ones they are all decreased.

In previous work [15], we proposed to increase 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 \( \rho_i^k \). 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 $\Delta_{k+1}^i$ is also increased when the mesh size parameter $\delta_k^i$ is finer than the original one $\delta_0^i$ and the ratio $\rho_k^i$ exceeds the square of another ratio parameter $\rho_l^k$ for some index $l \in N$. Formally, we replace the line marked by a star (*) in Algorithm 4 by the following: set $x^{k+1} = t$ and

$$
\Delta_{k+1}^i = \begin{cases} 
\text{increase}(\Delta_k^i) & \text{if } |d_i|/\rho_k^i > a_t, \text{ or if } \delta_k^i < \delta_0^i \text{ and } \rho_k^i > (\rho_l^k)^2 \text{ for some } l \in N, \\
\Delta_k^i & \text{otherwise.}
\end{cases}
$$

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

### 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 are 1, 0.01 and 0.005, respectively. Of course, each one of these situations could be handled inside the blackbox by making the variable integer and then to multiply it by the corresponding granularity. But this requires to alter the blackbox, the bounds, the starting point, and it masks the actual numerical values to the user. Instead, we propose to handle the granularities within the MADS algorithm.

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

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

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

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

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

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

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

$$
\Delta_{k+1}^i = \begin{cases} 
\text{decrease}(\Delta_k^i) & \text{if } i \in N \setminus G, \\
\delta_i^{\text{min}} \times \max\{1, \text{decrease}\left(\frac{\Delta_k^i}{\delta_i^{\text{min}}}\right)\} & \text{if } i \in G.
\end{cases}
$$

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.
The corresponding mesh size parameter is obtained from the poll size parameter as follows:

\[
\delta_i^k := \begin{cases} 
(10)^{b_i^k - |b_i^k - b_i^0|} & \text{if } i \in N \setminus \mathcal{G}, \\
\delta_i^{\min} \times \max \left\{ 1, (10)^{b_i^k - |b_i^k - b_i^0|} \right\} & \text{if } i \in \mathcal{G}.
\end{cases}
\] (11)

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

\[
\rho_i^k := a_i^k \times \min \left\{ 10^{b_i^k}, 10^{b_i^0} \right\} \in \mathbb{N}.
\]

On successful iterations, the mesh size vector is updated as in the previous subsection, except that Equation (9) becomes:

\[
\Delta_{i+1}^k = \begin{cases} 
\text{increase}(\Delta_i^k) & \text{if } |d_i|/\rho_i^k > a_i, \text{ or if } \\
\Delta_i^k & \text{if } \delta_i < \delta_i^0 \text{ and } \rho_i^k > (\rho_i^k)^2 \text{ for some } l \in N \setminus \mathcal{G}, \end{cases}
\] (12)

for every index \( i \in N \). The difference with Equation (9) is that the comparison on the ratio parameters \( \rho_i^k > (\rho_i^k)^2 \) 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 to 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**

\[\text{Input: } \mathcal{G} \subseteq N, \text{ the indices of variables with minimal granularity} \]

\[x^0 \in \mathbb{R}^n, \text{ the initial point, with } x_i^0 / \delta_i^{\min} \in \mathbb{Z} \text{ for each } i \in \mathcal{G} \]

\[\Delta^0, \text{ the initial poll size vector where } \Delta_i^0 \text{ is the value } a_i^0 \times (10)^{b_i^0} \text{ in } \mathcal{P} \]

\[\text{which is the closest to } \alpha_i^0 \text{ from Equation (7)} \]

\[\text{for iteration } k = 0, 1, 2, \ldots \text{ do} \]

- Let \( \delta^k \) be the mesh size vector from Equation (11)
- Set \( \rho^k := \text{diag}(\delta^k)^{-1} \Delta^k \in \mathbb{N}^n \), the ratio vector
- Search step: Test finitely many trial points on the mesh \( M^k \) (5)
- Poll step: Test \( P^k := \{ x^k + \text{diag}(\delta^k) d : d \in D^k \} \subset M^k \) where \( D^k \in \mathbb{Z}^{n \times n} \) is a positive spanning set satisfying \(-\rho^k \leq d \leq \rho^k \) 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 \( \Delta^{k+1} \) as in Equation (12)
- else
  - set \( x^{k+1} = x^k \) and \( \Delta^{k+1} \) as in Equation (10)
3.4 Convergence of MADS with minimal granularity and controlled decimals

The algorithmic modifications of Algorithm 5 versus the MADS algorithm from [15] correspond to the way that the poll and mesh size parameters are updated from one iteration to another. The update rules were similar to those proposed for the generalized pattern search (GPS) algorithm of Torczon [66], in which both the mesh and poll size parameters were identical (in fact, there was 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 these algorithms, these parameters were updated by multiplying them by an integer power of a rational number $\tau$. In [15], the parameters $\delta^k$ and $\Delta^k$ are replaced by the vectors $\delta^k$ and $\Delta^k$, as in the present work, but again, each component of these vectors is an integer power of $\tau$. The key element in the convergence analysis was 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]\(^2\). The proof of this result is far from trivial, as it uses the hypothesis that the number $\tau$ 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 following 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 $k = 0, 1, \ldots, p - 1$.

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

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

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

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

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

\(^2\)This paper won the 1999 SIAM outstanding paper prize.
and setting $w^k = \frac{1}{\delta_{\min}} \text{diag}(\delta^k) z^k$ we get

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

and $w_i^k = \frac{\delta_i^k}{\delta_{\min}} z_i^k$ is integer because the value $\frac{\delta_i^k}{\delta_{\min}}$ 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 ones shown in [2], in which the results 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 $\{x^k\}_{k \in \mathcal{K}}$ be a refining subsequence converging to the refined point $x^*$. Then $x^*$ is the limit of a sequence of mesh local optimizers $\{x^k\}_{k \in \mathcal{K}}$ on meshes that get infinitely fine with respect to the continuous variables and on the finest mesh with respect to the granular variables:

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

The above result may be strengthened by assuming Lipschitz continuity with respect to the continuous variables. For a given $x^*$, we denote $f_{x^*}(y) = f(x)$ 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 $\{x^k\}_{k \in \mathcal{K}}$ be a refining subsequence converging to the refined point $x^* \in \mathbb{R}^n$, and let $\mathcal{D} = \left\{ \frac{d}{\|d\|} : d \in \mathcal{D}^k, k \in \mathcal{K} \right\}$ 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 denote $\Omega_{x^*} = \{x \in \Omega : x_i = x_i^* \forall i \in \mathcal{G}\}$ to be the restriction of $\Omega$ 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 $\mathcal{D} = \{d/\|d\| : d \in D^k, k \in K\}$ 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

$$f_{x^*}^0(x^*; d) \geq 0 \quad \forall d \in T^H_{\Omega x^*}(x^*) .$$

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 Clarke or contingent 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), 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 test cases is limited.

**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 pseudo-random 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.

In this work, the relative performance of algorithms is assessed by data profiles [53], which require 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

$$\bar{f}_{\text{feas}} - f(x_e) \geq (1 - \tau) \left( \bar{f}_{\text{feas}} - f(x^b) \right)$$

where the reference value $\bar{f}_{\text{feas}}$ 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 is failed. For computational problems without constraints, $\bar{f}_{\text{feas}} = f(x_0)$, 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 $\tau$, versus the number of groups of $n + 1$ evaluations.

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

### 4.1 Algorithms and solvers

Computational results using the MADS algorithm are generated using the version 3.9.0 of the NOMAD [39] software package, freely available at [www.gerad.ca/nomad](http://www.gerad.ca/nomad).

The MADS algorithm with $n+1$ poll directions [13] with or without the use of quadratic models, as in [24], is considered.

Two variants are used. First, “GMesh”, corresponds to MADS with the new granular mesh. Then, 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 [15], where the mesh size is multiplied or divided by an integer power of a rational number $\tau$, 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], and 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 default values of the anisotropy trigger parameter $a_t$ (see Eq. 12). The characteristics and sources of these problems are summarized in Table 1. The number of variables ranges from $n = 2$ to 20; 19 problems having constraints ($m > 0$) other than bound constraints.

In all the figures and tables below, the new algorithm introduced in this paper is called GMesh and the previous MADS algorithm is called XMesh. We use 10 run instances for
For each problem, with different random seeds. Results with different values of the parameter $a_t$ are presented in Figure 2, with precisions $\tau \in \{10^{-3}, 10^{-5}, 10^{-7}\}$, with and without the use of quadratic models.

The data profiles reveal that the effect of the 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.
Figure 2: 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 without quadratic models (top) and with quadratic models enabled (bottom).

remaining of the 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 1 indicates problems that can be modified to include integers). The modification 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 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 to compare the performance of GMesh and XMesh on 10 run instances per algorithm and problem.

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>n</th>
<th>cont.</th>
<th>int.</th>
<th>bin.</th>
<th>Bnds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISO1</td>
<td>[55]</td>
<td>12</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO2</td>
<td>[55]</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO4</td>
<td>[55]</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO6</td>
<td>[55]</td>
<td>15</td>
<td>9</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO7</td>
<td>[55]</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO8</td>
<td>[55]</td>
<td>15</td>
<td>5</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MISO9</td>
<td>[55]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I1</td>
<td>[57]</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>SO-I2</td>
<td>[57]</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I3</td>
<td>[57]</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>SO-I4</td>
<td>[57]</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>SO-I5</td>
<td>[57]</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>SO-I6</td>
<td>[57]</td>
<td>13</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>SO-I7</td>
<td>[57]</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I8</td>
<td>[57]</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td>8</td>
<td>no</td>
</tr>
<tr>
<td>SO-I9</td>
<td>[57]</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I10</td>
<td>[57]</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I11</td>
<td>[57]</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>SO-I12</td>
<td>[57]</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>SO-I13</td>
<td>[57]</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SO-I14</td>
<td>[57]</td>
<td>13</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>SO-I15</td>
<td>[57]</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Description of selected discrete problems from [55, 56, 57].

Figure 3 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.

4.4 Comparison with other solvers on discrete problems

A subset of MISO, SO-I and SO-MI problems from Table 2 is also used to compare the performance of GMesh, XMesh, versus the DFL, MISO, and BFO solvers.

The MISO code cannot solve directly 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 with a single initial point from the MISO and SO-MI sets are available.

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 pseudo-random number generator). Hence, a single run instance is performed per problem but 10 different starting points are considered. The total is 120 computational problems.

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 sur-
rogate 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) have been considered. Preliminary experiments have shown that the sampling strategy in the MISO solver allows to perform a better exploration of the design space than what is obtained with the default settings of NOMAD. For a fair comparison, the optional VNS search [6] available for XMesh and GMesh to escape local optima has been enabled on these problems. The DFL and BFO solvers have no equivalent option available.

For DFL, multiple evaluations of points are not accounted for in the statistics.

For the MISO solver, the execution time to create new points tend to increase significantly with the number of already evaluated points (see Table 3). To obtain a practical overall execution time for MISO, we have considered a maximum of 150 × n blackbox evaluations with a limit of 2,000 evaluations. This behavior of the MISO solver is due to the cost of constructing surrogates, that grows rapidly with the number of evaluated points. It highlights the fact that MISO should rather be used, as intended by its author, only with costly blackboxes.

The BFO solver default settings have been considered. The only indicated parameters corresponded to the problem definition.

Figure 4 presents the data profiles comparing the two versions of MADS with DFL, MISO, and BFO. DFL is usually fast to find a good solution, but then stops without using the whole evaluation budget. It indicates the lack of mechanism to escape local optima. If given the time, eventually the other solvers will find better points. MISO performs well in terms of number of function evaluations, but not in terms of elapsed time. GMesh is slightly better than XMesh, and MADS in general compares well. BFO is always outperformed.
Table 3: Optimization times for 3 selected problems with $n \in \{5, 10, 15\}$ from a single starting point with evaluation budget of $150 \times n$ and a limit of 2,000 evaluations for MISO. Runs stopping before reaching the maximum evaluation budget are tagged with the ‘*’ symbol.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Name</th>
<th>GMesh</th>
<th>XMesh</th>
<th>DFL</th>
<th>MISO</th>
<th>BFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>MISO4</td>
<td>6</td>
<td>5</td>
<td>1*</td>
<td>69</td>
<td>0.1*</td>
</tr>
<tr>
<td>10</td>
<td>SOMI11</td>
<td>16</td>
<td>16</td>
<td>3*</td>
<td>298</td>
<td>0.3</td>
</tr>
<tr>
<td>15</td>
<td>MISO6</td>
<td>58</td>
<td>57</td>
<td>8*</td>
<td>2,168</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Figure 4: Data profiles obtained with convergence tolerance $\tau = 10^{-3}$, $\tau = 10^{-5}$ and $\tau = 10^{-7}$ on 12 continuous, integer and binary analytical problems without constraints (see Table 2) from 10 starting points, with MADS, DFL, MISO, and BFO.

4.5 Tuning trust-region parameters

This section presents computational results on the parameter-tuning optimization problem outlined in Section 1.1. The blackbox problem is taken from [16], 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 TR with parameter values $p \in \Omega$ (the superscript stands for the year that 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 [16], the maximum number of iterations was set to 1,000. For the classical TR parameters
The objective function value was $f^{2006}(p^c) = 13,641$ 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 [16] 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 and 9 problems for $p^*$.

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

Table 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 (8) problems accounting for 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.

The computational experiments of [16] were conducted more than 10 years ago. Now, with a maximum number of iterations of 1,000, on an Intel(R) Core(TM) i7 CPU 3.40GHz 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 4 processors (6 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 have increased the limit to 9999 iterations and set a 180 seconds CPU time limit 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 trust-region (TR) algorithm but the measure is more suited to the current computers capability. Again, If $x \notin \Omega$ then $f^{2018}(p) = +\infty$. We now obtain $f^{2018}(p^c)=1,008$ seconds and there are 2 problems that do not converge within the time limit (none reached the iteration limit) on an Intel(R) Core(TM) i7 CPU 3.40GHz computer. With $p^*$, 2 problems reaches the maximum CPU time limit and $f^{2018}(p^*) = 1,024$ seconds. Table 4.5 breaks down the computational times.
of the 2006 and 2018 versions, for both points $p^c$ and $p^*$.

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 value for $f^{2018}(p^c)$ on several consecutive runs ranges between 1,001 and 1,099 seconds. Hence, the RobustMADS algorithm [14] 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. Each optimization run requires approximately one day.

Controlling the mesh granularity is straightforward when using MADS with GMesh as stated in Section 3, with the $\delta_{\min}$ parameter.

Table 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 6. Note that the optimized solutions are equivalent in terms of quality, but that the ones provided by GMesh as much more elegant and simpler to report, thereby increasing the chances that these values get actually used in practice.

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Solution</th>
<th>$f^{2018}$</th>
<th>Improv. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>$p^c = (0.25, 0.75, 0.5, 2)$</td>
<td>1,008.0</td>
<td>0</td>
</tr>
<tr>
<td>XMesh</td>
<td>$p^c = (0.2939819787, 0.979406601, 0.4716387306, 1.47417761)$</td>
<td>733.6</td>
<td>27</td>
</tr>
<tr>
<td>GMesh no $\delta_{\min}$</td>
<td>$p^{00} = (0.672010424, 0.685829734, 0.061485394, 1.34816385)$</td>
<td>727.0</td>
<td>28</td>
</tr>
<tr>
<td>GMesh $\delta_{\min} = 0.005$</td>
<td>$p^{005} = (0.845, 0.99, 0.485, 1.575)$</td>
<td>697.6</td>
<td>31</td>
</tr>
<tr>
<td>GMesh $\delta_{\min} = 0.01$</td>
<td>$p^{01} = (0.74, 0.99, 0.17, 1.34)$</td>
<td>688.7</td>
<td>32</td>
</tr>
<tr>
<td>GMesh $\delta_{\min} = 0.05$</td>
<td>$p^{05} = (0.2, 0.9, 0.2, 1.3)$</td>
<td>768.4</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 5: Solutions obtained for the TR parameter tuning problem with different algorithms. The last column indicates the relative improvement over $p^c$.

5 Discussion

This works 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 that the new MADS variant (GMesh) is slightly better than the previous variant (XMesh) on both continuous and discrete problems, and that it is competitive, if not better, than the state-of-the-art DFO solvers DFL, MISO, and BFO.
Table 6: Detailed computational time of the objective function value $f^{2018}(p^c)$ 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 8 problems accounting for 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.

The new strategy is compatible with all the extensions of MADS: The biobjective version of the algorithm [17], the extreme [9] and progressive barrier [10] approaches for handling constraints, the parallel space decomposition technique [11], the recent dynamic scaling of [15], and the robust version of the algorithm [14], for instance.

References


[63] K. Socha. ACO for Continuous and Mixed-Variable Optimization. In M. Dorigo, M. Birattari, C.Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, Ant Colony Optimization and


