Mathematical Programming Computation (2020) 12:673–702
https://doi.org/10.1007/s12532-020-00182-7

FULL LENGTH PAPER

An algorithmic framework based on primitive directions and nonmonotone line searches for black-box optimization problems with integer variables

Giampaolo Liuzzi[1] · Stefano Lucidi[2] · Francesco Rinaldi[3]

Received: 2 February 2018 / Accepted: 6 February 2020 / Published online: 23 February 2020

Abstract
In this paper, we develop a new algorithmic framework to solve black-box problems with integer variables. The strategy included in the framework makes use of specific search directions (so called primitive directions) and a suitably developed nonmonotone line search, thus guaranteeing a high level of freedom when exploring the integer lattice. First, we describe and analyze a version of the algorithm that tackles problems with only bound constraints on the variables. Then, we combine it with a penalty approach in order to solve problems with simulation constraints. In both cases we prove finite convergence to a suitably defined local minimum of the problem. We report extensive numerical experiments based on a test bed of both bound-constrained and generally-constrained problems. We show the effectiveness of the method when compared to other state-of-the-art solvers for black-box integer optimization.

Keywords Derivative free optimization · Black box problems · Integer variables · Nonmonotone line search

Mathematics Subject Classification 90C56 · 90C10 · 90C30

✉ Giampaolo Liuzzi
giampaolo.liuzzi@iasi.cnr.it

Stefano Lucidi
lucidi@diag.uniroma1.it

Francesco Rinaldi
rinaldi@math.unipd.it

[1] Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", Consiglio Nazionale delle Ricerche, Via dei Taurini 19, 00185 Rome, Italy

[2] Dipartimento di Ingegneria Informatica Automatica e Gestionale "A. Ruberti", "Sapienza" Università di Roma, Via Ariosto 25, 00185 Rome, Italy

[3] Dipartimento di Matematica "Tullio Levi-Civita", Università di Padova, Via Trieste, 63, 35121 Padua, Italy

1 Introduction

Many real-world problems coming from different fields (e.g., engineering, economics, biology) can be modeled using black-box functions. Those functions represent the experimentally obtained behavior of a system and in practice are given by means of specific simulation tools. Hence no internal or analytical knowledge for the functions is provided. Furthermore, evaluating the function at a given point is usually (or might be) an expensive task in terms of computational resources, and only a limited budget of evaluations is available for the optimization. On top of that, real systems are often described by means of unrelaxable integer variables (e.g., number of nurses in a ward, number of coils in a magnet, and so on) that need to be properly handled. When dealing with this kind of problems, it is neither possible to cut away part of the feasible set nor to get an optimality gap for the solutions (like the exact algorithmic frameworks in integer programming usually do). Certifying global optimality is exceptionally difficult in the black-box setting, even if we assume convexity for the given problem [19].

In general, when exact methods (i.e., methods that can certify global optimality) cannot be applied, heuristic ones are considered. *Explorative search methods* (e.g., variable neighborhood search [28], iterated local search [25]), which look for a new solution in some neighborhood of a given point and perturb the point and/or the neighborhood when specific conditions are met. On the other hand, *population based methods* (e.g., genetic algorithms [11,16]), which use a set (i.e., a population) of points at each iteration to generate new solutions, are classic heuristic approaches for problems with integer variables (see, e.g., [8,15] for further details). However, those two classes of approaches may not be suitable for optimization problems with computationally expensive objective and constraint functions because they often require a large number of evaluations to find good solutions.

So, our goal is to develop a method that explores the integer lattice and reduces as much as possible the objective function value (while satisfying the constraints) without consuming too many function evaluations.

In the paper we then consider the following optimization problem:

$$\begin{aligned} &\min f(x) \\ &s.t.\ x \in \mathcal{C} \cap \mathbb{Z}^n, \end{aligned} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a black-box function (i.e., the mathematical representation of $f(x)$ is *not available*) and $\mathcal{C} \subset \mathbb{R}^n$ is a given set (whose description usually includes simple bound constraints on the variables and general simulation constraints). In particular, for Problem (1), we require that

(i) the feasible set $\mathcal{C} \cap \mathbb{Z}^n$ is not empty and contains a finite number of points;
(ii) the objective function f is well-defined for every feasible point $x \in \mathcal{C} \cap \mathbb{Z}^n$.

According to the taxonomy of constraints given in [21], if there are simulation constraints defining set $\mathcal{C}$, we assume that they belong to the class QRSK, i.e., that they are Quantifiable, Relaxable, Simulation, Known. We also assume that

– the objective function and the simulation constraints (if any) are not defined when $x \notin \mathbb{Z}^n$, i.e., the integrality constraint is not relaxable;

– the black-box functions are computationally expensive in this framework.

Such problems can be solved by means of the so-called "derivative-free" methods [6,9].

Direct search methods are simple and intuitive derivative-free approaches. The algorithms in this class suitably sample, at each iteration, the objective function over a specific set of points belonging to the integer lattice and choose the best point (in terms of objective function value) as the new iterate. In the sampling phase, they can either use a *stencil* (i.e., a predetermined set of search directions with a common fixed stepsize, see, e.g., [1,3,5,26]) or a *skewed stencil* (i.e., a predetermined set of search directions with dynamically changing stepsizes, see, e.g., [22,23]) whose shape is modified depending on a line-search procedure. More specifically, in [5] the authors propose a Generalized Pattern Search (GPS) algorithm for mixed variable programming problems with bound constraints. A filter GPS approach for nonlinear constrained problems with discrete variables is analyzed in [3]. The Mesh Adaptive Direct Search (MADS) algorithm has been extended to handle categorical variables in [1]. The recent paper [7] introduces a modification to the MADS algorithm for problems with granular variables, i.e., variables with a controlled number of decimals. This modification includes a new updating of the underlying mesh that progressively increases the precision. An algorithmic framework combining in a suitable way a line search with respect to the continuous variables and a simple (stencil based) local search with respect to the discrete variables is described in [26]. In [22], a specific line-search approach is used to handle bound-constrained problems with integer variables. In [23], the authors extend the line-search algorithm to generally-constrained black-box problems with integer variables by means of a sequential penalty approach.

Recently, a more sophisticated way to handle integer variables in the bound-constrained case was proposed in [34]. The algorithm therein, called BFO, implements a mesh based direct search that combines fixed stencils with tree-based searches.

Model based methods represent another important class of derivative-free approaches. The algorithms belonging to this class build a model of the objective function at each iteration and select the point minimizing the model as the new iterate in case a decrease of the original objective function is guaranteed. Surrogate or quadratic models are commonly used for this class of methods (see, e.g., [10,30–33]). In [10], the authors describe the method implemented in the library RBFOpt. The algorithm builds and iteratively refines a surrogate model (based on radial basis functions) of the unknown objective function.

Surrogate models are also used in [30–32]. In [32], the authors define a surrogate based algorithm, called SO-I, for solving purely integer optimization problems that have computationally expensive black-box objective functions and that may have computationally expensive constraints. Another model-based algorithmic framework for mixed-integer problems is described in [31]. In [30], the authors describe MISO, an algorithm for mixed-integer problems with bound constraints that allows the choice of various surrogate models and sampling strategies. The use of quadratic models in a trust region method for mixed-integer problems is analyzed in [33]. A different model-based approach is described in [19] for the global optimization of black-box convex integer problems. The authors consider an underestimator that does not require

access to (sub)gradients of the objective but, rather, uses secant linear functions that interpolate the objective function at previously evaluated points.

If we focus on direct search methods, we can easily see that the search of points that monotonically reduce the objective function, together with the use of a predetermined set of directions for exploring the integer lattice can cause the algorithm to get stuck in points that cannot be further improved (and this might happen pretty soon in the optimization process). Even the use of a line search that dynamically changes the skewed stencil might not be enough in order to escape those points. Indeed, it is possible to build examples where both strategies cannot move away from the starting point.

In this paper, we define a new algorithmic framework of the direct search type that tries to overcome this issue. The three main features that characterize our strategy are the following:

- a stencil enrichment by means of a suitable set of search directions (the so called *primitive directions*) as soon as the algorithm gets stuck;
- a nonmonotone version of the line search defined in [22,23] for getting more freedom when changing the shape of the stencil;
- a simple penalty approach, similar to the one described in [23], for handling the case when set $\mathcal{C}$ is described by means of black-box functions.

The use of a nonmonotone acceptance rule is very important in this context since it improves the performance of our framework. Indeed, trying to get a new point that strictly reduces the objective function, like we do when using a monotone line search, might either get small movements along the search direction (especially when dealing with an objective function with steep sided valleys) or require the generation of many primitive directions in order to escape a point (this usually happens when the objective function is locally "flat").

Nonmonotone acceptance rules have already been used in derivative-free continuous optimization (see, e.g., [12,14,17]). Anyway, to the best of our knowledge, this is the first time that a nonmonotone discrete line search is embedded into an algorithmic framework that handles black-box problems with integer variables.

About the theoretical results reported in the paper, we prove finite convergence of the algorithmic framework to a suitably defined local minimum of Problem (1) when $\mathcal{C}$ is:

- a set defined by simple bounds on the variables;
- a more general set described by black-box functions (in this case, to prove convergence, we need to assume that some constraint qualification condition holds for the problem).

An extensive numerical analysis on a large testbed of both bound-constrained and generally-constrained problems is also reported. More specifically, we first investigate the effects of using enriched stencils and a nonmonotone acceptance rule in our algorithmic framework. Then, we show the effectiveness of our integer lattice exploration strategy when compared to the strategies embedded into three direct search methods, namely NOMAD (v.3.8.1) [2,20], BFO [34] and DFL [22,23]. In order to understand if our simple direct search method is competitive with algorithms that use models,

we further include in the analysis the comparison with the model-based version of NOMAD and with MISO [30].

The paper is organized as follows. In Sect. 2, we describe the algorithmic framework for black-box problems with bound constraints. Then, in Sect. 3, we explain how to handle problems with general simulation constraints. We hence report numerical experiments both for problems with bound constraints and simulation constraints, respectively in Sects. 4 and 5. Finally, we draw some conclusions in Sect. 6.

Now, we report some definitions that will be useful in the next sections. First of all, we introduce the important concept of *stencil*, a set of directions used to determine the points where a function is sampled.

Definition 1 *(Stencil)* Given a point $x \in \mathbb{R}^n$, a stepsize $\alpha > 0$ and p distinct directions $d_i \in \mathbb{R}^n$, $i = 1, \ldots, p$, a stencil is the following set of points:

$$\mathcal{S}(x, \alpha, d_1, \ldots, d_p) = \{x \pm \alpha\, d_i, \quad i = 1, \ldots, p\}.$$

A particular stencil is, for instance, the *coordinate* stencil, that is

$$\mathcal{S}(x, \alpha, e_1, \ldots, e_n) = \{x \pm \alpha e_i, \quad i = 1, \ldots, n\},$$

where $e_i \in \mathbb{R}^n$ is the i-th column of the identity matrix. Next, we introduce the definition of *skewed stencil* which is obtained from Definition 1 by allowing different stepsizes for each direction.

Definition 2 *(Skewed stencil)* Given a point $x \in \mathbb{R}^n$, p stepsizes $\alpha_i > 0$ and p distinct directions $d_i \in \mathbb{R}^n$, $i = 1, \ldots, p$, a skewed stencil is the following set of points:

$$\mathcal{S}(x, \alpha_1, \ldots, \alpha_p, d_1, \ldots, d_p) = \{x \pm \alpha_i\, d_i, \quad i = 1, \ldots, p\}.$$

Now, we formally introduce the basic concept of *divisor*.

Definition 3 *(Divisor)* Given two integers a and b, we say that a is a divisor of b, and write $a|b$, if an integer c exists such that $b = ca$.

When $a|b$, we also say that a divides (or is a factor of) b, or that b is a multiple of a. Let $v \in \mathbb{Z}^n$. We call $d \in \mathbb{Z}$ a common divisor of $v_1, \ldots, v_n$ if $d|v_i$, with $i = 1, \ldots, n$. Then, the *greatest common divisor* of $v_1, \ldots, v_n$, denoted as $GCD(v_1, \ldots, v_n)$, is a (positive) common divisor such that all other common divisors of $v_1, \ldots, v_n$ divide d.

Now, we give a few definitions that will be useful when describing in depth our algorithm. We start by introducing the concept of primitive vector:

Definition 4 *(Primitive vector)* A vector $v \in \mathbb{Z}^n$ is called primitive if $GCD(v_1, \ldots, v_n) = 1$.

Remark 1 Given any two points $x, y \in \mathbb{Z}^n$, such that $x \neq y$, we have $x - y = \alpha d$, with $d \in \mathbb{Z}^n$ a primitive vector and $\alpha \in \mathbb{N}$. Hence, starting from a point $x \in \mathbb{Z}^n$ any other point $y \in \mathbb{Z}^n$ can be reached by choosing a suitable stepsize $\alpha \in \mathbb{N}$ along a specific primitive direction $d \in \mathbb{Z}^n$. □

Then, we formally define the concept of feasible primitive direction, which represents an important feature in our framework:

Definition 5 *(Feasible primitive direction)* Given a point $\bar{x} \in \mathcal{C} \cap \mathbb{Z}^n$, a primitive direction d is feasible at $\bar{x}$ for $\mathcal{C}$ when $\beta \in \mathbb{N}$ exists such that

$$\bar{x} + \alpha d \in \mathcal{C} \cap \mathbb{Z}^n, \quad \text{for all } \alpha \leq \beta, \quad \alpha \in \mathbb{N}.$$

We further denote with $D(\bar{x})$ the set of all feasible primitive directions at a given point $\bar{x}$.

Definition 6 *(Discrete neighborhood)* Given a point $\bar{x} \in \mathcal{C} \cap \mathbb{Z}^n$ and a parameter $\beta \in \mathbb{N}$, the discrete neighborhood of $\bar{x}$ is

$$\mathcal{N}(\bar{x}, \beta) = \{x \in \mathcal{C} \cap \mathbb{Z}^n : x = \bar{x} + \alpha d, \text{ with } \alpha \leq \beta, \ \alpha \in \mathbb{N} \text{ and } d \in D(\bar{x})\}.$$

Remark 2 Note that $\bar{x} \notin \mathcal{N}(\bar{x}, \beta)$. Furthermore, recalling Remark 1, the discrete neighborhood $\mathcal{N}(\bar{x}, \beta)$ can coincide with $(\mathcal{C} \cap \mathbb{Z}^n) \backslash \{\bar{x}\}$, provided that the parameter β is chosen sufficiently large and that every primitive vector d is feasible at $\bar{x}$, i.e., $d \in D(\bar{x})$.

An example of discrete neighborhood is given in Fig. 1. Obviously, the concept of discrete neighborhood is only ideal. Indeed, building up such a neighborhood is an expensive task that cannot be efficiently done in practice. This is the reason why we need to replace $D(\bar{x})$ in the definition above with a suitably chosen subset of directions, thus getting the following definition.

Definition 7 *(Weak discrete neighborhood)* Given a point $\bar{x} \in \mathcal{C} \cap \mathbb{Z}^n$, a parameter $\beta \in \mathbb{N}$ and a subset $D \subset D(\bar{x})$, the weak discrete neighborhood of $\bar{x}$ is

$$\mathcal{N}_w(\bar{x}, \beta, D) = \{x \in \mathcal{C} \cap \mathbb{Z}^n : x = \bar{x} + \alpha d, \text{ with } \alpha \leq \beta, \ \alpha \in \mathbb{N} \text{ and } d \in D\}.$$

Now, we can formally give the definition of a local minimum for Problem (1).

Definition 8 (β-local minimum point) Given $\beta \in \mathbb{N}$, a point $x^* \in \mathcal{C} \cap \mathbb{Z}^n$ is a β-local minimum of Problem (1), if

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}(x^*, \beta). \tag{2}$$

Following the same path as before, we get another definition that makes more sense from a practical point of view.

Definition 9 (Weak β-local minimum point) Given $\beta \in \mathbb{N}$, a point $x^* \in \mathcal{C} \cap \mathbb{Z}^n$ is a weak β-local minimum of Problem (1), if a subset $D \subset D(x^*)$ exist such that

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}_w(x^*, \beta, D). \tag{3}$$

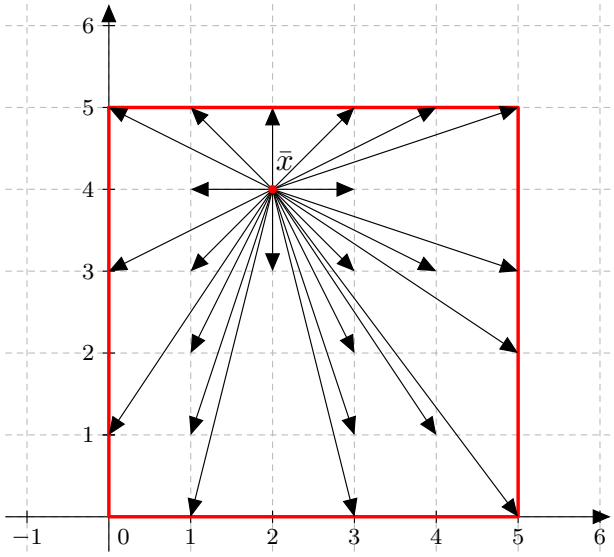

Fig. 1 Discrete neighborhood: example of the discrete neighborhood $\mathcal{N}(\bar{x}, \beta)$ with $\bar{x} = (2,\ 4)^\top$, $\beta = 1$, $\mathcal{C} = \{x \in \mathbb{R}^2 :\ 0 \leq x_1 \leq 5,\ 0 \leq x_2 \leq 5\}$. $\mathcal{N}(\bar{x}, \beta)$ consists of all the points within the red square (boundary of $\mathcal{C}$) for which there is an arrow head

2 A nonmonotone algorithm for black-box optimization problems with bound constraints

In the first part of the paper, we focus on bound-constrained integer programming problems of the following form:

$$\begin{aligned} &\min f(x) \\ &s.t.\ l_i \leq x_i \leq u_i, \qquad i = 1, \ldots, n \\ &\quad x \in \mathbb{Z}^n. \end{aligned} \tag{4}$$

Note that Problem (4) is a particularization of Problem (1), namely when we consider $\mathcal{C} = \{x \in \mathbb{R}^n :\ l_i \leq x_i \leq u_i,\ i = 1, \ldots, n\}$. In this case, we assume $l_i, u_i \in \mathbb{Z}$ and $-\infty < l_i < u_i < \infty$, for all $i = 1, \ldots, n$, so that $\mathcal{C} \cap \mathbb{Z}^n$ contains, obviously, a limited number of points. Also note that the objective function f is defined for every feasible point x. From now on, let us denote the set of bound constraints as follows

$$X = \{x \in \mathbb{R}^n : l_i \leq x_i \leq u_i,\ i = 1, \ldots, n\}.$$

In this section, we describe a new algorithm for black-box optimization problems with bound constraints. It basically combines the use of primitive feasible directions with a suitable nonmonotone line search that only explores points in the integer lattice. The detailed scheme of the algorithm is reported below (see Algorithm 1). As we can easily see, there are three different phases at each iteration. In the first phase (**Steps 3–14**), we try to improve the current iterate x_k. More specifically, we consider the

incumbent solution y, i.e., a vector that represents the latest point accepted in Phase 1, and set it equal to x_k at **Step 3**. Then, we select a direction from a specific set $D \subseteq D(x_k)$ at **Step 5** and explore this direction at **Step 6** by means of a nonmonotone line search in order to find a new point that both guarantees a sufficiently large movement along the search direction and a reduction with respect to the current reference value f^{ref}. The reference value f^{ref} is the maximum among the last M objective function values accepted by the algorithm. Those values are stored in the set W which is managed like a queue (that is according to a first-in-first-out policy). Hence, procedure pop(W) extracts from W the oldest function value, whereas procedure push(f, W) inserts into W the function value f. Phase 1 ends as soon as we either find such an improving point or D becomes the empty set (i.e., all directions in D have been explored, but no reduction has been obtained with respect to the reference value). We note that the iterate x_k, the directions $d \in D_k$ and the start stepsizes $\tilde{\alpha}_k^{(d)}$ might be seen as a dynamically changing skewed stencil which is suitably modified at each iteration of the algorithm depending on the outcomes of the line search. If the line search failed (i.e., $\alpha = 0$), then we suitably reduce the stepsize at **Step 7**. Otherwise, the line search gets a point guaranteeing a reduction with respect to the reference value, and the algorithm performs the following operations:

- **Step 9**: it updates the incumbent solution y and the start stepsize related to the last direction explored (i.e., it enlarges the stencil with respect to the direction);
- **Step 10**: it updates (if necessary) the point x_{min}, i.e., the best point obtained so far;
- **Steps 11–12**: it updates the set W (i.e., the queue needed to compute the reference value in the next iteration) and the reference value f^{ref}.

When Phase 1 is over, Phase 2 (**Steps 15–28**) starts. In Phase 2, the sets of search directions D (those to be used in Phase 1) and D_k (those generated so far) are possibly updated. More specifically, if Phase 1 of the algorithm succeeded in modifying the incumbent solution y (i.e., $y \neq x_k$ and condition at **Step 15** is not satisfied), D_k stays the same and D is set equal to D_k (see **Step 27**). Otherwise, **Step 16** is executed and the algorithm checks whether the *Nonmonotone line search* failed along all the directions in D_k and if the start stepsize $\tilde{\alpha}_k^{(d)}$ is 1 for all of those directions (i.e., the dynamically changing skewed stencil defined by directions $d \in D_k$ and stepsizes $\tilde{\alpha}_k^{(d)}$ shrank to its minimum size in Phase 1). If this is not the case, D_k stays the same and D includes only the directions that failed with stepsize greater than one (see **Step 24**). Otherwise, the algorithm tries to enrich D_k (by performing **Steps 17–22**). If D_k equals the set $D(x_k)$ of primitive feasible directions in x_k, then the algorithm checks whether x_k is the point with the best objective function value. If this is the case, it stops the execution returning the point x^*, i.e., the best point found, otherwise it moves to the best point found so far (see **Step 19**), thus D and D_k are set equal to $D(x_k)$. If the set of generated search directions is still smaller than $D(x_k)$, it is enriched ($D_{k+1} \supset D_k$) and the set D includes only the new directions generated to enrich D_k (see **Step 21**). Finally, in Phase 3 the new iterate x_{k+1} is generated.

In order to improve the performance of the algorithm, it is possible to reuse the direction d in case a failure is obtained by the nonmonotone line search in Phase 1. In particular, if the line search at **Step 6** fails along the search direction d, we might sim-

ply consider direction $\bar{d} = -d$ and perform another search along $\bar{d}$ before selecting another direction from D. We actually included this option in our practical implementation, but we decided to report the simplified version of the scheme here. This choice allows to improve readability avoiding that the algorithm scheme becomes excessively cumbersome.

Algorithm 1 NonMonotone Black-Box Optimization Algorithm (NM-BBOA)

Data. $x_0 \in X \cap \mathbb{Z}^n$, $D = D_0 \subset D(x_0)$ a set of initial directions, $\tilde{\alpha}_0^{(d)} = 1$, for each $d \in D_0$. $W = \{f(x_0)\}$, $f^{ref} = \bar{f}_0 = f(x_0)$, $x_{min} = x_0$, $M \geq 1$, $M \in \mathbb{N}$.

For $k = 0, 1, \ldots$

PHASE 1 - Explore points around x_k

Set $y = x_k$

While $D \neq \emptyset$ and $y = x_k$ **do**

Select and explore direction

Choose $d \in D$ set $D = D \setminus \{d\}$

Compute α by the *Nonmonotone Line Search*$(\tilde{\alpha}_k^{(d)}, x_k, d, f^{ref}; \alpha)$

Update start stepsize, reference value and queue

If $\alpha = 0$ **then** set $\tilde{\alpha}_{k+1}^{(d)} = \max\{1, \lfloor \tilde{\alpha}_k^{(d)}/2 \rfloor\}$

else

Set $y = x_k + \alpha d$ and $\tilde{\alpha}_{k+1}^{(d)} = \alpha$

If $f(y) < f(x_{min})$ **then** $x_{min} = y$

If $|W| = M$ **then** pop(W)

push$(f(y), W)$, $f^{ref} = \max_{f \in W}\{f\}$

End If

End While

PHASE 2 - Update set of search directions

If $y = x_k$ **then**

If *Nonmonotone Line Search* failed with $\tilde{\alpha}_k^{(d)} = 1$ for all $d \in D_k$ **then**

If $D_k = D(x_k)$ **then**

If $f(x_k) = f(x_{min})$ **then return** $x^* = x_k$

else Set $y = x_{min}$ and $D = D_{k+1} = D(x_k)$

else

Generate $D_{k+1} \supset D_k$, set $\tilde{\alpha}_{k+1}^{(d)} = 1$, for all $d \in D_{k+1}$ and $D = D_{k+1} \setminus D_k$

End If

else

Set $D_{k+1} = D_k$ and $D = \{d \in D_k$: *Nonm. Line Search* failed with $\tilde{\alpha}_k^{(d)} > 1\}$

End If

else

Set $D_{k+1} = D_k$ and $D = D_k$

End If

PHASE 3 - Update iterates

Set $x_{k+1} = y$, $\bar{f}_{k+1} = f^{ref}$

End For

The detailed scheme of the Nonmonotone Line Search procedure is reported in Algorithm 2. This procedure is one of the novelties of the proposed approach and is new in the context of black-box problems with integer variables. First, it calculates the start stepsize for the search. This is chosen as the minimum between $\tilde{\alpha}_k^{(d)}$ (defining the dynamically changing skewed stencil) and the largest stepsize $\bar{\alpha}$ that can be taken along the search direction d (See Initialization). If at **Step 1** the start stepsize is greater than zero and the function reduces along the search direction with respect to the reference value f^{ref}, the search starts expanding the stepsize and keeps doing it (by running **Steps 2–4**) until either the maximum stepsize is reached or a reduction with respect to the reference value cannot be guaranteed anymore (See **Step 3**). More specifically, we set, at **Step 2**, the trial stepsize β equal to the minimum between $\bar{\alpha}$ and 2α. At **Step 3**, we check if α is equal to the maximum stepsize $\bar{\alpha}$ or the objective function value in the trial point $x + \beta d$ cannot guarantee a reduction with respect to the reference value f^{ref}. If one of these two conditions is satisfied, then we set the start stepsize $\tilde{\alpha}_k^{(d)}$ related to the direction d equal to the stepsize α (thus modifying the skewed stencil) and return α (see **Step 5**). Otherwise, we set α equal to the trial stepsize β and go to **Step 2**, that is we keep expanding the trial stepsize because there is still hope to improve the function along the search direction.

We would like to note that the line search moves along the direction by always guaranteeing feasibility (the points chosen are on the integer lattice).

Algorithm 2 Nonmonotone Line Search

Input. $\tilde{\alpha}_k^{(d)}, x_k, d, f^{ref}$

Initialization. Compute the largest $\bar{\alpha}$ such that $x_k + \bar{\alpha} d \in X \cap \mathbb{Z}^n$. Set $\alpha = \min\{\bar{\alpha}, \tilde{\alpha}_k^{(d)}\}$.

Step 1. **If** $\alpha > 0$ and $f(x_k + \alpha d) < f^{ref}$ **then** go to Step 2
else Set $\alpha = 0$ and go to Step 5

Step 2. Let $\beta = \min\{\bar{\alpha}, 2\alpha\}$

Step 3. **If** $\alpha = \bar{\alpha}$ or $f(x_k + \beta d) \geq f^{ref}$ **then** set $\tilde{\alpha}_{k+1}^{(d)} = \alpha$ and go to Step 5

Step 4. Set $\alpha = \beta$ and go to Step 2

Step 5. Return α

In the following Theorem, we prove convergence of the method to a β-local minimum with $\beta = 1$.

Theorem 1 *Let $\{x_k\}$ and $\{\bar{f}_k\}$ be the sequences of solutions and of reference values, respectively, generated by NM-BBOA. Then, the algorithm cannot cycle and produces a β-local minimum point with $\beta = 1$.*

Proof First, we observe that the sequence $\{\bar{f}_k\}$ is bounded from below, since the number of distinct points in the feasible set is finite. Then, let us define

$$K = \{k : x_{k+1} \neq x_k\},$$

that is the subsequence of successful iterations given by NM-BBOA, and

$$H(k, \bar{k}) = \{h \in K : k < h \leq \bar{k}\}, \text{ for } \bar{k} > k,$$

which represents the set of successful iterations in between k and $\bar{k}$ (possibly including $\bar{k}$). Then, let $\bar{k}(M)$ be the index such that

$$|H(k, \bar{k}(M))| = M.$$

For each $k \in K$, we have that

$$f(x_{k+1}) < \bar{f}_k. \tag{5}$$

Moreover, from the updating rules of f^{ref} and the definition of $\bar{f}_k$, we have that

$$\bar{f}_{k+1} \leq \bar{f}_k. \tag{6}$$

Then, remembering that $|X \cap \mathbb{Z}^n| < \infty$, we can define

$$0 < \delta = \min_{x,y \in X \cap \mathbb{Z}^n} \left\{ |f(x) - f(y)| : f(x) \neq f(y) \right\}.$$

So that we have

$$\bar{f}_{\bar{k}(M)} < \bar{f}_k - \delta. \tag{7}$$

We prove now that NM-BBOA does not cycle. Suppose, in order to obtain a contradiction, that $\{x_k\}$ is an infinite sequence. If this is the case, it is easy to see that the set K is also infinite. Since the procedure does not terminate, a point $\tilde{x}$ (which is not a local minimum) is generated an infinite number of times. By (6) and (7), there exists an iteration $\tilde{k}$ such that

$$\bar{f}_{\tilde{k}} \leq f(\tilde{x}).$$

Furthermore, as $\tilde{x}$ is generated an infinite number of times, there exists an iteration $\hat{k} \geq \tilde{k}$ such that

$$f(\tilde{x}) < \bar{f}_{\hat{k}}.$$

Hence, we have

$$f(\tilde{x}) < \bar{f}_{\hat{k}} \leq \bar{f}_{\tilde{k}} \leq f(\tilde{x}),$$

which shows that the local-search procedure cannot cycle.

Finally, we prove that the point produced x^* is a β-local minimum of the problem (with $\beta = 1$). When NM-BBOA stops, let $\bar{k}$ be the last iteration index, so that $x^* = x_{\bar{k}}$ and $x_{\bar{k}} = x_{min}$. Furthermore, $D_{\bar{k}} = D(x^*)$ is the set of all the feasible and primitive directions at x^*. We thus have

$$f(x^*) \leq \bar{f}_{\bar{k}} \tag{8}$$

and, by **Step** 7 of Algorithm 1 and the instructions of Algorithm 2,

$$\bar{f}_{\bar{k}} \leq f(x^* + d) \quad \forall\, d \in D(x^*). \tag{9}$$

Then, combining Inequalities (8) and (9), we get that x^* is a β-local minimum with $\beta = 1$. □

Obviously, it is possible to develop a procedure that explores a larger discrete neighborhood (i.e., a neighborhood with $\beta > 1$). In order to do that, we just need to suitably modify the way we set the start stepsize when generating a new direction ($\tilde{\alpha}_{k+1}^{(d)} = \beta$ at **Step 21**) and we need to set to β the stepsizes that are equal to one when we get a successful iteration (if $\tilde{\alpha}_k^{(d)} = 1$ then set $\tilde{\alpha}_{k+1}^{(d)} = \beta$ at **Step 27**). Parameter β should anyway be carefully chosen in order to keep the exploration computationally cheap. It is easy to understand that such a choice becomes even more critical when getting a small budget of evaluations.

3 Handling of general simulation constraints

We now consider the following problem

$$\begin{aligned} &\min \ f(x) \\ &s.t. \ g(x) \leq 0 \\ &\quad x \in X \cap \mathbb{Z}^n, \end{aligned} \tag{10}$$

where $g : \mathbb{R}^n \to \mathbb{R}^m$ corresponds to $m \geq 1$ simulation constraint functions [21]. The above problem is Problem (1) where

$$\mathcal{C} = \{x \in \mathbb{R}^n : \ g(x) \leq 0, \ l \leq x \leq u\} = X \cap \mathcal{F}$$

and $\mathcal{F} = \{x \in \mathbb{R}^n : \ g(x) \leq 0\}$. In order to handle the nonlinear constraints, we use a simple penalty approach (see, e.g., [23]). Specifically, given Problem (10) and a penalty parameter $\epsilon \in \mathbb{R}_+$, we introduce the following penalty function:

$$P(x; \epsilon) = f(x) + \frac{1}{\epsilon} s(x),$$

where $s(x) = \sum_{i=1}^{m} \max\{0, g_i(x)\}$ and $\epsilon > 0$. Then, we consider the following bound-constrained problem

$$\begin{aligned} &\min \ P(x; \epsilon) \\ &s.t. \ x \in X \cap \mathbb{Z}^n \end{aligned} \tag{11}$$

Now, we prove the equivalence between the original Problem (10) and the penalized Problem (11). In particular, we will prove that there exists a threshold value $\bar{\epsilon}$ for the penalty parameter such that, for any $\epsilon \in (0, \bar{\epsilon})$, any minimum of the penalized problem is also a minimum of the original problem and viceversa.

Theorem 2 *Given Problem* (10) *and considering Problem* (11)*, a threshold value* $\bar{\epsilon} > 0$ *exists such that for every* $\epsilon \in (0, \bar{\epsilon})$*, any global minimum point* $\bar{x}$ *of* (11) *is also a global minimum of* (10) *and viceversa.*

Proof We first prove that any minimum point $\bar{x}$ of (11) is also a minimum of (10). Let us define the following penalty parameter

$$\bar{\epsilon} = \frac{a}{b}, \tag{12}$$

with

$$a = \min\{s(x)\colon x \in X \cap \mathbb{Z}^n,\, g(x) \not\leq 0\} \tag{13}$$

and

$$b = \max\{f(y) - f(x)\colon x, y \in X \cap \mathbb{Z}^n,\, g(x) \not\leq 0,\, g(y) \leq 0\}.$$

By contradiction, we assume that there exists a minimum point $\bar{x}$ of Problem (11),with $\epsilon < \bar{\epsilon}$, that is not feasible for Problem (10). For any feasible point y of Problem (10), we have:

$$f(y) - f(\bar{x}) \leq b = \frac{1}{\bar{\epsilon}} a < \frac{1}{\epsilon} s(\bar{x}),$$

with $\epsilon \in (0, \bar{\epsilon})$. Hence, we can write

$$f(y) < f(\bar{x}) + \frac{1}{\epsilon} s(\bar{x}),$$

thus contradicting the fact that $\bar{x}$ is minimum for Problem (11) with $\epsilon < \bar{\epsilon}$.

We now prove that any minimum point $\bar{x}$ of (10) is a minimum of (11) for any $\epsilon < \bar{\epsilon}$. For any point $x \in X \cap \mathbb{Z}^n$ not feasible for Problem (10) we have

$$f(\bar{x}) - f(x) \leq b = \frac{1}{\bar{\epsilon}} a < \frac{1}{\epsilon} s(x),$$

with $\epsilon \in (0, \bar{\epsilon})$. Hence, we can write

$$f(\bar{x}) < f(x) + \frac{1}{\epsilon} s(x),$$

and $\bar{x}$ is also minimum for Problem (11) for any $\epsilon < \bar{\epsilon}$. □

In order to prove that every local minimum of the penalized problem is also a local minimum of the original problem, we introduce the following assumption.

Assumption 1 For every $x \in X \cap \mathbb{Z}^n$ not feasible for Problem (10), i.e., $g(x) \not\leq 0$, there exists a direction $\bar{d} \in D(x)$ such that

$$s(x + \bar{d}) = \sum_{i=1}^{m} \max\{0, g_i(x + \bar{d})\} < \sum_{i=1}^{m} \max\{0, g_i(x)\} = s(x).$$

The assumption, which will also be considered when studying the convergence of the method, is basically a kind of Mangasarian-Fromowitz constraint qualification condition for integer problems. The condition simply says that, when we get a point that is not feasible for the original problem, we can always find a primitive direction that guarantees a reduction of the violation. This sounds pretty reasonable when dealing with the class of problems considered in here.

Now, we can prove that there exists a threshold value $\bar{\epsilon}$ for the penalty parameter such that, for any $\epsilon \in (0, \bar{\epsilon})$, any local minimum of the penalized problem is also a local minimum of the original problem.

Theorem 3 *Let Assumption 1 hold. Given Problem (10) and considering Problem (11), a threshold value $\bar{\epsilon} > 0$ exists such that for every $\epsilon \in (0, \bar{\epsilon})$, any β-local minimum point $\bar{x}$ of (11) is also a β-local minimum of (10), with $\beta = 1$.*

Proof Let us define the following penalty parameter

$$\bar{\epsilon} = \frac{a}{b}, \tag{14}$$

with

$$a = \min\{s(x) - s(x+d), x \in X, d \in D(x), s(x) - s(x+d) > 0\}$$

and

$$b = \max\{f(x+d) - f(x), x \in X, d \in D(x), f(x+d) - f(x) > 0\}.$$

By contradiction, we assume that there exists a local minimum point $\bar{x}$ of Problem (11) that is not feasible for Problem (10). Taking into account Assumption 1, we can find a direction $\bar{d} \in D(\bar{x})$, such that:

$$f(\bar{x}+\bar{d}) - f(\bar{x}) \leq b = \frac{1}{\bar{\epsilon}}a < \frac{1}{\epsilon}\left[s(\bar{x}) - s(\bar{x}+\bar{d})\right],$$

with $\epsilon \in (0, \bar{\epsilon})$. Hence, we can write

$$f(\bar{x}+\bar{d}) + \frac{1}{\epsilon}s(\bar{x}+\bar{d}) < f(\bar{x}) + \frac{1}{\epsilon}s(\bar{x}),$$

thus contradicting the fact that $\bar{x}$ is local minimum for Problem (11). □

The algorithm we use in the constrained case, called NM-BBOA_CP, has the same structure as the NM-BBOA algorithm for bound-constrained integer programs. The detailed scheme is reported in Algorithm 3. It is easy to see that the main differences between the two are the following:

1. the function f is replaced by the penalty function P;
2. a specific rule is used for the update of the penalty parameter ϵ.

As we can easily see, the algorithm checks, in Phase 2, if the update is timely. More specifically, if *Nonmonotone line search* failed along all the directions in D_k and the start stepsize is 1 for all of those directions (i.e., the dynamically changing skewed stencil defined by directions $d \in D_k$ and stepsizes $\tilde{\alpha}_k^{(d)}$ shrank to its minimum size in Phase 1), the algorithm checks if either the violation of the constraints is larger than a reference value μ_k (which goes to zero as k goes to infinity) or all of the directions in $D(x_k)$ have been generated. If this is the case, the penalty parameter decreases, otherwise it stays the same.

We finally prove finite convergence of the method to a local minimum. Again it is appropriate to note that the local minimum obtained is related to a discrete neighborhood with $\beta = 1$. We further point out that, in order to prove the result, Assumption 1 is used.

Theorem 4 *Let Assumption 1 hold. Let* $\{x_k\}$ *and* $\{\bar{P}_k\}$ *be the sequences of solutions and of reference values, respectively, generated by NM-BBOA_CP.*
Then, the algorithm terminates after a finite number of iterations $\bar{k}$ *and the produced point* $x_{\bar{k}}$ *is a* β*-local minimum of the original Problem* (10) *with* $\beta = 1$.

Proof We assume, by contradiction, that the sequence $\{x_k\}$ is infinite. Hence, also sequence $\{\epsilon_k\}$ is infinite. Considering the detailed instructions of Algorithm NM-BBOA_CP, for every iteration k, either we have $\epsilon_{k+1} = \epsilon_k$ or $\epsilon_{k+1} = \theta\epsilon_k < \epsilon_k$. Thus,

$$\lim_{k\to\infty} \epsilon_k = \tilde{\epsilon} \geq 0.$$

Then, only two different cases can happen:

Case 1. $\epsilon_k = \tilde{\epsilon}$ when k sufficiently large. In this case, taking into account (13), we have that $\mu_k < a$ for k sufficiently large. Thus, the generated points are all feasible. Hence, the proof is a verbatim repetition of the proof given for Theorem 1.
Case 2. $\epsilon_k \to 0$. We first define

$$K = \{k : \epsilon_{k+1} \neq \epsilon_k\}.$$

Taking into account the fact that X is compact, we can now consider a further subsequence $K_1 \subset K$ such that $x_k = \tilde{x}$ for all $k \in K_1$. When k is sufficiently large, we have $D_k = D(\tilde{x})$, and, from the instructions of the algorithm, we can write

$$P(\tilde{x} + d; \epsilon_k) \geq \bar{P}_k \geq P(\tilde{x}; \epsilon_k), \quad \forall\, d \in D(\tilde{x}). \tag{15}$$

Hence, multiplying by ϵ_k and considering the limit for $k \to \infty$, we have

$$\sum_{i=1}^{m} \max\{0, g_i(\tilde{x} + d)\} \geq \sum_{i=1}^{m} \max\{0, g_i(\tilde{x})\}, \quad \forall\, d \in D(\tilde{x}).$$

Algorithm 3 NonMonotone Black-Box Optimization Algorithm for Constrained Problems (NM-BBOA_CP)

Data. $x_0 \in X \cap \mathbb{Z}^n$, $D = D_0 \subset D(x_0)$ a set of initial directions, $\tilde{\alpha}_0^{(d)} = 1$, for each $d \in D_0$. $\epsilon_0 > 0$, $\theta \in (0, 1)$ and a sequence $\{\mu_k\} \downarrow 0$. $W = \{P(x_0; \epsilon_0)\}$, $P^{ref} = \bar{P}_0 = P(x_0; \epsilon_0)$, $x_{min} = x_0$, $M \geq 1$, $M \in \mathbb{N}$.

For $k = 0, 1, \ldots$

PHASE 1 - Explore points around x_k

Set $y = x_k$

While $D \neq \emptyset$ and $y = x_k$ **do**

Select and explore direction

Choose $d \in D$ set $D = D \setminus \{d\}$

Compute α by the *Nonmonotone Line Search*$(\tilde{\alpha}_k^{(d)}, x_k, d, P^{ref}; \alpha)$

Update start stepsize, reference value and queue

If $\alpha = 0$ **then** set $\tilde{\alpha}_{k+1}^{(d)} = \max\{1, \lfloor \tilde{\alpha}_k^{(d)}/2 \rfloor\}$

else

Set $y = x_k + \alpha d$ and $\tilde{\alpha}_{k+1}^{(d)} = \alpha$

If $P(y; \epsilon_k) < P(x_{min}; \epsilon_k)$ **then** $x_{min} = y$

If $|W| = M$ **then** pop(W)

push$(P(y; \epsilon_k), W)$, $P^{ref} = \max_{P \in W}\{P\}$

End If

End While

PHASE 2 - Update set of search directions and penalty parameter

If $y = x_k$ **then**

If *Nonmonotone Line Search* failed with $\tilde{\alpha}_k^{(d)} = 1$ for all $d \in D_k$ **then**

Set `upd=FALSE`

If $D_k = D(x_k)$ **then**

If $(\|g^+(x_k)\| = 0)$ **then**

If $f(x_k) = f(x_{min})$ **then return** $x^* = x_k$

else Set $y = x_{min}$ and $D = D_{k+1} = D(x_k)$

else

Set `upd=TRUE` and $D = D_{k+1} = D(x_k)$

End If

else

Generate $D_{k+1} \supset D_k$, set $\tilde{\alpha}_{k+1}^{(d)} = 1$, for all $d \in D_{k+1}$ and $D = D_{k+1} \setminus D_k$

End If

If $(\|g^+(x_k)\| > \mu_k)$ or (`upd`) **then** Set $\epsilon_{k+1} = \theta\epsilon_k$

else Set $\epsilon_{k+1} = \epsilon_k$

else

Set $\epsilon_{k+1} = \epsilon_k$

Set $D_{k+1} = D_k$ and $D = \{d \in D_k : \textit{Nonm. Line Search}$ failed with $\tilde{\alpha}_k^{(d)} > 1\}$

End If

else

Set $\epsilon_{k+1} = \epsilon_k$

Set $D_{k+1} = D_k$ and $D = D_k$

End If

PHASE 3 - Update iterates

Set $x_{k+1} = y$, $\bar{P}_{k+1} = P^{ref}$

End For

Thus we get, by recalling Assumption 1, that $\tilde{x}$ is feasible and, thanks to inequality (15), it is a local minimum for Problem (10). This contradicts the fact that the algorithm does not terminate.

Then, the theorem is proved. □

4 Numerical experiments on bound-constrained problems

In this section we report the results of the numerical experience and comparison of our nonmonotone algorithm (NM-BBOA) with other solvers from the literature on problems with only bound constraints on the variables. We also report comparison between algorithms obtained from NM-BBOA by disabling Phase 2 (thus obtaining an algorithm using dynamically changing skewed stencils) or disabling both the nonmonotone line-search procedure and Phase 2 (thus coming up with an algorithm using fixed stencil). Furthermore, we investigate to what extent the nonmonotonicity is useful in this context.

In order to evaluate the relative performances of the algorithms on bound-constrained problems, we use the 61 problems listed in Table 1.

More precisely, we use 48 unconstrained problems from sections 2 and 3 of [27], i.e., unconstrained minimax problems and general nonsmooth unconstrained problems, plus 13 bound-constrained mixed-integer problems from [30–32]. Then, for the 48 unconstrained problems, we add bound constraints on the variables as

$$\ell^i = (\tilde{x}_0)^i - 10 \leq \tilde{x}^i \leq (\tilde{x}_0)^i + 10 = u^i, \quad i = 1, \ldots, n,$$

where $\tilde{x}_0$ is the provided starting point for the problem. Furthermore, given the *continuous* or mixed-integer bound-constrained optimization problem

$$\begin{array}{ll} \min \ \tilde{f}(\tilde{x}) & \\ s.t. \ \ell^i \leq \tilde{x}^i \leq u^i, & i = 1, \ldots, n \\ \quad \tilde{x}^i \in \mathbb{Z}, & \forall\, i \in I_d \subseteq \{1, \ldots, n\} \\ \quad \tilde{x}^i \in \mathbb{R}, & \forall\, i \notin I_d \end{array}$$

we consider the discretized problem

$$\begin{array}{ll} \min \ f(x) & \\ s.t. \ \ell^i \leq x^i \leq u^i, & \forall\, i \in I_d, \\ \quad 0 \leq x^i \leq 100, & \forall\, i \notin I_d \\ \quad x \in \mathbb{Z}^n & \end{array} \tag{16}$$

where $f(x) = \tilde{f}(\tilde{x})$ with

$$\tilde{x}^i = \begin{cases} x^i, & \text{for all } i \in I_d, \\ \ell^i + x^i(u^i - \ell^i)/100, & \text{for all } i \notin I_d. \end{cases}$$

Table 1 Unconstrained and bound-constrained test problems collection

Problem name	Source	n	Problem name	Source	n
crescent	[27]	2	filter	[27]	9
cb2	[27]	2	polak 2	[27]	10
cb3	[27]	2	maxquad	[27]	10
dem	[27]	2	gill	[27]	10
wolfe	[27]	2	wong2	[27]	10
lq	[27]	2	polak 3	[27]	11
ql	[27]	2	osborne 2	[27]	11
mifflin 1	[27]	2	steiner 2	[27]	12
mifflin 2	[27]	2	shell dual	[27]	15
wf	[27]	2	watson	[27]	20
spiral	[27]	2	wong3	[27]	20
banex	[27]	2	maxl	[27]	20
pbc3	[27]	3	maxq	[27]	20
bard	[27]	3	tr48	[27]	48
evd 52	[27]	3	mxhilb	[27]	50
oet5	[27]	4	l1hilb	[27]	50
oet6	[27]	4	goffin	[27]	50
gamma	[27]	4	SOMI prob.10	[31]	5
kowalik-osborne	[27]	4	SO-I prob. 2	[32]	5
rosen-suzuki	[27]	4	SO-I prob. 7	[32]	10
polak 6	[27]	4	SO-I prob. 9	[32]	12
davidon 2	[27]	4	SO-I prob.10	[32]	30
shor	[27]	5	SO-I prob.13	[32]	10
colville 1	[27]	5	SO-I prob.15	[32]	12
exp	[27]	5	SO-I prob.16	[32]	8
pbc1	[27]	5	MISO prob. 6	[30]	15
hs78	[27]	5	MISO prob. 7	[30]	2
evd61	[27]	6	MISO prob. 8	[30]	15
elattar	[27]	6	MISO prob. 9	[30]	3
transformer	[27]	6	MISO prob.10	[30]	60
wong1	[27]	7			

As concerns the starting point x_0 for Problem (16), we set

$$(x_0)^i = \begin{cases} \lfloor (\ell^i + u^i)/2 \rfloor, & i \in I_d, \\ 50, & i \notin I_d, \end{cases} \tag{17}$$

and note that, when $i \notin I_d$, $(x_0)^i$ is nothing but

$$(x_0)^i = \lfloor 100((\tilde{x}_0)^i - \ell^i)/(u^i - \ell^i) \rceil,$$

where $\lfloor\cdot\rceil$ and $\lfloor\cdot\rfloor$ denote, respectively, the nearest integer and the floor operators.

The code related to NM-BBOA [24] and all the problems used in the experiments are available for download at the *Derivative-Free Library* homepage

http://www.iasi.cnr.it/~liuzzi/DFL/

following the links **DFLINT** [24] (code) and **TESTINT** (collection of problems).

4.1 The algorithms

In this section we briefly describe the algorithms that we used in the comparison. Apart from our proposed solver NM-BBOA (with $D_0 = \{e_1, \ldots, e_n\}$ and $M = 4$), we consider:

- M-BBOA, the monotone version of NM-BBOA, i.e., the one obtained by setting $M = 1$ in algorithm NM-BBOA;
- NM-DCS (M-DCS), nonmonotone (respectively, monotone) dynamically changing (skewed) stencil algorithm, i.e., the nonmonotone (monotone) version of the algorithm obtained from NM-BBOA (M-BBOA) where Phase 2 is disabled so that $D_{k+1} = D_k = D_0$ for all k;
- NM-FS (M-FS), nonmonotone (respectively, monotone) fixed stencil algorithm, i.e., the algorithm obtained from NM-DCS (M-DCS) where we inhibit the step expansion within the nonmonotone search procedure;
- Derivative Free Linesearch Algorithm for box constriants (DFL box), the linesearch based derivative-free algorithm for bound-constrained mixed-integer problems described in [22];
- Derivative Free Linesearch Algorithm for general constraints (DFL gen), the linesearch based derivative-free algorithm for inequality constrained mixed-integer problems described in [23];
- Nonlinear Optimization by Mesh Adaptive Direct Search (NOMAD v.3.8.1[1]), an implementation of the mesh adaptive direct search method [2,20];
- Brute Force Optimizer (BFO), the brute force derivative-free optimizer described in [34];
- Mixed-Integer Surrogate Optimization framework (MISO), a model-based approach using surrogates [30].

We note that DFL box, BFO and MISO can only handle bound-constrained problems. On the contrary, NOMAD, DFL gen and the algorithms herein proposed (i.e., BBOA, DCS and FS) can explicitly handle problems with general simulation constraints.

All the algorithms have been run specifying a maximum of 5000 function evaluations. Since the problems in our benchmark sets have at most 60 variables, this value represents a reasonable choice in practice (see, e.g., [29]). For each of the parameters listed below, we tried different values and used the one that resulted in the best reliability (Feature (iii)). For algorithms NM-BBOA and M-BBOA we use $\tilde{\alpha}_0^{(d)} = 50$,

[1] A new version of NOMAD with better functionalities, e.g. the Nelder Mead search and the possibility to specify the direction type for poll intensification, was released while this paper was under review (for further details visit the website https://www.gerad.ca/nomad/).

whereas, for algorithms DCS and FS (both NM and M versions) $\tilde{\alpha}_0^{(d)} = 1$ was used. As concerns **Step 21** of algorithms NM-BBOA and M-BBOA, namely the procedure that given set D_k returns $D_{k+1} \supset D_k$, it is implemented according to [4]. Specifically, the procedure described in Algorithm 4 below is used. The algorithm generates a vector belonging to the Halton sequence at **Step 5**. Then, at **Step 6**, a specific rounding gives the adjusted direction with all integer components. At **Step 7**, the algorithm checks if the direction is a prime vector and does not belong to D_k, and eventually updates D_k (see **Step 8**).

As regards solvers DFL box [22] and DFL gen [23], we note that they have not been developed to solve purely integer problems. For this reason, in order to use the solvers in the numerical experiments, we had to slightly adapt them both.

NOMAD has been run by using the default parameter settings except for

```
initial_mesh_size = 5; direction_type = ortho 2n.
```

In order to better understand the effectiveness of our exploration strategy when compared with the pure MADS strategy used by NOMAD for integer problems, we further run NOMAD with the option

```
disable models,
```

and report in the comparisons both versions of NOMAD.

The solver BFO has been run using default values for its parameters.

As concerns the solver MISO, we adopted the standard sampling strategy (i.e., `cptvl`) and surrogate model radial basis function (i.e., `rbf_c`). Furthermore, we provided MISO with a single initial starting point x_0 as defined in (17).

Algorithm 4 Procedure to generate new set of search directions

Input. $t > 0$, $\eta > 0$, D_k
If $\eta < 50\sqrt{n}/2$
  For $h = 1, \ldots, 1000$
    Set $t \leftarrow t + 1$
    Let u_t be the t-th vector in the n dimensional Halton sequence [18]
    Compute

$$q_t(\eta) = \left\lfloor \eta \frac{2u_t - e}{\|2u_t - e\|} \right\rceil \in \mathbb{Z}^n \cap \left[-\eta - \frac{1}{2}, \eta + \frac{1}{2} \right]^n .$$

    If $q_t(\eta)$ is a prime vector and $q_t(\eta) \notin D_k$
      Set $D_{k+1} = D_k \cup \{q_t(\eta)\}$
      return (`success`, t, η, D_{k+1})
    Endif
  End For
Endif
return (`failure`, t, η, D_k)

Data and performance profiles Performance of the different codes is assessed using data and performance profiles from [29]. Specifically, let S be a set of algorithms and P a set of problems. For each $s \in S$ and $p \in P$, let $t_{p,s}$ be the number of function

evaluations required by algorithm s on problem p to satisfy the condition

$$f(x_k) \leq f_L + \tau(f(x_0) - f_L) \tag{18}$$

where $0 < \tau < 1$ and f_L is the best objective function value achieved by any solver on problem p. Then, performance and data profiles of solver s are the following functions

$$\rho_s(\alpha) = \frac{1}{|P|} \left| \left\{ p \in P : \frac{t_{p,s}}{\min\{t_{p,s'} : s' \in S\}} \leq \alpha \right\} \right|,$$
$$d_s(\kappa) = \frac{1}{|P|} \left| \left\{ p \in P : t_{p,s} \leq \kappa(n_p + 1) \right\} \right|$$

where n_p is the dimension of problem p.

In [29], it is argued that performance and data profiles give insights on the relative performances of the compared algorithms. Three features of the profiles are particularly relevant, namely

(i) for performance profiles, the values $\rho_s(0)$;
(ii) how steeply the curves rise;
(iii) how high the curves rise.

We would like to highlight that the limiting value of $\rho_s(\alpha)$ as $\alpha \to \infty$ is the percentage of problems that can be solved with the available budget of function evaluations. Hence

$$d_s(\hat{\kappa}) = \lim_{\alpha \to \infty} \rho_s(\alpha),$$

where $\hat{\kappa}$ is the maximum number of simplex gradients performed with the available budget. This limit represents the reliability of the solver for a given tolerance τ (see [29] for further details). Thus, Feature (ii) can be used to evaluate the efficiency of the algorithms, and Feature (iii) to evaluate their reliability.

4.2 Results

First, we analyze our codes NM-BBOA, M-BBOA, NM-DCS, M-DCS, NM-FS, and M-FS. Such a comparison is reported in Fig. 2 for values of the tolerance parameter τ in $\{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$. It can be noted that both NM-BBOA and M-BBOA are quite efficient and reliable when compared with the other methods. In Fig. 3, we hence report results related to NM-BBOA and M-BBOA against NOMAD (with and without models), MISO, DFL box and BFO. From Fig. 3, we can conclude that NM-BBOA, M-BBOA and MISO are the best performing solvers on this set of problems. To better analyze the behavior of the methods on the bound-constrained problems, in Fig. 4, we report the results obtained when only problems with $n \geq 10$ variables are considered. As we can see, the profiles reported in Fig. 4 are noticeably different from those reported in Fig. 3. Here, we can still say that M-BBOA and MISO are the most efficient solvers. To better understand the respective performances of M-BBOA, NM-BBOA and MISO, in the following table we report

CPU times (in seconds) required by the methods to solve all of the problems in the collection. It is worth noting that MISO is by far the most time consuming solver

Solver	CPU time (s)
BFO	8.32
NOMAD	8.43
DFL box	9.66
NOMAD (w/mod.)	851.36
M-BBOA	676.77
NM-BBOA	658.03
MISO	93046.14

(requiring almost 25 hours to complete). Hence, we can say that the good efficiency of MISO is (at least to some extent) due to the complicated linear algebra used by the solver to construct and optimize the surrogate models during the optimization process.

When reliability is considered, NM-BBOA is the solver of choice if one seeks relatively high level of precision. This can be explained by considering that the nonmonotone algorithm might take uphill steps especially at the beginning of the optimization process. This could in turn justify a less steeper descent of the nonmonotone algorithm with respect to the monotone one. However, the nonmonotone algorithm should have a greater ability to escape from local minima, thus converging to better minimum points. This is certainly the case for larger problems, see e.g. Fig. 4, but does not seem to emerge from Fig. 3, where instead NM-BBOA and M-BBOA appear to be almost equivalent in terms of reliability. We suspect that the considered

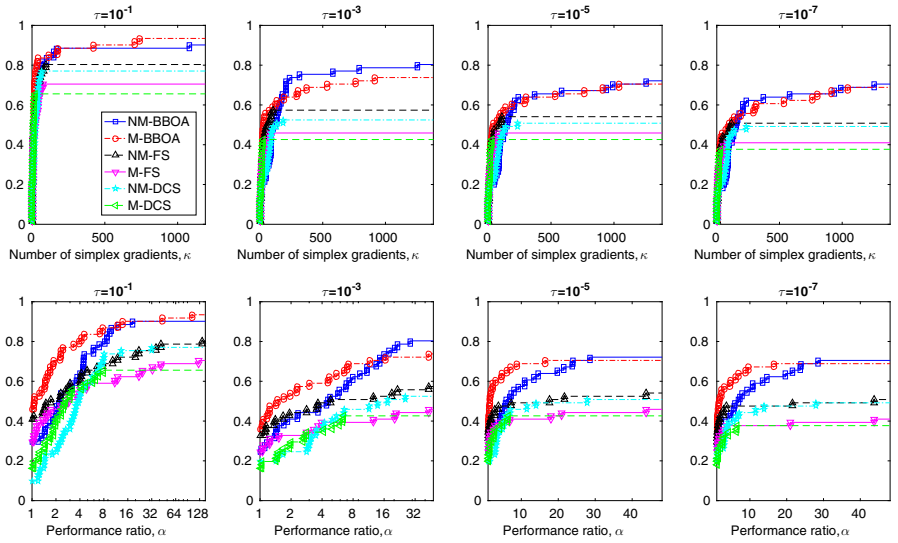

Fig. 2 Comparison between BBOA, DCS and FS, both monotone and nonmonotone, on the 61 bound-constrained problems

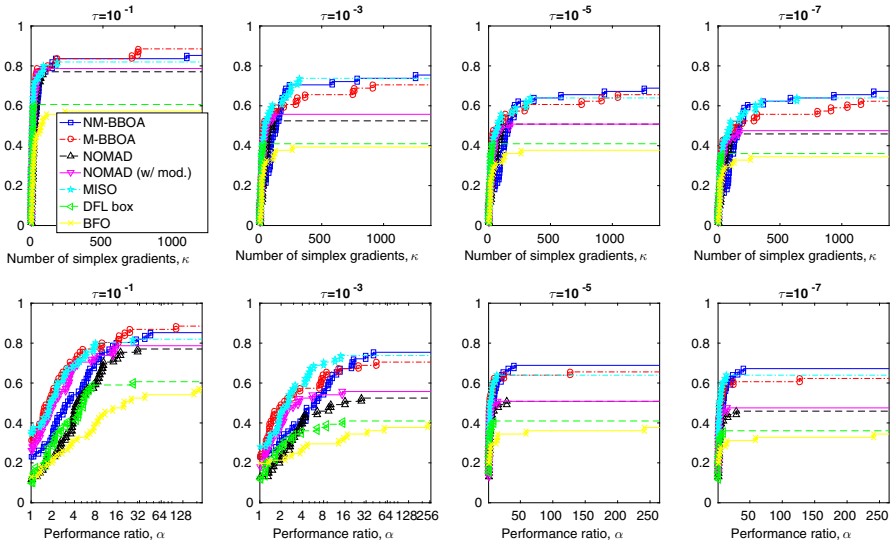

Fig. 3 Comparison between NM-BBOA, M-BBOA, NOMAD (3.8.1) with/without models, MISO, DFL box and BFO on the 61 bound-constrained problems

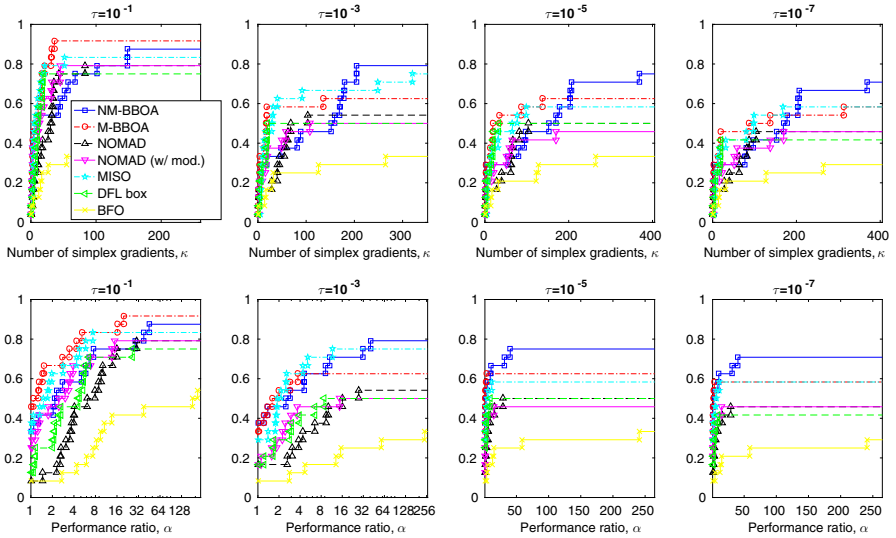

Fig. 4 Comparison between NM-BBOA, M-BBOA, NOMAD (3.8.1) with/without models, MISO, DFL box and BFO on the bound-constrained problems with $n \geq 10$

set of problems is not adequate for the advantages of the nonmonotonicity to clearly emerge. Hence, we better investigate this aspect in the next subsection where a class of hard global optimization problems is considered.

By focusing again on the data profiles reported in Figs. 3 and 4, we can finally remark that the only methods making a noticeable progress when the number of

Table 2 Effect of varying parameter β in algorithms M-BBOA and NM-BBOA on the ability to find the global minimizers for the hard bound-constrained problems

β	M-BBOA		NM-BBOA	
	# succ.	CPU time	# succ.	CPU time
1	45	1290.01	57	1320.25
2	62	1426.37	61	1286.93
5	68	1688.40	70	1335.70
10	70	1452.64	72	1384.21
20	71	1441.81	74	1447.48
30	68	1445.79	75	1404.10
50	67	1599.75	80	1416.26

simplex gradients significantly increases are MISO, NM-BBOA and M-BBOA. This implies that all of the other methods in some cases settle on a worse local/mesh-isolated solution than these solvers do.

4.3 Results on a class of hard global optimization problems

In this section we study the impact of a parameter $\beta \geq 1$ on the ability of the proposed algorithm to find the global minimum of Problem (4). To this aim, we define a class of hard bound-constrained problems. More specifically, we consider problem

$$\begin{aligned} &\min\ \varphi(x) \\ &s.t.\ 0 \leq x_i \leq 100, \quad i = 1, 2 \\ &\quad x \in \mathbb{Z}^2 \end{aligned} \tag{19}$$

where

$$\varphi(x) = \min_{j=1,\ldots,20} \ln\left(\|c_j - x\| + \sigma_j\right)$$

with c_j, $j \in \{1, \ldots, 20\}$, random feasible points for Problem (19), $\sigma_j = 10^{-2}$, $j \in \{1, \ldots, 20\} \backslash \bar{J}$, and $\sigma_j = 10^{-6}$, $j \in \bar{J}$, where $\bar{J}$ is a random subset of $\{1, \ldots, 20\}$ with $|\bar{J}| = 3$. Note that, by definition of $\varphi(x)$, it results

$$\varphi(c_j) = \begin{cases} \ln(10^{-6}) & j \in \bar{J}, \\ \ln(10^{-2}) & j \notin \bar{J}. \end{cases}$$

As a first step, we investigate the effect of varying parameter β (both in M-BBOA and NM-BBOA) on the ability of the algorithm to find the global minimizers. For the sake of clarity, we denote M-BBOA(β) and NM-BBOA(β), respectively, M-BBOA and NM-BBOA when parameter $\beta > 1$ is used. Note that M-BBOA and NM-BBOA use a value $\beta = 1$. All the algorithms have been run on a set of 100 randomly generated problems of the form (19), allowing a maximum of 5000 function evaluations.

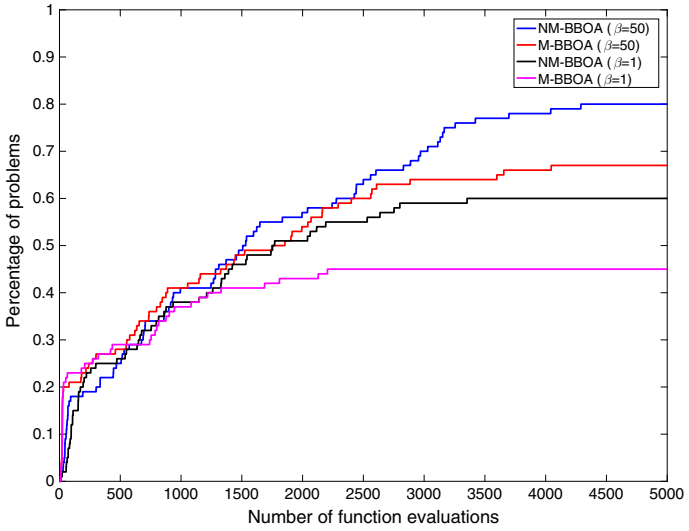

Fig. 5 Comparison between NM-BBOA (using $\beta = 1$), M-BBOA (using $\beta = 1$), modified NM-BBOA (using $\beta = 50$), and modified M-BBOA (using $\beta = 50$) on hard global optimization problems

Results of such experimentation are reported in Table 2. For $\beta = 1, 2, 5, 10, 20, 30$ and 50, we report the number of times the algorithm found the global minimum out of the 100 runs, and the CPU time required to solve all the 100 problems.

As we can see, both the codes perform better (in terms of number of times they found the global minimum point) when $\beta = 50$. We can also note, from the reported CPU times, that NM-BBOA($\beta = 50$) is only 10% slower than NM-BBOA($\beta = 2$) to solve all of the 100 problems. As concerns M-BBOA, we can see that M-BBOA($\beta = 50$) is 24% slower than M-BBOA($\beta = 1$).

Now, we compare the performance of M-BBOA and NM-BBOA with their respective modified versions, i.e., those that explore larger neighborhoods. Taking into account Remark 2, the comments at the end of Sect. 2, and the considerations in the above paragraph, we used $\beta = 50$ in the modified version of the algorithms. In Fig. 5, we plot the percentage of problems solved to global optimality (y axis) with the given number of function evaluations (x axis). As we can easily see, NM-BBOA($\beta = 50$), when the number of evaluations is large enough (i.e., larger than 1500), is able to find a global minimum with higher probability. Furthermore, if we allow a sufficient number of function evaluations (say 2000), a ranking between the algorithms can be done. Precisely and as expected, NM-BBOA($\beta = 50$) is the best method. The second best method is M-BBOA($\beta = 50$), third best is NM-BBOA, and M-BBOA is the worst one on this set of problems. These results seem to indicate that, on problems with many local minima if we are interested in finding good points, we can suitably modify the proposed algorithm into two different ways. First, we can increase parameter M that determines the amount of non-monotonicity of the algorithm which can help to escape from (useless) local minima. Second, we can operate on the parameter β thus allowing the algorithm to explore larger neighborhoods, which again can help escape from local minima.

5 Numerical experiments on problems with general simulation constraints

This section is devoted to the analysis of the results obtained by the proposed algorithms on generally-constrained test problems. Furthermore, comparison with NOMAD (version 3.8.1) [2,20] is reported.

We defined a collection of constrained problems by adding to every bound-constrained problem (among the 48 problems from [27] described in Table 1) the following families of nonlinear constraints.

$$
\begin{aligned}
&\tilde{g}_j(\tilde{x}) = (3-2x_{j+1})x_{j+1} - x_j - 2x_{j+2} + 1 \le 0, && j = 1,\dots,n-2\ (n \ge 3);\\
&\tilde{g}_j(\tilde{x}) = (3-2x_{j+1})x_{j+1} - x_j - 2x_{j+2} + 2.5 \le 0, && j = 1,\dots,n-2\ (n \ge 3);\\
&\tilde{g}_j(\tilde{x}) = x_j^2 + x_{j+1}^2 + x_j x_{j+1} - 2x_j - 2x_{j+1} + 1 \le 0, && j = 1,\dots,n-1\ (n \ge 2);\\
&\tilde{g}_j(\tilde{x}) = x_j^2 + x_{j+1}^2 + x_j x_{j+1} - 1 \le 0, && j = 1,\dots,n-1,\ (n \ge 2);\\
&\tilde{g}_j(\tilde{x}) = (3-0.5x_{j+1})x_{j+1} - x_j - 2x_{j+2} + 1 \le 0, && j = 1,\dots,n-2\ (n \ge 3);\\
&\tilde{g}_1(\tilde{x}) = \sum_{j=1}^{n-2}((3-0.5x_{j+1})x_{j+1} - x_j - 2x_{j+2} + 1) \le 0, && (n > 3).
\end{aligned}
$$

This way we obtain a set of 240 test problems with $n \in [2, 50]$ and $m \in [1, 49]$. Note that, as already discussed, given the *continuous* constraint function $\tilde{g}_j(\tilde{x})$, $j = 1,\dots,m$, we consider the discretized constraint function $g_j(x)$ such that $g_j(x) = \tilde{g}_j(y)$ with $y_i = \ell_i + x_i(u_i - \ell_i)/100$, $i = 1,\dots,n$.

We would like to note that parameter μ_k needs to be properly set and updated in NM-BBOA. In practice, we initialize $\mu_0 = 1$ and use the following updating rule

$$\mu_{k+1} = \max\{\varepsilon,\ 0.5\mu_k\},$$

with $\varepsilon > 0$ a reasonably small feasibility tolerance, e.g., $\varepsilon = 10^{-8}$.

5.1 Results

When constraints are present in the definition of the problem, performance and data profiles as proposed in [29] cannot be directly used. We adapt the procedure to construct performance and data profiles as proposed in [13]. Specifically, we consider the convergence test

$$\hat{f}_0 - f(x) \ge (1-\tau)(\hat{f}_0 - f_L),$$

where $\hat{f}_0$ is the objective function value of the worst feasible point determined by all the solvers (note that in the bound-constrained case, $\hat{f}_0 = f(x_0)$), and f_L is computed for each problem as the smallest value of f (at a feasible point) obtained by any solver within 5000 function evaluations. Note that when a point is not feasible (i.e., $g(x) \not\le 0$) we set $f(x) = +\infty$.

In Fig. 6, we report the comparison between algorithms NM-BBOA_CP, NOMAD (3.8.1) where use of models is, respectively, allowed and inhibited through the option

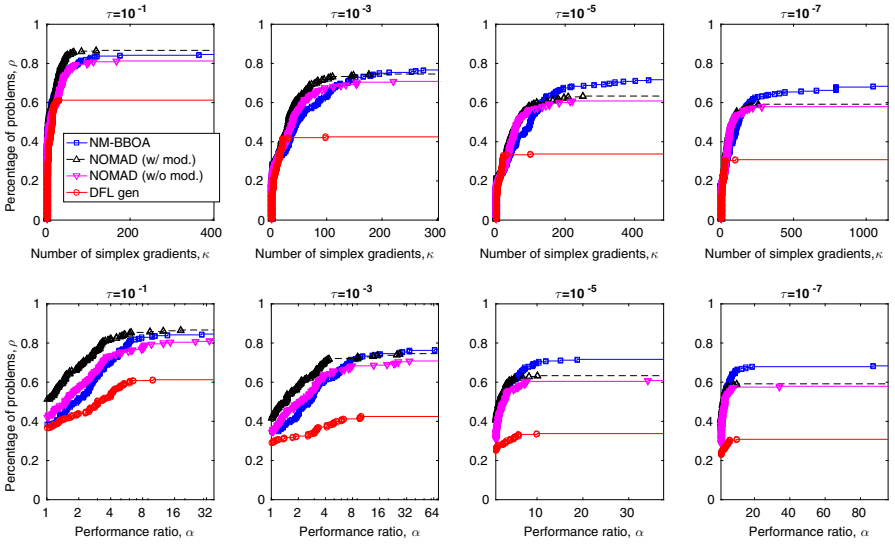

Fig. 6 Comparison between NM-BBOA_CP, NOMAD (3.8.1) with and without models on the 240 generally-constrained problems

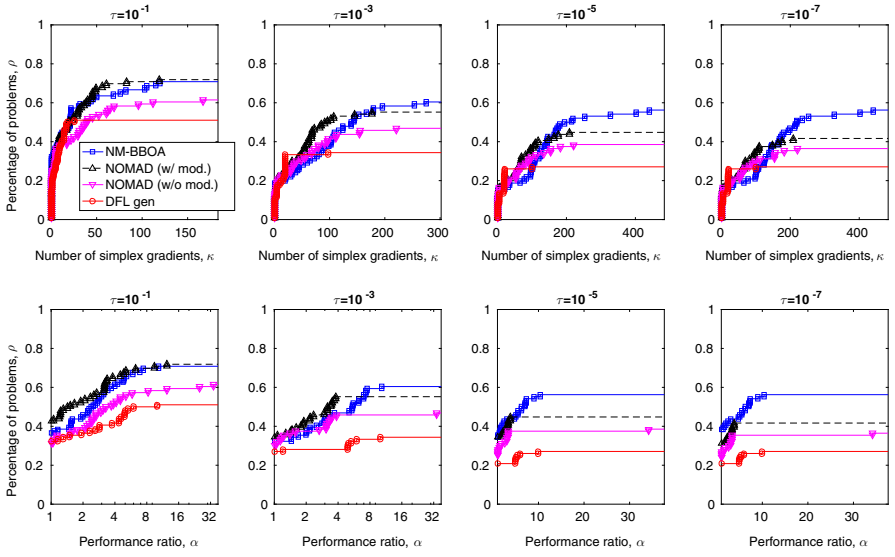

Fig. 7 Comparison between NM-BBOA_CP and NOMAD (3.8.1) without models on the 96 generally-constrained problems with $n \geq 10$

`disable models`, and DFL gen. As we can see, NOMAD using models is (obviously) better than the version of NOMAD without models. We can also see that NOMAD with models is slightly more efficient than NM-BBOA_CP but considerably less reliable. Also, note that NM-BBOA_CP is (almost) as efficient as NOMAD without models but, again, considerably more reliable. Finally, note that DFL gen is

less efficient with respect to the other solvers. This is mainly due to the lattice exploration strategy performed by DFL gen, which either is not able to obtain feasibility or recovers feasible points with high objective function value. To better investigate the situation, we repeat the same comparisons but on a restricted test set obtained by selecting problems with $n \geq 10$ among the 240 constrained problems, thus obtaining a subset of 96 problems. Such a comparison is reported in Fig. 7. As it can be noted, NM-BBOA_CP is now both more efficient and more reliable than NOMAD without models. Again, when we allow the use of models within NOMAD, the gap in terms of efficiency is considerably reduced but NM-BBOA_CP is still more reliable than NOMAD. Again, we note that DFL gen is less efficient on this set of problems.

6 Conclusions

In this paper, we developed a tailored strategy for solving black-box problems with integer variables. The use of primitive directions combined with a suitably developed nonmonotone line search gives a high level of freedom when exploring the integer lattice and further guarantees a high level of reliability. We first described and analyzed in depth a version of the algorithm that handles bound-constrained problems. Then, we tackled the generally-constrained case by embedding a penalty approach in the algorithmic framework.

We also included an extensive numerical analysis on a large testbed of both bound-constrained and generally-constrained problems. As a first step, we both studied the effects of using enriched stencils and compared monotone vs nonmonotone acceptance rules in our algorithmic framework. The results showed that

- sampling the function by such a dynamically changing set of search directions can significantly improve the performance of the algorithm;
- the use of a nonmonotone line search can get better results in terms of reliability especially when dealing with noisy problems.

A comparison with NOMAD (with and without models), BFO and MISO was carried out on bound-constrained problems. The results we reported allow us to conclude that our strategy gives good performances both in terms of efficiency and reliability, and the gap between our algorithm and the others becomes noticeable as we focus on problem with a number of variables greater or equal than 10 and ask for higher precisions.

The results on problems with simulation constraints showed that our strategy clearly outperforms DFL gen and is very competitive with the version of NOMAD that does not embed models. When compared to NOMAD with models, our method can only guarantee better performances in terms of reliability. The results change if we focus on instances with a number of variables larger or equal than 10. Indeed, in this case, our algorithm outperforms NOMAD both in terms of efficiency and reliability when precision is sufficiently high (and the gap between the two increases as we ask for higher precisions).

Some preliminary results on a class of hard global optimization problems (with bound constraints) further highlighted potential of the algorithm in finding global

minima when larger neighborhoods are explored (i.e., when a $\beta > 1$ is properly chosen in the algorithm).

Future research might follow two different lines:

- Development of an extension that can handle mixed-integer problems. This challenging task deserves an in depth computational and theoretical analysis. Indeed, it is not clear how to embed continuous line searches in the framework and integrate them with the discrete ones in order to both guarantee convergence and have good performances in practice.
- Inclusion of models in the algorithmic framework (i.e., understanding how those models should interact with the existing strategies). It is important to notice that the use of models would imply heavy changes in the structure of the framework and would also require a fresh new analysis of the convergence properties.

References

1. Abramson, M.A., Audet, C., Chrissis, J.W., Walston, J.C.: Mesh adaptive direct search algorithms for mixed variable optimization. Optim. Lett. **3**(1), 35–47 (2009)
2. Abramson, M.A., Audet, C., Couture, G., Dennis Jr., J.E., Le Digabel, S.: The NOMAD project. http://www.gerad.ca/nomad
3. Abramson, M.A., Audet, C., Dennis Jr., J.E.: Filter pattern search algorithms for mixed variable constrained optimization problems. Pac. J. Optim. **3**(3), 477–500 (2007)
4. Abramson, M.A., Audet, C., Dennis Jr., J.E., Le Digabel, S.: OrthoMADS: a deterministic MADS instance with orthogonal directions. SIAM J. Optim. **20**(2), 948–966 (2009)
5. Audet, C., Dennis Jr., J.E.: Pattern search algorithms for mixed variable programming. SIAM J. Optim. **11**(3), 573–594 (2001)
6. Audet, C., Hare, W.: Derivative-Free and Blackbox Optimization. Springer Series in Operations Research and Financial Engineering. Springer, Berlin (2017)
7. Audet, C., Le Digabel, S., Tribes, C.: The mesh adaptive direct search algorithm for granular and discrete variables. SIAM J. Optim. **29**(2), 1164–1189 (2019)
8. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. CSUR **35**(3), 268–308 (2003)
9. Conn, A., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization, vol. 8. SIAM, Philadelphia (2009)
10. Costa, A., Nannicini, G.: RBFOpt: an open-source library for black-box optimization with costly function evaluations. Math. Program. Comput. **10**(4), 597–629 (2018). https://doi.org/10.1007/s12532-018-0144-7
11. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
12. Diniz-Ehrhardt, M., Martínez, J., Raydán, M.: A derivative-free nonmonotone line-search technique for unconstrained optimization. J. Comput. Appl. Math. **219**(2), 383–397 (2008)
13. Fasano, G., Liuzzi, G., Lucidi, S., Rinaldi, F.: A linesearch-based derivative-free approach for nonsmooth constrained optimization. SIAM J. Optim. **24**(3), 959–992 (2014)
14. García-Palomares, U.M., Rodríguez, J.F.: New sequential and parallel derivative-free algorithms for unconstrained minimization. SIAM J. Optim. **13**(1), 79–96 (2002)
15. Gendreau, M., Potvin, J.Y.: Handbook of Metaheuristics, vol. 2. Springer, Berlin (2010)
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co. Inc, Boston, MA (1989)
17. Grippo, L., Rinaldi, F.: A class of derivative-free nonmonotone optimization algorithms employing coordinate rotations and gradient approximations. Comput. Optim. Appl. **60**(1), 1–33 (2015)
18. Halton, J.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer. Math. **2**, 84–90 (1960)
19. Larson, J., Leyffer, S., Palkar, P., Wild, S.M.: A method for convex black-box integer global optimization. arXiv preprint arXiv:1903.11366 (2019)

20. Le Digabel, S.: Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm. ACM Trans. Math. Softw. **37**(4), 44:1–44:15 (2011)
21. Le Digabel, S., Wild, S.M.: A taxonomy of constraints in simulation-based optimization. arXiv preprint arXiv:1505.07881 (2015)
22. Liuzzi, G., Lucidi, S., Rinaldi, F.: Derivative-free methods for bound constrained mixed-integer optimization. Comput. Optim. Appl. **53**(2), 505–526 (2012)
23. Liuzzi, G., Lucidi, S., Rinaldi, F.: Derivative-free methods for mixed-integer constrained optimization problems. J. Optim. Theory Appl. **164**(3), 933–965 (2015)
24. Liuzzi, G., Lucidi, S., Rinaldi, F.: DFLINT—an algorithm for black-box inequality and box constrained integer nonlinear programming problems (2020). https://doi.org/10.5281/zenodo.3653742. http://www.iasi.cnr.it/~liuzzi/DFL
25. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, pp. 320–353. Springer, Boston, MA (2003). https://doi.org/10.1007/0-306-48056-5_11
26. Lucidi, S., Piccialli, V., Sciandrone, M.: An algorithm model for mixed variable programming. SIAM J. Optim. **15**(4), 1057–1084 (2005)
27. Lukšan, V., Vlček, J.: Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical report VT798-00, Institute of Computer Science, Academy of Sciences of the Czech Republic (2000)
28. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
29. Moré, J., Wild, S.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**(1), 172–191 (2009)
30. Müller, J.: MISO: mixed-integer surrogate optimization framework. Optim. Eng. **17**(1), 177–203 (2016)
31. Müller, J., Shoemaker, C.A., Piché, R.: SO-MI: a surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. Comput. Oper. Res. **40**(5), 1383–1400 (2013)
32. Müller, J., Shoemaker, C.A., Piché, R.: SO-I: a surrogate model algorithm for expensive nonlinear integer programming problems including global optimization applications. J. Glob. Optim. **59**(4), 865–889 (2014)
33. Newby, E., Ali, M.M.: A trust-region-based derivative free algorithm for mixed integer programming. Comput. Optim. Appl. **60**(1), 199–229 (2015)
34. Porcelli, M., Toint, P.L.: BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables. ACM Trans. Math. Softw. **44**(1), 1–25 (2017)