

# Heuristic methods for noisy derivative-free bound-constrained mixed-integer optimization

Morteza Kimiaei<sup>1\*</sup> and Arnold Neumaier<sup>1</sup>

<sup>1,2</sup>Fakultät für Mathematik, Universität Wien,  
Oskar-Morgenstern-Platz 1, A-1090, Wien, Austria.

\*Corresponding author(s). E-mail(s): [kimiaeim83@univie.ac.at](mailto:kimiaeim83@univie.ac.at);

Contributing authors: [Arnold.Neumaier@univie.ac.at](mailto:Arnold.Neumaier@univie.ac.at);

## Abstract

This paper discusses **MATRS**, a new matrix adaptation trust region strategy for solving noisy derivative-free mixed-integer optimization problems with simple bounds. **MATRS** repeatedly cycles through five phases, mutation, selection, recombination, trust-region, and mixed-integer in this order. But if in the mutation phase a new best point (point with lowest inexact function value among all evaluated points so far) is found, the selection, recombination, and trust region phases are skipped. Similarly, if the recombination phase finds a new best point, the trust region phase is skipped. The mixed-integer phase is always performed. To search for new best points, the mutation and recombination phases use extrapolation whereas the mixed-integer phase performs a mixed-integer line search along directions going into a valley. Numerical results on several collections of test problems show that **MATRS** is competitive with state-of-the-art derivative-free mixed-integer optimization solvers.

**Keywords:** Mixed-integer; derivative-free noisy optimization; heuristic optimization; randomized optimization; evolution strategy; trust region; line search

**2020 AMS Subject Classification:** 90C1; 90C30; 90C56; 90C15.

# 1 Introduction

The design and development of derivative-free optimization (DFO) algorithms have numerous applications in science, engineering, industry, and chemistry. The books of AUDET & HARE [2] and CONN et al. [7] and the survey paper of LARSON et al. [21] discussed DFO algorithms and their applications. In this paper, we introduce MATRS, a *new mixed-integer matrix adaptation trust region strategy*. The pure-integer search of MATRS is a development of the bound-constrained derivative-free integer optimization solver IMATRS discussed in the unpublished manuscript by KIMIAEI & NEUMAIER [18]. MATRS finds solutions of noisy derivative-free mixed-integer bound-constrained optimization problems

$$\begin{aligned} \min & f(x) \\ \text{s.t.} & x \in \mathbf{x}, x_i \in s_i \mathbb{Z}, i \in I. \end{aligned} \quad (1)$$

Here

$$\mathbf{x} := \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \bar{x}\} \text{ with } \underline{x}, \bar{x} \in \mathbb{R}^n (\underline{x} < \bar{x}) \quad (2)$$

is a **box**,  $I$  is a subset of  $\{1, \dots, n\}$ ,  $s_i > 0$  is a resolution factor, and the real-valued function  $f : \mathbf{x} \rightarrow \mathbb{R}$  is defined on the feasible set

$$C := \{x \in \mathbf{x} \mid x_i \in s_i \mathbb{Z}, \text{ for } i \in I\}. \quad (3)$$

Standard mixed-integer problems are covered for  $s_i = 1$ ; other scaling factors define **granular variables**  $x_i$  (named so in AUDET et al. [3]) whose values are fixed integral multiples of  $s_i$ . Granular variables were first handled in the SNOBFIT algorithm (HUYER & NEUMAIER [15]). They are needed, e.g., if variables are required to be represented in a fixed point format. Standard mixed-integer solvers can handle granular variables by a change of variables  $x_i \leftarrow x_i/s_i$ .

We assume that the function  $f$  is available only by a noisy oracle, returning an approximate function value  $\tilde{f}(x)$  of  $f(x)$ . The **noise**  $\tilde{f}(x) - f(x)$  is **deterministic** if calling the oracle repeatedly at the same point returns the same approximate function value, and **stochastic** otherwise. Sources of deterministic noise may be modelling, truncation, and/or discretization errors or rounding errors, and the sources of stochastic noise may be inaccurate measurements or stochastic simulation.

The point with smallest function value among all function values of the points evaluated by MATRS so far is called the **best point**.

## 1.1 Related work

DFO techniques for mixed-integer problems are almost always extensions of techniques for continuous solvers that add features for handling integer variables.

The survey of PLOSKAS & SAHINIDIS [28] investigated the behavior of integer and mixed-integer DFO solvers. DFLINT [23] is an integer DFO solvers, while BFO [29], DFLBOX [22], DFNDFL [12], NOMAD [1], MISO [26], and SNOBFIT [15] are mixed-integer DFO solvers.

The papers of MORÉ & WILD [25], RIOS & SAHINIDIS [30], KIMIAEI [17], and KIMIAEI & NEUMAIER [20] survey the behavior of DFO algorithms with continuous variables.

**Direct search methods** evaluate  $f$  at trial points obtained from the best point by adding coordinate directions, directions from a fixed poll set, or random directions to find a reasonable reduction of the inexact function value. If such a reduction is found, the trial point is accepted as the new best point and the corresponding step size is expanded or remains unchanged. Otherwise, the trial point is discarded and the search is repeated with a reduce step size. NOMAD and BFO are the two well-known direct search solvers.

**Line search methods** use extrapolation to quickly leave a saddle point or maximizer in cases where the slope of the function at the current point is small, but no local minimizer is nearby. Extrapolation expands step sizes as long as reductions of inexact function values are found along a fixed random or coordinate direction. As in the direct search methods, the corresponding step size is reduced if no reduction of the inexact function value is found at the trial point. DFLINIT uses an integer line search and DFLBOX uses integer and continuous line searches. DFLINIT is an extended version of the integer line search of DFLBOX. DFNDFL uses DFLINIT for integer searches and DFN [10] for continuous searches.

**Space-filling methods** use sequences with good space filling properties for various purposes, such as

- the selection of initial points by global solvers (cf. HUYER & NEUMAIER [14]) with the goal of finding regions close to an approximate stationary point,
- the generation of well-distributed points on a unit simplex for evolutionary multi-objective optimization (cf. BLANK et al. [5]),
- the selection of initial sampled points for the construction of quadratic models (cf. HUYER & NEUMAIER [15]),
- the generation of integer directions for line searches. DFLINT (cf. Table 2, below) calls `generate_dirs` to generate integer directions using the Halton sequences, a family of low-discrepancy sequences (cf. DICK & PILLICHSHAMMER [8] and NIEDERREITER [27]).

**Model-based methods** approximate the objective function values by quadratic model functions whose approximate gradient and Hessian matrix are obtained by interpolation or fitting. To avoid large steps, these models are

## 4 1. INTRODUCTION

constrained by trust regions. The constrained solutions of these models are chosen as the directions. The trial point is accepted as the new best point if the agreement between the objective function and the model function is good. In this case, the trust region is extended or remains unchanged. Otherwise, the trial point is discarded and the trust region is reduced to find small steps in the hope of finding a reduction of the inexact function value in the next attempt. **NOMAD** provides model-based direct search methods and **SNOBFIT** is a model-based solver. **MISO** is another model-based solver that uses various types of radial basis functions, sampling techniques, and initial experimental design options.

**Matrix adaptation evolution strategies (MAES)** use a covariance matrix or an affine scaling matrix to define the newly sampled points (e.g., see for the continuous search the recent paper by KIMIAEI & NEUMAIER [19]). They alternate three different phases:

**Mutation phase.** In this phase, some mutation points are generated, each of which is the sum of the previous recombination point (defined below, initially an initial point) and the mutation direction, scaled by a fixed step size (initially given and then computed in the third phase). Each mutation direction is the product of the corresponding distribution direction and the affine scaling matrix, which is adaptively determined in a heuristic manner. The distribution directions are selected from a normal distribution.

**Selection phase.** The inexact function values of the mutation points are sorted in ascending order, and the distribution directions and mutation directions are sorted accordingly. Then a finite number of the sorted points with low inexact function values and the sorted corresponding directions are selected for the next phase.

**Recombination phase.** The new recombination point is then the sum of the previous recombination point and the recombination mutation direction, scaled by the recombination step size. The fixed step size in the mutation phase is the recombination step size. The recombination mutation direction is a weighted average of the selected mutation directions. The recombination distribution direction, which is a weighted average of the selected distribution directions, is used directly to obtain a new affine scaling matrix and indirectly to calculate a new recombination step size.

## 1.2 An overview of our new solver

In this section, we summarize the main features of **MATRS**, a new solver based on mixed-integer matrix adaptation trust region strategy. It is designed to find solutions of noisy derivative-free mixed-integer bound-constrained optimization problems of the form (1). Compared to **MAES**, **MATRS** preserves the

selection phase, improves the mutation and recombination phases, and adds two new phases, trust region and mixed-integer.

We write  $x_I$  and  $x_K$  for the subvectors of  $x$  indexed by  $I$  and  $K$ , where  $K$  is the set of indices not contained in  $I$ . Mutation and recombination points in the space of all  $x_I$  are called **integer mutation points** and **integer recombination points** and in the space of all  $x_K$  are called **continuous mutation points** and **continuous recombination points**. The same is true for the distribution, mutation and recombination directions.

### 1.2.1 Improved mutation phase

Integer and continuous distribution directions are cheaply generated by a new technique with good space-filling properties, discussed in Section 2.

Our improved continuous mutation phase has the following new features:

- A continuous derivative-free line search strategy produces continuous mutation points that are hopefully selected as new best points.
- Real initial step sizes of continuous line searches are found heuristically.

Our improved integer mutation phase has the following new features:

- An integer derivative-free line search strategy produces integer mutation points that are hopefully selected as new best points.
- Integer initial step sizes of integer line searches are heuristically found.

### 1.2.2 Improved recombination phase

Our improved continuous and integer recombination phases have the following new features:

- The weights used to compute the continuous and integer recombination directions are scaled with a randomized scaling vector.
- A new continuous trust region strategy is performed to hopefully find a new best point.
- A new integer trust region strategy is performed to hopefully find a new best point. Its subproblem is transformed in a new way into bound-constrained integer least squares problem.
- The product of the affine scaling matrix and its transpose is used inexpensively as an approximate symmetric Hessian matrix of the model function of the trust region subproblem.

## 6 2. SPACE-FILLING SEQUENCES

- Recombination step size is used as a good replacement for the initial trust region radius.

### 1.2.3 New mixed-integer phase

Our new mixed-integer phase has the following new features:

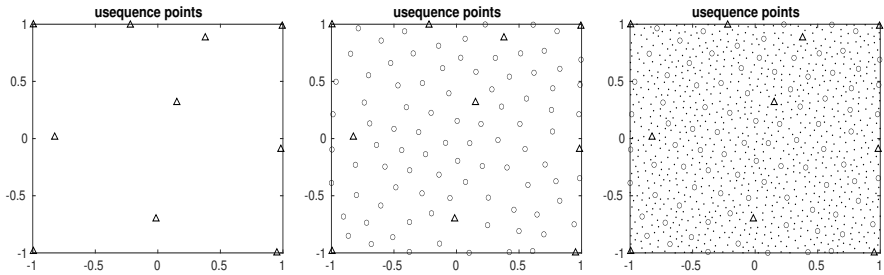
- Two new combination directions are computed with the goal of going into or moving down a valley.
- A mixed-integer line search in the space of all  $x$  is tried along exactly one of two combination directions.
- If the search in the space of all  $x$  is not possible, exactly one of integer line search in the space of all  $x_I$  and continuous line search in the space of all  $x_K$  along exactly one of combination directions is performed.

## 2 Space-filling sequences

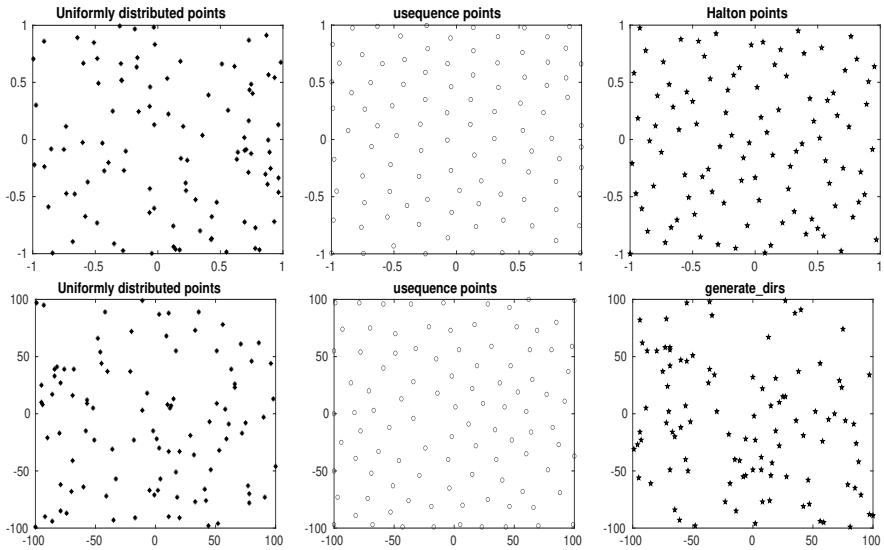
This section describes a new technique for cheaply generating sequences with good space-filling properties. Given a vector  $z$  with  $n$  nonnegative integral components, the technique generates sequences of well-distributed points,  $x \in \mathbb{R}^n$  with components  $x_i \in [-1, 1] \subseteq \mathbb{R}$  (if  $z_i = 0$ ) and  $x_i \in [-b; b] \subseteq \mathbb{Z}$  (if  $z_i = b > 0$ ), that are not too close too each other, no matter which interval segment of the sequence is considered (cf. Figure 1).

Space-filling methods generate sequences  $x_1, x_2, x_3, \dots$  of points or directions such that for any  $k$  the first  $k$  points fill the box  $[-1, 1]^n$  in a well-distributed way with large minimum distance. Deterministic low-discrepancy sequences such as the Halton sequences are expensive to generate and suffer from irregularities when the number of points is not very large. This can be seen in Figure 2.

Our new randomized procedure `usequence` is a cheap and improved substitute for the Halton sequences in the continuous case and the integer case. One can see from Figure 2 that the distribution of points generated by `usequence` is markedly superior to sequences based on either random placement or the Halton sequences.



**Fig. 1:** Plots of 10, 100, and 1000 real points belonging  $[-1, 1]^2$  generated by usequence.



**Fig. 2:** Plots of points generated by random generators, by usequence, and the Halton sequences. First row: The first 100 real points in  $[-1, 1]^2$ . Second row: The first 100 integer points in  $[-100 : 100]^2$ .

---

**Algorithm 1** usequence

---

**goal:** usequence generates a well-distributed sequence of points.

---

**function**  $D = \text{usequence}(n, m, z, p)$

---

**input:**  $n$ : dimension of the points

$m$ : number of points

$z$ : determines component types ( $z_i > 0$ : integer,  $z_i = 0$ : real)

$p$ : first point of the sequence

**output:**  $D$ : matrix whose columns are well-defined random points

---

```

1: for  $i = 1, \dots, n$  do                                 $\triangleright$  generate random reservoir matrix
2:   if  $z_i > 0$ ,  $R_{i:} = \text{randi}([-z_i, z_i], 1, m) \in \mathbb{Z}^{n \times m}$ .
3:   else,  $R_{i:} = 2 \text{rand}(1, m) - 1 \in \mathbb{R}^{n \times m}$ .
4:   end if
5: end for
6:  $R_{:,1} = p$  and  $j = 1$ .
7: for  $k = 1, \dots, m$  do
8:    $D_{:,k} = R_{:,j}$ .                                          $\triangleright$  next point
9:   for  $i = 1, \dots, n$  do                                 $\triangleright$  generate a new random vector
10:    if  $z_i > 0$ ,  $r_i = \text{randi}([-z_i, z_i], 1) \in \mathbb{Z}$ .
11:    else,  $r_i = 2 \text{rand} - 1 \in \mathbb{R}$ .
12:    end if
13:    end for
14:     $R_{:,j} = r$ .                                            $\triangleright$  reservoir update
15:     $\text{on} = \text{ones}(1, k)$ ;  $R' = r_{:, \text{on}}$ ;  $D' = D_{:, 1:k}$ ;  $\triangleright$  forming matrices  $R'$  and  $D'$ 
16:     $u = \sum_{\ell=1:k} (R'_{:\ell} - D'_{:\ell})^2$ ;  $u_j = \min_{\ell=1:n} u$   $\triangleright$  minimum squared distance value
17:     $\text{on} = k + \text{zeros}(1, n)$ ;  $D'' = D_{:, \text{on}}$ ;               $\triangleright$  forming the matrix  $D''$ 
18:     $s = \sum_{\ell=1:m} (R_{:\ell} - D''_{:\ell})^2$ ;                     $\triangleright$  distance between  $R$  and  $D''$ 
19:    if  $k = 1$ ,  $u = s$ ;
20:    else,  $u = \min(u, s)$ ;                                 $\triangleright$  minimum squared distance vector  $u$ 
21:    end if
22:    Find  $j = \max_{t=1:n} u_t$  and choose  $R_{:,j} \in \{R_{:,l}\}_{l=1}^k$ .
23: end for

```

---



**usequence** first generates an initial random matrix of reservoir points from which a sequence of well-distributed points is constructed. Each but the first point in the sequence is obtained by picking from the reservoir a point that has the minimum squared distance from the points already selected. The reservoir is updated by replacing the point chosen by a new random point.

In the **usequence** algorithm,  $j$  is the index of the reservoir vector with largest minimum distance from the vectors already chosen,  $\text{rand}(n, 1)$  is a real random vector whose  $n$  components are independent and uniformly distributed in  $[0, 1]$ ,  $\text{randi}([a_i, b_i], n, 1)$  generates integer random vectors whose  $n$  components are independent and uniformly distributed in  $[a_i, b_i]$ , and  $p$  is the first point of the initial reservoir vector. Here  $\text{ones}(1, n)$  (line 15) is a  $1 \times n$  vector whose components are one and  $\text{zeros}(1, n)$  (line 17) is a  $1 \times n$  vector whose components are zero.

### 3 MATRS

MATRS is an improved matrix adaptation trust region strategy that includes: Integer and continuous mutation phases, selection phase, integer and continuous recombination phases, integer and continuous trust region, and mixed-integer line search phase. MATRS improves its mutation and recombination phases using line search strategies to find new best points, as well as heuristic optimization techniques. When these strategies are unable to find new best points, the trust region phase is performed to find new best points. At the end of each iteration of MATRS, the mixed-integer phase is performed to find new best points, regardless of whether or not the line search and trust region strategies can find such best points.

MATRS alternately performs **cMATRS**, a continuous matrix adaptation trust region strategy, in the space of all  $x_K$  and **iMATRS**, an integer matrix adaptation trust region strategy, in the space of all  $x_I$ . Then, the mixed-integer phase performs a mixed-integer line search strategy, called **miLSS**, along combination directions to hopefully reduce the inexact function values. The goal of combination directions is to leave points with large inexact function values, go into a valley, and move down along that valley.

Figure 3 is a flowchart for the ingredients of MATRS:

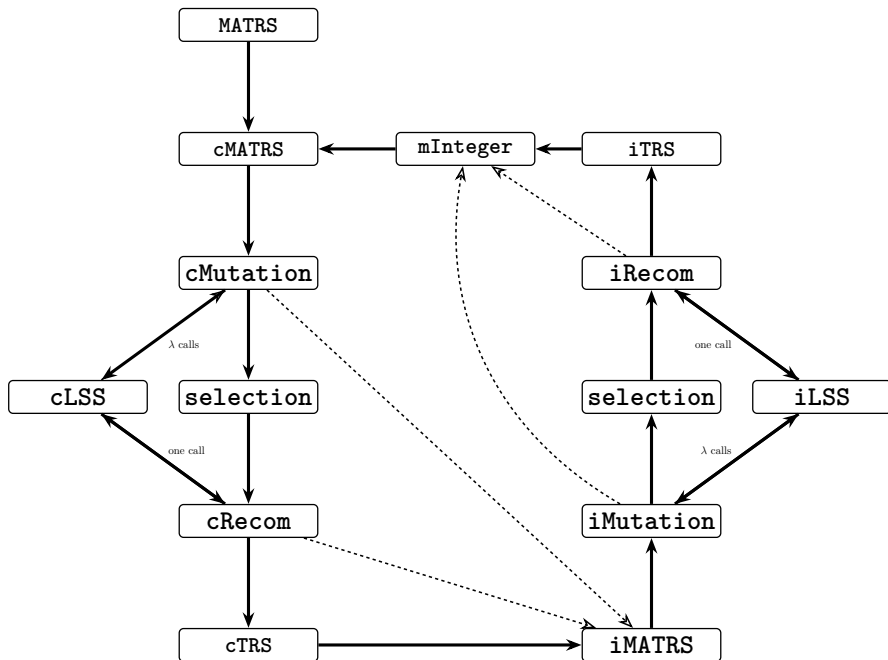
**Integer searches:** **iMATRS** calls **iMutation**, an integer mutation phase, to generate a finite number  $\lambda$  of integer mutation points by performing **iLSS**, an integer line search strategy, along  $\lambda$  integer mutation directions. If none of  $\lambda$  integer mutation points can be a new best point, **iMATRS** performs the selection phase to sort inexact function values at  $\lambda$  integer mutation points in ascending order and select  $0 < \mu < \lambda$  mutation points, distribution directions, and mutation directions as selected information. Then **iRecom**, an integer recombination phase, is called to generate a new best point by performing **iLSS**. If

**iLSS** cannot find a new best point, **iMATRS** calls **iTRS**, an integer trust region strategy, to find such a best point.

**Continuous searches:** **cMATRS** calls **cMutation**, a continuous mutation phase, to generate  $\lambda$  continuous mutation points by performing **cLSS**, a continuous line search strategy, along  $\lambda$  continuous mutation directions. If none of  $\lambda$  continuous mutation points can be a new best point, **cMATRS** performs the selection phase to sort inexact function values at  $\lambda$  continuous mutation points in ascending order and select  $\mu$  mutation points, distribution directions, and mutation directions as selected information. Then **cRecom**, a continuous recombination phase, is called to generate a new best point by performing **cLSS**. If **cLSS** cannot find a new best point, **cMATRS** calls **cTRS**, a continuous trust region strategy, to find such a best point.

**Mixed-integer searches:** To find new best points, **MATRS** calls **mInteger** to perform a mixed-integer line search strategy, called **miLSS**, along a combination direction or its opposite direction, one of which goes into or move down a valley, exactly in one of the spaces of all  $x$ ,  $x_I$ , and  $x_K$  in this order. In fact, **miLSS** is a mixed-integer version of **iLSS** and **cLSS**. **miLSS** reduces to **iLSS** in the space of all  $x_I$  or **cLSS** in the space of all  $x_K$ .

All ingredients of **MATRS** are described below. Both **iRecom** and **cRecom** use selected points obtained from the selection phase to update step sizes and affine scaling matrices.



**Fig. 3:** Flowchart for MATRS. Dash arrows indicate conditional jumps described in the main text.

---

**Algorithm 2** MATRS, *mixed-integer matrix adaptation trust region strategy*

---

**goal:** MATRS performs a new matrix adaptation trust region strategy

---

- 1: **while**  $\text{nf} < \text{nfmax}$  **do**   ▷  $\text{nf}$  denotes the number of function evaluations
  - 2:     • perform **cMATRS** with **cMutation**, **selection**, **cRecom**, and **cTRS**
  - 3:     on the space of all  $x_K$  to find a new best point.
  - 4:     • perform **iMATRS** with **iMutation**, **selection**, **iRecom**, and **iTRS**
  - 5:     on the space of all  $x_I$  to find a new best point.
  - 6:     • perform **mInteger** on exactly one of the spaces of all  $x$ ,  $x_K$ , and  $x_I$
  - 7:     in this order to find a new best point.
  - 8: **end while**
- 

**cMutation** and **iMutation**

- 9: Compute a finite number  $\lambda$  of the distribution and mutation directions.
- 10: Generate  $\lambda$  mutation points and their inexact function values by line searches (**cLSS** and **iLSS**).

**selection**

- 11: Sort directions, points, and their inexact function values obtained from the mutation phase.
- 12: Select some of them with respect to the ascending order of the corresponding function values for the recombination phase.

**cRecom** and **iRecom**

- 13: Compute a new recombination mutation direction.
- 14: Find a new recombination point and its inexact function value by line searches (**cLSS** and **iLSS**).
- 15: Compute a new recombination step size and a new affine scaling matrix using the selected information from the selection phase.

**cTRS** and **iTRS**

- 16: If the recombination point cannot be a new best point, perform trust region strategies (**cTRS** and **iTRS**) to find such a best point.

**mInteger**

- 17: Compute a new combination direction with the goal of going into a valley.
  - 18: Perform the mixed-integer line search **miLSS** along the combination direction to find a new best point.
-

### 3.1 The algorithm

MATRS takes as input the initial point  $x^0$ , the maximum number `nfmax` of inexact function evaluations, and all tuning parameters (which will be discussed in Section 5.1). It returns as output the last best point  $x_{\text{best}}$  and its inexact function value  $\tilde{f}_{\text{best}}$ .

Until an approximate stationary point is found, MATRS repeatedly performs the four phases, continuous mutation, selection, continuous recombination, and continuous trust region of `cMATRS` in the space of all  $x_K$ , the four phases, integer mutation, selection, integer recombination, and integer trust region of `iMATRS` in the space of all  $x_I$ , and the mixed-integer phase by performing `miLSS` along a combination direction or its opposite direction exactly in one of the spaces of all  $x$ ,  $x_K$ , and  $x_I$  in this order.

The selected information in the selection phase are mutation points, distribution directions, and mutation directions. These information are used in the recombination phase to update a new recombination step size and a new affine scaling matrix.

### 3.2 cMutation

This section explains `cMutation` and its components such as continuous distribution and mutation directions, how `cLSS` computes continuous mutation points, and the requirements (such as real initial step sizes, largest allowed real step sizes, and real mutation step sizes) for  $\lambda$  calls to `cLSS` by `cMATRS` in each iteration of MATRS.

The goal of `cMutation` is to generate continuous mutation points, which can be best points. This can be achieved by performing `cLSS` along continuous mutation directions or their opposite directions, which uses extrapolation to leave regions near the saddle point or maximizer. As long as at least one of the mutation points is selected as a new best point, the selection and continuous recombination phases are skipped. In this case, `cMATRS` is actually reduced to `cMutation`, which is a continuous multi-line search due to the  $\lambda$  calls to `cLSS`. Otherwise, if none of the  $\lambda$  continuous mutation points is chosen as a new best point, selection and continuous recombination phases are performed.

The new features of `cMutation` are

- the computation of the continuous distribution directions by `usequence` to be well-distributed, neither too close to each other;
- determining the initial real step sizes  $\alpha_{\text{init}}^{i\ell}$  ( $i = 1, \dots, \lambda$ ) for `cLSS` based on the largest allowed real step sizes  $\bar{\alpha}^{i\ell}$  ( $i = 1, \dots, \lambda$ ), the real recombination

step size  $\sigma_\ell$ , and the list  $\mathbf{a}_K^\ell$  of real mutation step sizes that are neither too small nor too large avoiding line search failure;

- updating  $\mathbf{a}_K^\ell$  in a new way that affects the determination of  $\alpha_{\text{init}}^{i\ell}$ ;
- finding continuous mutation points (possibly best points) with cLSS to leave regions near the saddle point or maximizer.

Given the sample size  $\lambda > 0$ , counter  $i \in \{1, 2, \dots, \lambda\}$  for the number of mutation points, and counter  $\ell \in \{0, 1, 2, \dots\}$  for the number of iterations, we define ingredients of cMutation below.

**Continuous distribution directions:** These directions  $p_{\text{dd}}^{i\ell}$  ( $i = 1, \dots, \lambda$ ) in the  $\ell$ th iteration of MATRS are random directions chosen from the normal distribution  $\mathcal{N}(0, I)$  with zero mean and variance  $I$ . Here  $I$  is an identity matrix.

**Continuous mutation directions:** In the  $\ell$ th iteration of MATRS, the  $i$ th continuous mutation direction

$$p_{\text{md}}^{i\ell} = M_\ell p_{\text{dd}}^{i\ell} \in \mathbb{R}^{|K|}$$

is computed, where  $M_\ell \in \mathbb{R}^{|K| \times |K|}$  is the  $\ell$ th affine scaling matrix (updated in Section 3.5), and  $p_{\text{dd}}^{i\ell}$  is the  $i$ th continuous distribution direction. Indeed,  $p_{\text{md}}^{i\ell}$  ( $i = 1, \dots, \lambda$ ) are chosen from  $\mathcal{N}(0, M_\ell M_\ell^T)$ .

Denote by  $A_{:,k}$  the  $k$ th column of the matrix  $A$ . Then,  $A_{:,i:k}$  includes the columns between the  $i$ th and  $k$ th columns of  $A$ .

**Changing continuous distribution directions:** If cLSS cannot reduce the inexact function value along continuous mutation directions  $p_{\text{md}}^{i\ell}$  for all  $i \in \{1, 2, \dots, \lambda\}$ , the set of random directions should be changed in a new randomized way after the continuous recombination phase is performed. The goal of **usequence** is to generate a sequence of finite continuous random vectors in the  $K$ -dimensional unit cube plus a new combination direction that for each leading subsequence, arbitrary vectors are not too close to each other. To enrich **usequence**, we replace the first column of the  $|K| \times N$  random reservoir matrix  $R^\ell$  by our new combination direction

$$p_K^{\text{init},\ell} = x_K^{\text{best},\ell} - x_K^{\text{rmd},\ell-1}, \quad p_K^{\text{init},\ell} = \text{sc} \frac{p_K^{\text{init},\ell}}{\|p_K^{\text{init},\ell}\|_\infty},$$

where  $x_K^{\text{rmd},\ell-1}$  is the recombination point evaluated in the  $(\ell - 1)$ th iteration of MATRS and  $x_K^{\text{best},\ell}$  is the  $\ell$ th best point found by cMutation,  $N \geq \lambda$  is a positive tuning parameter, and **sc** is a positive tuning parameter. There is

a good chance to find decreases in the inexact function value by performing cLSS along  $p_K^{\text{init},\ell}$ , leaving points with large inexact function values and going into or moving down a valley.

cMATRS finds a new set

$$D^{\ell+1} = \kappa * \text{usequence}(|K|, N, z, p_K^{\text{init},\ell})$$

of real distribution directions. Here  $\kappa$  is updated in each iteration of cMATRS. If no best point is found and  $\kappa < \kappa_{\max}$ , cMATRS increases  $\kappa$  by one to generate a new set of continuous distribution directions with the goal of finding new best points in a larger neighborhood of the old best point; otherwise, it sets  $\kappa = 1$  (in this case, distribution directions are selected from  $\mathcal{N}(0, I)$ ), where the tuning parameter  $\kappa_{\max} > 1$  is an upper bound for  $\kappa$ .

**Continuous mutation points:** At the  $\ell$ th iteration of MATRS, we perform cLSS along the  $i$ th continuous distribution direction  $p_{\text{md}}^{i\ell}$  or its opposite directions to obtain the  $i$ th continuous mutation point

$$x_K^{i\ell} = x_K^{\text{best}} + \alpha^{i\ell} p_{\text{md}}^{i\ell}, \quad (4)$$

where  $\alpha^{i\ell}$  is initially the initial real step size  $\alpha_{\text{init}}$  and updated by cLSS. It is a scaling factor for the mutation phase in the  $\ell$ th iteration of MATRS.

**Update of the vector  $\mathbf{a}_K^\ell$  of real step sizes:** Let the  $\lambda \times 1$  vector  $\mathbf{a}_K^\ell$  be the list of real step sizes of cMutation used to update the initial real step size  $\alpha_{\text{init}}^{i\ell}$  for  $i = 1, 2, \dots, \lambda$  (see (6), below). The vector  $\mathbf{a}_K^{i\ell}$  is initially a tuning vector with real components ( $\mathbf{a}_K^{i0} \geq 1$  for  $i = 1, 2, \dots, n$ ) and updated depending on whether or not decreases in the inexact function values are found at the mutation points. We now describe how this vector is updated. After cLSS is terminated to find the  $i$ th continuous mutation point in the  $\ell$ th iteration of MATRS, the corresponding step size of a point with the lowest inexact function value among all points evaluated in extrapolation is stored in  $\mathbf{a}_K^{i\ell}$  if possible. Otherwise, unlike [23] with  $\mathbf{a}_K^{i\ell} = \mathbf{a}_K^{i0}/\nu$  ( $\nu > 1$  is a given tuning parameter),  $\mathbf{a}_K^{i\ell} = \sigma_\ell$  is stored for use in the next iteration. The reason for this new choice is that  $\sigma_\ell$  does not become too small, avoiding getting stuck before an approximate stationary point is found. This is a new property of our algorithm, which is against line search failures.

**Largest allowed real step sizes  $\bar{\alpha}^{i\ell}$  needed for cLSS:** Before cLSS is executed, it requires  $\bar{\alpha}^{i\ell} \geq 1$  for each  $i = 1, 2, \dots, \lambda$  at the  $\ell$ th iteration of MATRS, while maintaining feasibility. To find  $\bar{\alpha}^{i\ell}$ , we denote  $p_j := (p_{\text{md}}^{i\ell})_j$  and

compute

$$\overline{K} := \{j \in K \mid p_j > 0, x_j^{\text{best}} < \overline{x}_j\}, \quad \overline{\beta} := \min\{(\overline{x}_j - x_j^{\text{best}})/p_j \mid j \in \overline{K}\};$$

if  $\overline{\beta}$  is empty,  $\overline{\beta} = \infty$ , and

$$\underline{K} := \{j \in K \mid p_j < 0, x_j^{\text{best}} > \overline{x}_j\}, \quad \underline{\beta} := \min\{(x_j - x_j^{\text{best}})/p_j \mid j \in \underline{K}\};$$

if  $\underline{\beta}$  is empty,  $\underline{\beta} = \infty$ . Then, the largest allowed real step size

$$\overline{\alpha}^{i\ell} = \min(\overline{\beta}, \underline{\beta}) \quad (5)$$

is computed.

**Initial real step sizes  $\alpha_{\text{init}}^{i\ell}$  for cLSS.** Given the recombination step size  $\sigma_\ell$  (computed by (16) below), for each  $i = 1, 2, \dots, \lambda$  at the  $\ell$ th iteration of MATRS, we first compute the initial real step size by the new formula

$$\alpha_{\text{init}}^{i\ell} := \min\left(\overline{\alpha}^{i\ell}, \sqrt{\sigma_\ell \mathbf{a}_K^{i\ell}}\right) \quad (6)$$

whose goal is to be neither too small nor too large to avoid line search failures. In the  $\ell$ th iteration of MATRS, cLSS now takes  $\alpha_{\text{init}}^{i\ell}$ ,  $\overline{\alpha}^{i\ell}$ , and  $p_{\text{md}}^{i\ell}$  as given input and performs continuous extrapolation along  $p_{\text{md}}^{i\ell}$  or its opposite direction to find the  $i$ th mutation point, which hopefully can be a new best point. If this mutation point cannot be selected as a new best point, the selection phase and the continuous recombination phase are performed to find a new recombination point in the hope of being a new best point.

**Computation of real step sizes and trial points inside cLSS:** We describe how to update the step sizes within the extrapolation phase of cLSS. At the beginning,  $\alpha = \alpha_{\text{init}}^{i\ell}$  is chosen,  $p_{\text{md}}^{i\ell} \in \mathbb{R}^{|K|}$  is given, and the initial continuous trial point

$$x_K^0 = x_K^{\text{trial}} = x_K^{\text{best}} + \alpha p_{\text{md}}^{i\ell}$$

and its inexact function value  $f^0 = \tilde{f}^{\text{trial}} := \tilde{f}(x^{\text{trial}})$  are calculated. If the conditions  $\tilde{f}^{\text{trial}} < \tilde{f}^{\text{best}}$  and  $\alpha < \overline{\alpha}^{i\ell}$  hold, given a tuning factor  $\nu > 1$ , the new continuous trial point

$$x_K^{\text{trial}} = x_K^{\text{best}} + \min\left(\overline{\alpha}^{i\ell}, \nu\alpha\right) p_{\text{md}}^{i\ell} \quad (7)$$



and its inexact function value  $\tilde{f}^{\text{trial}}$  are calculated. As long as the conditions  $\alpha < \bar{\alpha}^{i\ell}$  and  $\tilde{f}^{\text{trial}} < \tilde{f}^{\text{best}}$  hold, an extrapolation step along  $p_{\text{md}}^{i\ell}$  or its opposite direction is performed by expanding the real step size to

$$\alpha' = \min(\bar{\alpha}^{i\ell}, \nu\alpha) \quad (8)$$

and computing the new continuous trial point  $x_K^{\text{trial}} = x_K^{\text{best}} + \alpha' p_{\text{md}}^{i\ell}$  and its inexact function value  $\tilde{f}^{\text{trial}}$ . Both (7) and (8) are defined as in [23].

After the extrapolation in cLSS is finished, a point with the lowest inexact function values among all trial points found by extrapolation in the  $\ell$ th iteration of MATRS is chosen as a new best point and, as described above, the corresponding step size is stored in  $\mathbf{a}_K^{i\ell}$ .

If extrapolation cannot be performed along  $p_{\text{md}}^{i\ell}$  for  $i = 1, \dots, \lambda$  in the  $\ell$ th iteration of MATRS,  $\lambda$  trial points are chosen as the mutation points and then the selection and recombination phases are performed.

### 3.3 iMutation

This section explains iMutation and its components such as integer distribution and integer mutation directions, how iLSS computes integer mutation points, and the requirements (such as integer initial step sizes, largest allowed integer step sizes, and integer mutation step sizes) for  $\lambda$  calls to iLSS by iMATRS in each iteration of MATRS. iMutation has the same structure as cMutation and the same goal, but with differences in distribution directions and updating step sizes.

The goal of iMutation is to generate integer mutation points, which can be new best points. This can be achieved by performing iLSS along integer mutation directions or their opposite directions, which uses extrapolation to leave regions near the saddle point or maximizer. As long as at least one of the integer mutation points is chosen as a new best point, the selection and integer recombination phases are skipped. In this case, iMATRS is actually reduced to iMutation, which is an integer multi-line search due to the  $\lambda$  calls to iLSS. Otherwise, none of the  $\lambda$  integer mutation points is chosen as a new best point, the selection and integer recombination phases are performed, which are discussed in the next sections.

The new features of iMutation are

- the computation of the integer distribution directions by **usequence** to be well-distributed, neither too close to each other;

- determining the initial integer step sizes  $\alpha_{\text{init}}^{i\ell}$  ( $i = 1, \dots, \lambda$ ) for **iLSS** based on the largest allowed integer step sizes  $\bar{\alpha}^{i\ell}$  ( $i = 1, \dots, \lambda$ ), the integer recombination step size  $\sigma_\ell$ , and the list  $\mathbf{a}_I^\ell$  of integer mutation step sizes that are neither too small nor too large avoiding line search failure;
- updating  $\mathbf{a}_I^\ell$  in a new way that affects the determination of  $\alpha_{\text{init}}^{i\ell}$ ;
- finding integer mutation points with **iLSS** to leave regions near the saddle point or maximizer.

**Integer distribution directions:** Integer distribution directions  $p_{\text{dd}}^{i\ell} \in \mathbb{R}^{|I|}$  are chosen from a set of permuted coordinate directions, unlike **cMutation**, which selects continuous distribution directions from the normal distribution.

**Changing integer distribution directions:** If **iLSS** cannot reduce the inexact function value along the integer mutation directions  $p_{\text{md}}^{i\ell}$  (for  $i = 1, 2, \dots, \lambda$ ) in the  $\ell$ th iteration of **MATRS**, the set of integer random directions should be changed in a new way after performing the integer recombination phase. The idea is to generate a sequence of finite integer random vectors plus a new integer combination direction in the  $I$ -dimensional unit cube such that for each leading subsequence, arbitrary vectors are neither too close to each other. The new integer combination direction

$$p_I^{\text{init},\ell} = x_I^{\text{best},\ell} - x_I^{\text{rnd},\ell-1}, \quad p_I^{\text{init},\ell} = \text{sc} \left[ p_I^{\text{init},\ell} / \|p_I^{\text{init},\ell}\|_\infty \right]$$

is defined in a new way, where  $x_I^{\text{rnd},\ell-1}$  is the recombination point evaluated in the  $(\ell - 1)$ th iteration of **MATRS** and  $x_I^{\text{best},\ell}$  is the  $\ell$ th best point found by **iMutation** and **sc** is a positive integer tuning parameter. Many small improved steps have accumulated in this integer combination direction, starting from a point with a small inexact function value to a point with an even smaller inexact function value, and moving into a valley. Thus, further progress of the inexact function value can be expected as **iMATRS** continues along this integer combination direction. The goal of **usequence** is to generate integer directions with the largest minimum distance, so **iLSS** has a good chance of finding mutation points with small inexact function values by going along these directions. This differs from **LIUZZI** et al. [23], which use Halton sequences to generate integer directions. To enrich **usequence**, we replace the first column of the  $|I| \times N$  random matrix  $R^\ell$  by the new combination direction  $p_I^{\text{init},\ell}$ . There is a good chance to find decreases in the inexact function value by performing **iLSS** along  $p_I^{\text{init},\ell}$ , leaving points with large inexact function values and going into or moving down a valley.

iMATRS finds a new set

$$D^{\ell+1} = \text{usequence}(|I|, N, \kappa z, p_I^{\text{init}, \ell})$$

of integer distribution directions. Here  $N \geq \lambda$  is a positive tuning parameter,  $\kappa$  is updated in each iteration of iMATRS and  $z_i \geq 1$  for  $i \in I$  are the tuning parameters to restrict the components of the well-defined distributed directions. If no new best point is found and  $\kappa < \kappa_{\max}$ , iMATRS increases  $\kappa$  by one to generate a new set of integer distribution directions with the goal of finding new best points in a larger neighborhood of the old best point; otherwise, it sets  $\kappa = 1$  (in which case permuted distribution directions are used), where  $\kappa_{\max} > 1$  is an upper bound for  $\kappa$  and a tuning parameter.

**Update of the vector  $\mathbf{a}_I^\ell$  of integer step sizes in iMutation:** Let the  $\lambda \times 1$  vector  $\mathbf{a}_I^\ell$  be the list of integer step sizes of iMutation used to update the initial integer step size  $\alpha_{\text{init}}^{i\ell}$  for  $i = 1, 2, \dots, \lambda$  (see (9), below). Unlike [23], we do not reduce the  $i$ th component of the vector  $\mathbf{a}_I^\ell$  in the  $\ell$ th iteration of MATRS by  $\mathbf{a}_I^{i\ell} = \lfloor \mathbf{a}_I^{i\ell} / \nu \rfloor$ , where  $\nu > 1$  is a tuning parameter, if iLSS cannot find a new best point along the  $i$ th integer mutation direction  $p_{\text{md}}^{i\ell}$ . In this case, like cMutation,  $\mathbf{a}_I^{i\ell} = \sigma_\ell$  is saved. This choice avoids tiny step sizes and so null steps in the presence of large noise. Here  $\sigma_\ell$  is the  $\ell$ th integer step size, which is computed by (27) below. Otherwise, if iLSS finds a new best point (as a result of extrapolation) along the  $i$ th integer mutation direction  $p_{\text{md}}^{i\ell}$ , the corresponding step size is saved in  $\mathbf{a}_I^{i\ell}$  like [23].

**Update of initial integer step sizes  $\alpha_{\text{init}}^{i\ell}$  for iLSS:** The initial integer step size  $\alpha_{\text{init}}^{i\ell}$  is computed for each  $i \in \{1, 2, \dots, \lambda\}$  in the  $\ell$ th iteration of MATRS unlike cMutation. In this case, we compute the initial step size by the new formula

$$\alpha_{\text{init}}^{i\ell} := \min \left( \left\lfloor \sqrt{\sigma_\ell \mathbf{a}_I^{i\ell}} \right\rfloor, \max(1, \lfloor \bar{\alpha}^{i\ell} \rfloor) \right), \quad (9)$$

where  $\bar{\alpha}^{i\ell}$  is computed by (5) and  $\sigma_\ell$  is computed by (27) below.

**Update of integer step sizes and trial points inside iLSS:** Like [23], we construct the integer version

$$x_I^{\text{trial}} = x_I^{\text{best}} + \alpha'' p_{\text{md}}^{i\ell}, \quad \alpha'' = \min \left( \bar{\alpha}^{i\ell}, \nu \alpha \right) \quad (10)$$

of the two formulas (7) and (8), respectively.

### 3.4 Selection

This section discusses the selection phase whose goal is to sort the points and directions obtained from the integer and continuous mutation phases such that points with low inexact functions values and the corresponding directions are selected for use in the recombination phase.

Let  $x^{i\ell}$  ( $i = 1, \dots, \lambda$ ) be the sequence of (integer and continuous) mutation points found in the  $\ell$ th iteration of MATRS and denote by  $\mu$  the number of selected points in the recombination phase. The inexact function values  $\tilde{f}(x^{i\ell})$  ( $i = 1, \dots, \lambda$ ) of mutation points  $x^{i\ell}$  ( $i = 1, \dots, \lambda$ ) are sorted in ascending order

$$\tilde{f}(x^{\pi 1, \ell}) \leq \tilde{f}(x^{\pi 2, \ell}) \leq \tilde{f}(x^{\pi \mu, \ell}) \leq \tilde{f}(x^{\pi(\mu+1), \ell}) \leq \dots \leq \tilde{f}(x^{\pi \lambda, \ell}),$$

where  $\pi$  is a permutation of  $\{1, 2, \dots, \lambda\}$ . Then, accordingly the distribution directions  $p_{\text{dd}}^{\pi i, \ell}$  ( $i = 1, \dots, \lambda$ ) and the mutation directions  $p_{\text{md}}^{\pi i, \ell}$  ( $i = 1, \dots, \lambda$ ) are obtained. Finally, we select the  $\mu$  best information

$$x^{\pi i, \ell}, \tilde{f}(x^{\pi i, \ell}), p_{\text{dd}}^{\pi i, \ell}, p_{\text{md}}^{\pi i, \ell} \quad (i = 1, \dots, \mu)$$

for computing new recombination points.

### 3.5 cRecom

This section discusses cRecom and its main components such as recombination direction and point and how to calculate them.

The goal of cRecom is to find a new continuous recombination point in the hope of being a new best point. It computes the continuous recombination mutation direction and the initial real and maximum allowed real step sizes, and then performs cLSS along the continuous recombination mutation direction or its opposite direction to compute a new continuous recombination point motivated by [19].

The new features of cRecom are

- the scale of the weights of the recombination direction in a randomized way with the goal of reordering a fair sort in the selection phase due to noise;
- the determination of the initial real step size for cLSS that is neither too small nor too large to perform successful extrapolation;

**Continuous recombination mutation direction:** We compute the continuous recombination mutation direction

$$p_{\text{rmd}}^\ell := \sum_{i=1}^{\mu} \bar{w}_i p_{\text{md}}^{\pi i, \ell} \in \mathbb{R}^{|K|} \quad (11)$$

and the continuous recombination distribution direction

$$p_{\text{rdd}}^\ell := \sum_{i=1}^{\mu} \bar{w}_i p_{\text{dd}}^{\pi i, \ell} \in \mathbb{R}^{|K|} \quad (12)$$

with real components. Here  $\bar{w} := \beta \odot w$  is the product of the weighted vector  $w$  satisfying

$$\sum_{i=1}^{\lambda} w_i = 1 \quad \text{and} \quad w_1 \geq w_2 \geq \dots \geq w_\mu > 0 = w_{\mu+1} = \dots = w_\lambda \quad (13)$$

component-wise in the random scaling vector

$$\beta \sim \mathcal{N}(0, 2I - 1), \quad \beta := \beta / \|\beta\|.$$

Indeed, by scaling the weights  $w_i$  there is a good chance to reorder a fair sort in the selection phase due to a high noise. This is a new property of our algorithm. In Section 4, we compute numerically  $w_i$  for  $i = 1, \dots, \lambda$  satisfying (13).

**Update of affine scaling matrix:** As in [4], we update the affine scaling matrix

$$M_{\ell+1} := \left(1 - \frac{c_1 + c_\mu}{2}\right) M_\ell + \frac{c_1}{2} M_\ell P_\ell^\sigma (P_\ell^\sigma)^T + \frac{c_\mu}{2} \sum_{i=1}^{\mu} w_i p_{\text{md}}^{\pi i, \ell} (p_{\text{dd}}^{\pi i, \ell})^T, \quad (14)$$

where  $0 < c_\mu \leq 1$  is a learning rate for updating  $M_{\ell+1}$  and  $c_1 \leq 1 - c_\mu$  is a learning rate for the rank-one-update of  $M_{\ell+1}$ . Here, the evolution path

$$P_0^\sigma := 0, \quad P_\ell^\sigma := (1 - c_\sigma) P_{\ell-1}^\sigma + \bar{c}_\sigma p_{\text{rdd}}^\ell, \quad \text{for } \ell \geq 1 \quad (15)$$

is defined, where the normalization constant

$$\bar{c}_\sigma := \sqrt{c_\sigma(2 - c_\sigma)\mu_w}$$

using the variance effective selection mass

$$\mu_w := \frac{\|w\|_1^2}{\|w\|_2^2} = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \in [1, \mu]$$

is defined; see Section 4 for how  $c_1$ ,  $c_\mu$ , and  $c_\sigma$  are numerically computed.

In (14), the first term includes the previous information and accumulates the information, while the second term is the rank-one update whose goal is to increase the probability of  $p_{\text{dd}}^{\pi i, \ell}$  ( $i = 1, \dots, \mu$ ) for the next iteration, by maximizing the log-likelihood of  $p_{\text{dd}}^{\pi i, \ell}$  ( $i = 1, \dots, \mu$ ), and the third term is the rank- $\mu$  update, whose goal is to take the mean of the estimated affine scaling matrices from all iterations.

The first goal of the evolution path computed by (15) and used in the second term (14) is to remedy losing the sign of  $p_{\text{dd}}^{\pi i, \ell}$  ( $i = 1, \dots, \mu$ ) in the third term of (15) because

$$p_{\text{dd}}^{\pi i, \ell} (p_{\text{dd}}^{\pi i, \ell})^T = -p_{\text{dd}}^{\pi i, \ell} (-p_{\text{dd}}^{\pi i, \ell})^T$$

and  $p_{\text{md}}^{\pi i, \ell} = M_\ell p_{\text{dd}}^{\pi i, \ell}$ . Its second goal is to update the recombination step size (see (16) below). Note that  $p_{\text{md}}^{\pi i, \ell}$  has been computed before in the mutation phase and here it only reuses, leading to  $\mathcal{O}(n^2)$  operations due to the vector-matrix products.

As a result, all three terms of (14) have different advantages and cause the affine scaling matrix behaves well in practice, compared to the rank-one update and rank- $\mu$  update.

**Update of real recombination step size:** We update the real step size

$$\sigma_{\ell+1} := \sigma_\ell \exp \tau_\ell \quad \text{with } \tau_\ell := d_\sigma^{-1} c_\sigma \left( e^{-1} \|P_\ell^\sigma\| - 1 \right) \quad (16)$$

and project it into  $[\sigma_{\min}, \sigma_{\max}]$ . Here,  $e_\sigma$  is an approximate value of the expected value  $\mathbf{E}(\|u\|)$  of the norm of the vector  $u \sim \mathcal{N}(0, I)$ , the constant  $0 < \sigma_{\max} < \infty$  is a maximum value for  $\sigma_t$ ,  $0 < \sigma_{\min} < 1$  is a minimum value for  $\sigma_t$ ,  $c_\sigma \leq 1$  is a learning rate for the cumulation for the step size,  $d_\sigma \approx 1$  is a damping parameter (cf. [13, Section 4]). Section 4 discusses how to numerically compute  $d_\sigma$ ,  $c_\sigma$ , and  $e_\sigma$ .

After the computation of the continuous recombination mutation direction and the updates of the real recombination step size  $\sigma_\ell$  and the affine scaling matrix  $M_\ell$ , cLSS is performed along the continuous recombination mutation direction  $p_{\text{dd}}^{\text{rmd}, \ell}$  or possibly its opposite direction. Before this run, we discuss how to find the  $\ell$ th initial real step size  $\alpha_{\text{init}}^\ell$  and the  $\ell$ th largest allowed real step size  $\bar{\alpha}^\ell$  such that it does not violate feasibility.

**The initial and largest real step sizes for cLSS:** To avoid too small and too large step sizes, we update the initial real step size

$$\alpha_{\text{init}}^{\ell} := \min \left( \sigma_{\ell}, \bar{\alpha}^{\ell} \right), \quad (17)$$

where the largest allowed step size  $\bar{\alpha}^{\ell}$  is computed by (5) with the difference that, in the computation of  $\underline{\beta}$  and  $\bar{\beta}$  in (5),  $p_{\text{imd}}^{i\ell}$  is replaced by  $p_{\text{rmd}}^{\ell}$ . Here  $\sigma_{\ell}$  is from (16). Indeed, (17) does not use the mutation step size vector  $\mathbf{a}_K^{\ell}$  unlike (6).

**A new recombination point:** To get this point, using  $\alpha^{\ell} = \alpha_{\text{init}}^{\ell}$  and  $\bar{\alpha}^{\ell}$ , cLSS is performed along  $p_{\text{rmd}}^{\ell} \in \mathbb{R}^{|K|}$  or its opposite direction, hopefully resulting in

$$x_K^{\text{best},\ell+1} = x_K^{\text{best},\ell} + \alpha^{\ell} p_{\text{rmd}}^{\ell}.$$

Here  $\alpha^{\ell}$  is the corresponding step size of the point  $x_K^{\text{best},\ell+1}$  with the lowest inexact function value  $\tilde{f}(x^{\text{best},\ell+1})$  among all evaluated points by extrapolation. If extrapolation cannot be performed along  $p_{\text{rmd}}^{\ell}$ , the new evaluated recombination point is rejected to be a new best point. Hence, cMATRS calls cTRS to find a new best point.

### 3.6 cTRS

This section discusses cTRS to find a new best point when both the continuous mutation and recombination phases are not able to find such a point.

The goal of cTRS is to avoid large steps, which are one of the causes of the failure of cLSS, and find new best points in regions close to the old best point that may not be searched by cLSS in the mutation and recombination phases.

The new features of cTRS are

- the use of recombination step sizes (neither too small nor too large) in the computation of the initial trust region radius in each call to cTRS by cMATRS to overcome the sensitivity of choosing this initial radius;
- the use of the product of the affine scaling matrix and its transpose as a cheap approximation to the Hessian matrix of the model function of the trust region subproblem.

cTRS works until the trust region radius is not below a given threshold  $0 < \Delta_{\text{min}} < 1$ . This is against getting stuck before an approximate stationary point is found when the trust region radius is too small. Indeed, this is one main difference of cTRS with other trust region methods. In each call to cTRS by

cMATRS, cTRS chooses the real initial trust region radius  $\Delta = \min(\Delta_{\max}, \sigma_\ell)$  (here  $\sigma_\ell$  is from (16) and  $0 < \Delta_{\max} < \infty$  is a tuning parameter) and repeatedly performs the following steps:

(CT<sub>1</sub>) **Trust region subproblem:** We define the trust region subproblem

$$\begin{aligned} \min \mathcal{Q}(p) &= \tilde{g}_{\text{best}}^T p + \frac{1}{2} p^T \tilde{G}_{\text{best}} p \\ \text{s.t. } \|p\| &\leq \Delta. \end{aligned} \quad (18)$$

Here as in HUYER & NEUMAIER [15] the approximate gradient vector  $\tilde{g}_{\text{best}}$  is obtained by fitting and the approximate symmetric Hessian matrix

$$\tilde{G}_{\text{best}} = M^T M$$

is chosen as a new choice without additional cost.

(CT<sub>2</sub>) **Solving trust region subproblem:** We define the continuous Cauchy step

$$p_{\text{ca}} := -t^* \tilde{g}_{\text{best}}, \quad t^* := \operatorname{argmin}\{\mathcal{Q}(-t\tilde{g}_{\text{best}}) \mid t \geq 0, \|t\tilde{g}_{\text{best}}\| \leq \Delta\} \quad (19)$$

and solve the trust region subproblem (18) in such a way that the conditions

$$\|p\| \leq \Delta \quad \text{and} \quad \mathcal{Q}(p) \leq \mathcal{Q}(p_{\text{ca}}) \quad (20)$$

are satisfied. Indeed, the continuous approximate Newton direction

$$p_{\text{an}} = -\tilde{G}_{\text{best}}^{-1} \tilde{g}_{\text{best}}$$

is calculated. This direction is accepted as the solution  $p = p_{\text{an}}$  of (18) forcing (20) if it is within the trust region. Otherwise, the continuous scaled approximate steepest descent step

$$p_{\text{sd}} := -\frac{\tilde{g}_{\text{best}}^T \tilde{g}_{\text{best}}}{\tilde{g}_{\text{best}}^T \tilde{G}_{\text{best}} \tilde{g}_{\text{best}}} \tilde{g}_{\text{best}} \quad (21)$$

is calculated. If it is outside the trust region, an estimated solution of (18) is the continuous Cauchy step  $p = p_{\text{ca}}$  computed by (19); otherwise, it is the continuous dogleg step

$$p = p_{\text{dg}} := p_{\text{dg}}(t) = p_{\text{sd}} + t(p_{\text{an}} - p_{\text{sd}}), \quad (22)$$

where  $t$  is obtained by solving the equation  $\|p_{\text{dg}}(t)\| = \Delta$ .



(CT<sub>3</sub>) **Trust region trial point:** We compute the continuous trial point

$$x_K^{\text{trial}} = \min \left( \bar{x}_K, \max \left( \underline{x}_K, x_K^{\text{best}} + \alpha p \right) \right)$$

and its inexact function value  $\tilde{f}^{\text{trial}} := \tilde{f}(x^{\text{trial}})$ .

(CT<sub>4</sub>) **Trust region condition:** Given the tuning parameter  $0 < \eta < \frac{1}{4}$ , if the sufficient descent condition

$$\rho_{\text{trial}} |\rho_{\text{trial}} - 1| > \eta \quad \text{with} \quad \rho_{\text{trial}} = \rho(x^{\text{trial}}) := (\tilde{f}^{\text{trial}} - \tilde{f}^{\text{best}}) / \tilde{g}_{\text{best}}^T p \quad (23)$$

is satisfied, the current iteration of cTRS is called **successful** and  $x_{\text{trial}}$  is accepted as a new best point. Then, using the tuning parameter  $c > 1$ , we expand the real trust region radius to

$$\Delta := c \max(\Delta, \|p\|_{\infty}). \quad (24)$$

Otherwise, the current iteration of cTRS is called **unsuccessful**. In this case, the real trust region radius is reduced to

$$\Delta := c^{-1} \min(\Delta, \|p\|_{\infty}). \quad (25)$$

As the iterations of cTRS are unsuccessful and  $\Delta > \Delta_{\min}$ ,  $\Delta$  is reduced to (25), the trust region subproblem is solved, and a new continuous trial point is generated. cTRS may find a new best point. The condition (23) was suggested by KIMIAEI [16] for bound-constrained ill-conditioned problems.

### 3.7 iRecom

This section discusses iRecom and its main components such as recombination direction and point and how to calculate them.

The goal of iRecom is to find a new best point. It computes the integer recombination mutation direction and the initial integer and maximum allowed integer step sizes, and then performs iLSS along the integer recombination mutation direction or its opposite direction in the hope of finding a new best point.

The new features of iRecom are

- the scale of the weights of the integer recombination direction in a randomized way with the goal of reordering a fair sort in the selection phase due to noise;
- the determination of the initial integer step size for iLSS that is neither too small nor too large to perform successful extrapolation.

**Integer recombination mutation direction:** We round the non-integer components of the continuous recombination mutation direction computed by (11) to

$$p_{\text{rmd}}^\ell := \left\lceil \sum_{i=1}^{\mu} \bar{w}_i p_{\text{md}}^{\pi_i, \ell} \right\rceil \in \mathbb{R}^{|I|}, \quad (26)$$

which is our integer recombination mutation direction. In this phase,  $p_{\text{rdd}}^\ell \in \mathbb{R}^{|I|}$  is computed as in the continuous case of (12), since it is used to update the evolution path, but is not intended to round the entries of the affine scaling matrix  $M_\ell \in \mathbb{R}^{|I| \times |I|}$  before computing the mutation directions  $p_{\text{md}}^{i\ell} = M_\ell p_{\text{dd}}^{i\ell}$  in the next mutation phase. When calculating the mutation directions, the non-integer components (if any) are rounded to integers.

**Update of affine scaling matrix:** We compute the affine scaling matrix updated by (14) without rounding its entries to integer. In fact as mentioned in Section 3.3 in the computation of integer mutation directions non-integer entries of these directions are rounded to integers.

**Update of integer recombination step size:** As in the continuous case, we compute the real step size by (16) and then round it to integer, resulting in our integer recombination step size

$$\sigma_{\ell+1} := \max \left( 1, \left\lfloor \sigma_\ell \exp \tau_\ell \right\rfloor \right), \quad (27)$$

where  $\tau_\ell$  is from (16).

**A new integer recombination point.** To get this point, using the initial integer  $\alpha^\ell = \alpha_{\text{init}}^{i\ell}$  and the integer largest allowed step size  $\bar{\alpha}^{i\ell}$ , iLSS is performed along  $p_{\text{rmd}}^\ell \in \mathbb{R}^{|I|}$  or its opposite direction, hopefully resulting in

$$x_I^{\text{best}, \ell+1} = x_I^{\text{best}, \ell} + \alpha^\ell p_{\text{rmd}}^\ell.$$

Here  $\alpha^\ell$  is the corresponding step size of the point  $x_I^{\text{best}, \ell+1}$  with the lowest inexact function value  $\tilde{f}(x^{\text{best}, \ell+1})$  among all evaluated points by the integer extrapolation.

If extrapolation cannot be performed along  $p_{\text{rmd}}^\ell$ , the new evaluated integer recombination point is rejected to be a new best point. Hence, iMATRS calls iTRS to find a new best point.

### 3.8 iTRS

This section discusses iTRS to find a new best point when both the integer mutation and recombination phases are not able to find such a point.

The goal of **iTRS** is to avoid large steps, which are one of the causes of the failure of **iLSS**, and find new best points in regions close to the old best point that may not be searched by **iLSS** in the integer mutation and recombination phases.

The new features of **iTRS** are

- the use of integer recombination step sizes (neither too small nor too large) as the initial trust region radius in each call to **iTRS**;
- the use of the product of the affine scaling matrix and its transpose as a cheap approximation to the Hessian matrix of the model function of the trust region subproblem;
- the transformation of the trust region subproblems into bound-constrained integer least squares problems.

If **iLSS** cannot find a new best point in the integer mutation and recombination phases, **iMATRS** uses **iTRS** with the goal of avoiding large steps and in the hope of finding a new best point. **iTRS** works until the integer trust region radius is not below one and iterations are unsuccessful. In each call to **iTRS** by **iMATRS**, **iTRS** chooses the initial integer trust region radius  $\Delta = \min(\Delta_{\max}, \max(\Delta_{\min}, \sigma_\ell))$  (the integer  $\sigma_\ell$  is from (27) and  $0 < \Delta_{\min} < \Delta_{\max} < \infty$  are tuning parameters) and then repeatedly performs the following steps:

(IT<sub>1</sub>) **Trust region subproblem:** By defining  $r := -M^{-T}\tilde{g}_{\text{best}}$  and since

$$\tilde{g}_{\text{best}}^T p + \frac{1}{2} p^T \tilde{G}_{\text{best}} p = \frac{1}{2} \|Mp - r\|_2^2 - \frac{1}{2} \|r\|_2^2 \quad (\text{the term } -\frac{1}{2} \|r\|_2^2 \text{ is constant}), \quad (28)$$

the trust region subproblem (18) with the constraint  $x_I^{\text{best}} + p \in \mathbf{x}$  is converted to the bound-constrained integer least squares problem

$$\begin{aligned} \min \quad & \frac{1}{2} \|Mp - r\|_2^2 \\ \text{s.t.} \quad & \|p\| \leq \Delta, \quad p \text{ integral}, \\ & x_I^{\text{best}} + p \in \mathbf{x}. \end{aligned} \quad (29)$$

In (28), the approximate gradient vector  $\tilde{g}_{\text{best}}$  is obtained by fitting as in HUYER & NEUMAIER [15] and the approximate symmetric Hessian matrix  $\tilde{G}_{\text{best}} = M^T M$  is chosen without additional cost.

(IT<sub>2</sub>) **Solving trust region subproblem:** We solve the bound-constrained integer least squares problem (29) by a variant of Schnorr–Euchner search [6, 11].

(IT<sub>3</sub>) **Trust region trial point:** We compute the integer trial point

$$x_I^{\text{trial}} = \min \left( \bar{x}_I, \max \left( \underline{x}_I, x_I^{\text{best}} + \alpha p \right) \right)$$

and its inexact function value  $\tilde{f}^{\text{trial}} := \tilde{f}(x^{\text{trial}})$ .

(IT<sub>4</sub>) **Trust region condition:** Given the tuning parameter  $0 < \eta < \frac{1}{4}$ , if the sufficient descent condition (23) is satisfied, the current iteration of iTRS is called **successful** and  $x_{\text{trial}}$  is accepted as a new best point. Then, iTRS ends. Otherwise, the current iteration of iTRS is called **unsuccessful**. Given the integer tuning parameters  $\bar{\Delta} > 1$  and  $1 < c < \infty$ , as the iterations of iTRS are unsuccessful, if  $\Delta > \bar{\Delta}$ ,  $\Delta$  is reduced to

$$\Delta := c^{-1} \lfloor \min(\|p\|_\infty, \Delta) \rfloor; \quad (30)$$

otherwise, to take advantage of small steps and increase the accuracy of the model function, the new formula

$$\Delta = \Delta - 1 \quad (31)$$

is used; because  $\Delta$  is reduced faster by (30) than  $\Delta$  is reduced by (31), (30) may ignore some small values for  $\Delta$ . In both cases, the trust region subproblem is solved, and a new integer trial point is generated.

### 3.9 mInteger, the mixed-integer phase

This section discusses mInteger and its two main ingredients miLSS and two combination directions. The goal of mInteger is find a significant decrease in the inexact function value by performing miLSS along combination direction going to or moving along a valley.

After performing cMATRS and iMATRS, regardless of whether or not the function value is reduced, miLSS is performed along exactly one of the two new combination directions defined below, or possibly their opposite directions, in the space of all  $x$ , if possible; otherwise, exactly in one of the spaces of all  $x_K$  and  $x_I$  in this order. Our experiments have shown that searching in the space of all  $x$  after searching in the space of all  $x_K$  and in the space of all  $x_I$  improves the efficiency and robustness of our algorithm.

Both cMATRS and iMATRS may generate many small improved steps accumulated by going along a combination direction, starting at a point with a small inexact function value and reaching a point with even smaller inexact function values, confirming that the iterations of the algorithm can enter a valley and move down to make further progress on the inexact function value.

miLSS is performed along one of the following new combination directions:

- We first compute the difference

$$d = \begin{pmatrix} d_K \\ d_I \end{pmatrix} \neq 0$$

of the two best points found by cMATRS and iMATRS. In this case, at least one of cMATRS and iMATRS can update the best point.

- If  $\|d'\| = 0$ , it means that both cMATRS and iMATRS could not reduce the inexact function value. In this case, to compute our new direction, we save a finite number  $m$  of evaluated points as the columns of the matrix  $X$  and their inexact function values as the components of the vector  $F$  in ascending order, and update  $X$  and  $F$  when the new trial point and its inexact function value are evaluated. Then, we randomly select an evaluated point whose place is between  $m/2$  and  $m$ , remove this selected point and its inexact function value from  $X$  and  $F$ , and add the new evaluated point and its inexact function value to  $X$  and  $F$ , so that the ascending order of inexact function values at these points is preserved. The difference

$$d = \begin{pmatrix} d_K \\ d_I \end{pmatrix} = X_{:1} - X_{:m} \neq 0$$

of the best point saved in  $X$  and the worst point saved in  $X$  is computed.

As described in Subsections 3.2 and 3.3, we compute the initial step size and the largest allowed step size in both continues and integer cases by (6) and (9), respectively, and then we evaluate the new trial point

$$x^{\text{trial}} = \begin{pmatrix} x_K^{\text{best}} + \alpha' d_K \\ x_I^{\text{best}} + \alpha'' d_I \end{pmatrix},$$

where  $\alpha'$  and  $\alpha''$  are computed by (8) and (10), respectively.

## 4 Implementation

We mention some implementation details of MATRS:

- Large steps in the mutation and recombination phases are one of causes for line search failure. To avoid these, we replace in cMATRS and iMATRS the affine scaling matrix by an identity matrix if its infinity norm is greater than a positive tuning parameter  $m_{\text{max}}$ .

## 30 4. IMPLEMENTATION

• Since **iTRS** may not find a new integer feasible trial point, we need different sample points whenever the gradient of the trust region subproblem is approximated or the radius of the trust region is differently updated from the formulas (30) and (31), so that the approximate solution of the trust region subproblem becomes different and the chance of finding a new integer feasible point increases. Therefore, we first randomly select points from the list of stored evaluated points to estimate the gradient of the trust region subproblem. If this change does not help to generate a new integer feasible point, we randomly use one of the formulas

$$\Delta = \lfloor \Delta + \text{sign}(\text{rand} - 0.5) \text{randi}([1, \Delta_{\min}], 1) \rfloor$$

or

$$\Delta = \lfloor \Delta / \text{randi}([1, \Delta_{\min}], 1) \rfloor$$

at most **stuckmax** until  $\Delta \geq 1$ . Here **stuckmax**  $\geq 1$  is a tuning parameter,  $\Delta_{\min} > 1$  is an integer tuning parameter, and **rand** and **randi** are as in Section 2.

• In **iTRS**, if the approximate solution of the trust region subproblem is zero, it is replaced by

$$p = x_{\text{best}} - x_1, \quad p := \Delta \lfloor p / \|p\| \rfloor,$$

where  $x_1$  is the first sample point used for the approximation of the gradient of the trust region subproblem.

• Following [4], both **iMATRS** and **cMATRS** compute the following parameters that are used to compute the integer and real step sizes  $\sigma$  and the affine scaling matrix  $M$  in each iteration of **MATRS**:

$$\begin{aligned} w_i^0 &:= \ln\left(\mu + \frac{1}{2}\right) - \ln i, \quad w_i := \frac{w_i^0}{\sum_{j=1}^{\mu} w_j^0} \quad \text{for } i = 1, \dots, \mu, \\ \mu_w &:= \frac{1}{\sum_{j=1}^{\mu} w_j^2}, \quad c_\sigma := \min\left(1.999, \frac{\mu_w + 2}{n + \mu_w + 5}\right), \\ e_\sigma &:= \sqrt{n}\left(1 - 1/(4n) - 1/(21n^2)\right), \quad c_1 := 2/\left((n + 1.3)^2 + \mu_w\right), \\ c_\mu &:= \min\left\{1 - c_1, \frac{2(\mu_w - 2 + 1)/\mu_w}{(n + 2)^2 + \mu_w}\right\}, \\ d_\sigma &:= 1 + c_\sigma + 2 \max\left\{0, \sqrt{\frac{\mu_w - 1}{n + 1}} - 1\right\}. \end{aligned}$$

## 5 Numerical results

In this section, we compare our solver **MATRS** with the four mixed-integer solvers, **BFO** of PORCELLI & TOINT [29], **NOMAD** of ABRAMSON et al. [1], and **MISO** of MÜLLER [26], **DFLBOX** of LIUZZI et al. [22], and the integer solver **DFLINT** of LIUZZI et al. [23] on test problems from the **BARON** collection of SAHINIDIS [31] for the dimensions  $2 \leq n \leq 30$ .

### 5.1 Codes compared

The details of codes compared are as follows:

- **NOMAD** (version 3.9.1), obtained from

<https://www.gerad.ca/nomad>

is a Mesh Adaptive Direct Search algorithm (MADS) [1].

- **BFO**, obtained from

<https://github.com/m01marpor/BFO>

is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by PORCELLI & TOINT [29].

- **MISO** is a bound-constrained mixed-integer surrogate optimization solver of MÜLLER [26]. We selected **MISO-CPTV** and **MISO-CPTV-local** of **MISO** from [26, Table 1] and renamed them **MISO1** and **MISO2**, respectively.

- **DFLBOX**, obtained from

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

is a derivative-free line search solver for mixed-integer bound-constrained optimization by LIUZZI et al. [22].

- **DFLINT**, obtained from

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

is a derivative-free line search solver for integer bound-constrained optimization by LIUZZI et al. [23].

- **MATRS** is available at

<https://github.com/GS1400/MATRS>

## 32 5. NUMERICAL RESULTS

and chooses the following default values for its tuning parameters:

**iMATRS:**

$$\lambda = \max(6, |I|), \quad \mu = 3 + \lceil \log(|I|) \rceil, \quad \sigma_0 = 1, \quad \eta = 10^{-20}, \quad c = \nu = 2, \\ \bar{\Delta} = 3, \quad \mathbf{sc} = 5, \quad \sigma_{\max} = 100, \quad m_{\max} = 5, \quad \mathbf{stuckmax} = 10, \quad \Delta_{\min} = 10, \\ \Delta_{\max} = 30, \quad \kappa_{\max} = 30, \quad N = 10^4.$$

**cMATRS:**

$$\lambda = \max(6, |K|), \quad \mu = 3 + \lceil \log(|K|) \rceil, \quad \sigma_0 = 1, \quad \eta = 10^{-20}, \quad c = \nu = 2, \\ \sigma_{\min} = 10^{-10}, \quad \mathbf{sc} = 10, \quad \sigma_{\max} = 10^{10}, \quad \Delta_{\min} = 10^{-3}, \quad m_{\max} = 5, \\ \Delta_{\max} = 1, \quad \kappa_{\max} = 20, \quad N = 10^4.$$

All compared solvers were used with the default parameters, except for NOMAD that uses the following option set

```
opts = nomadset('max_eval',nfmax,'max_iterations',
                2*nfmax,'model_search','1').
```

and that DFLBOX uses `alfa_stop = -∞`.

Unfortunately, the source code of DFNDFL [12] is Python and we could not run on in Matlab.

## 5.2 Test problems

Following [18], to construct mixed-integer test problems, we use three collections of test problems, namely the collections `global` (216 problems), `bcp` (230 problems), and `prince` (571 problems) from the BARON collection of SAHINIDIS [31] for the dimensions  $2 \leq n \leq 30$ , available at

<https://www.minlp.com/nlp-and-minlp-test-problems>.

As in [24, (16)] was done for discrete bound-constrained optimization problems, we define

$$\underline{x}_i := x_i^0 - 10, \quad \bar{x}_i := x_i^0 + 10, \quad \text{for } i = 1, 2, \dots, n,$$

and generate the continuous bound-constrained optimization problem

$$\begin{aligned} \min \quad & \Phi(x) \\ \text{s.t.} \quad & \underline{x}_i \leq x_i \leq \bar{x}_i, \quad \text{for } i = 1, 2, \dots, n. \end{aligned}$$



Here we denote by  $x^0 \in \mathbb{R}^n$  the standard initial points of unconstrained test problems from all above collections and choose  $x \in \mathbb{R}^n$ . Then, we transform this problem into the bound-constrained mixed-integer optimization problem

$$\begin{aligned} \min f(x) &:= \Phi \left( \begin{pmatrix} \underline{x}_I + 0.01(\bar{x}_I - \underline{x}_I)x_I \\ x_K \end{pmatrix} \right) \\ \text{s.t. } 0 &\leq x_i \leq 100, \quad \text{for } i \in I, \\ x_i &\leq x_i \leq \bar{x}_i, \quad \text{for } i \in K. \end{aligned}$$

We refer to three resulting mixed-integer problem collections: **globalMint** (598 problems), **bcpMint** (458 problems), and **princeMint** (1361 problems). With dimensions  $2 \leq n \leq 30$  and the three noise levels  $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ , this gives a total of  $3 \times 3180 = 9540$  mixed-integer test problems. We also refer to three resulting integer problem collections: **globalInt** (216 problems), **bcpInt** (230 problems), and **princeInt** (571 problems). With dimensions  $2 \leq n \leq 30$  and the three noise levels  $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ , this gives a total of  $3 \times 3180 = 1017$  integer test problems.

The type of noise is absolute uniform noise, i.e.,  $\tilde{f} = f + (2 * \text{rand} - 1)\omega$  with  $\text{rand} \sim \mathcal{N}(0, 1)$ . For all test problems, the initial points are chosen as

- $x_i^0 := 50$  for  $i \in I$  as in [23, Section 4];
- $x_i^0$  for  $i \in K$  as given in [31].

### 5.3 Tools for efficiency and robustness

We denote by **nfmax** the maximum number **nf** of function evaluations and by **secmax** the maximum time in seconds (**sec**). The budget available for each solver is limited by allowing at most **secmax** := 360 seconds of run time and at most **nfmax** :=  $1200n$  function evaluations for a problem with  $n$  variables. We chosen **secmax** and **nfmax** so that the best solver can solve at least 60% of the selected problems. As an example, in Table 1, for  $\omega = 0.001$  all solvers solve 98% problems and the first more robust solver **MATRS** solves 91% problems. In this result, **MATRS** terminates in 0.08% of problems because **nfmax** is reached, while it never terminates because **secmax** is reached. However, for integer problems, since it is difficult to find new integer feasible points, all solvers terminate due to reaching **secmax** at least once and increasing **secmax** does not change efficiency and robustness.

Let  $f_{\text{init}}$  denote the function value of the starting point (common to all solvers),  $f_{\text{opt}}$  denote the best point known to us, and  $f_s$  denote the best point found by the solver  $s$ . We say that the solver  $s$  solves a problem with dimension

## 34 5. NUMERICAL RESULTS

$n$  if the target accuracy

$$q_s := (f_s - f_{\text{opt}})/(f_{\text{init}} - f_{\text{opt}}) \leq \epsilon = 10^{-4}$$

is satisfied. Otherwise, it cannot solve such a problem since either `nfmax` or `secmax` were reached.  $q_s$  identifies the convergence speed of the solver  $s$  to reach a minimum of the smooth true function  $f$ .

Denote by  $\mathcal{S}$  the list of compared solvers and by  $\mathcal{P}$  the list of problems. We say that the solver  $s$  is most *efficient* on a collection if it has the lowest relative cost of function evaluations. A good tool to evaluate the efficiency of the compared solvers is the performance profile of DOLAN & MORÉ [9]. The performance profile of the solver  $s$

$$\rho_s(\tau) := \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} \mid pr_{p,s} \leq \tau \right\} \right| \quad (32)$$

counts the fraction of problems solved by the solver  $s$  such that the upper bound of the *performance ratio*  $pr_{p,s} := \frac{c_{p,s}}{\min(c_{p,\bar{s}} \mid \bar{s} \in S)}$  is  $\tau$ .

We say that the solver  $s$  is most *robust* on a collection if it has the highest number of solved problems. A good tool to evaluate the robustness of the compared solvers is the data profiles of MORÉ & WILD [25]. The data profile of the solver  $s$

$$\delta_s(\kappa) := \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} \mid cr_{p,s} \leq \kappa \right\} \right| \quad (33)$$

is the fraction of problems solved by the solver  $s$  with  $\kappa$  groups of  $n_p + 1$  function evaluations such that  $\kappa$  is the upper bound of the *cost ratio*  $cr_{p,s} := \frac{c_{p,s}}{n_p + 1}$ . Here  $n_p$  is the dimension of the problem  $p \in \mathcal{P}$  and  $c_{p,s}$  is the *cost measure* of the solver  $s$  to solve the problem  $p$ .

For a given collection  $S$  of solvers, the strength of a solver  $s \in S$  – relative to an ideal solver corresponding to the best solver for each problem – is measured for each given cost measure  $c_s$  by the number  $e_s$  given by

$$e_s := \begin{cases} \left( \frac{\min_{\bar{s} \in S} c_{\bar{s}}}{c_s} \right), & \text{if the solver } s \text{ solves the problem,} \\ 0, & \text{otherwise,} \end{cases}$$

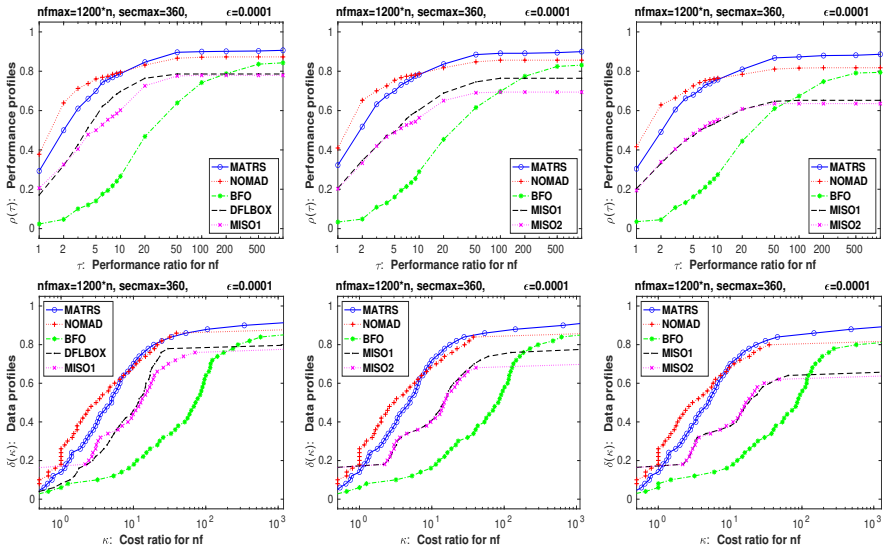
called the *efficiency* of the solver  $s$  with respect to this cost measure, which is the inverse of the performance ratio of the solver  $s$ . In all tables, efficiencies are given in percent. Larger efficiencies in this table imply a better average behavior, while a zero efficiency indicates failure. All values are rounded (against zero) to integers. In the table not recording efficiencies, a sign

- $n$  indicates that  $\mathbf{nf} \geq \mathbf{nfmax} = 1200n$  was reached.
- $t$  indicates that  $\mathbf{sec} \geq \mathbf{secmax} = 360$  seconds was reached.
- $f$  indicates that the solver  $s$  failed for other reasons, such as bugs or algorithmic terminations. In particular, MIS02 terminated in some cases with the message ‘index in position 2 exceeds array bounds’ and MIS01 terminated because  $\mathbf{secmax}$  was reached much more often than for the other solvers, even for problems with dimension 2. So changing  $\mathbf{secmax}$  does not help MIS01.

$\mathbf{nf}$  is used as the cost measure for the data and performance profiles, and both  $\mathbf{nf}$  and  $\mathbf{sec}$  are used as two cost measures for all tables.

## 5.4 A comparison of mixed-integer bound-constrained DFO solvers

## 5.4.1 Results for globalMint



**Fig. 4:** Plots for `globalMint` for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Performance profiles  $\rho(\tau)$  (first row) are in dependence of a bound  $\tau$  on the performance ratio (see (32)), while data profiles  $\delta(\kappa)$  (second row) are in dependence of a bound  $\kappa$  on the cost ratio (see (33)). Problems solved by no solver are ignored.

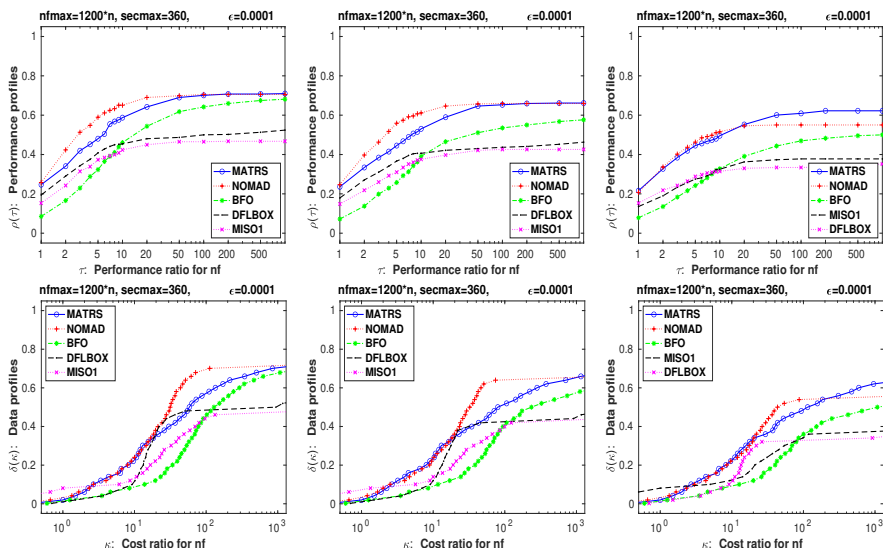
stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
591 of 598 problems solved						583 of 598 problems solved						574 of 598 problems solved								
$\omega = 0.001$						$\omega = 0.01$						$\omega = 0.1$								
dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %		dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %		dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %				
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
MATRS	549	49	0	0	51	50	MATRS	543	55	0	0	52	49	MATRS	535	63	0	0	50	57
NOMAD	522	0	0	76	58	41	NOMAD	512	0	0	86	57	48	NOMAD	489	0	0	109	57	43
BFO	513	60	0	25	10	17	BFO	504	65	0	29	10	15	BFO	488	71	0	39	10	16
DFLBOX	470	128	0	0	37	51	MISO1	457	0	141	0	36	22	MISO1	390	0	208	0	33	20
MISO1	466	0	132	0	35	22	MISO2	415	0	44	139	34	22	MISO2	380	0	47	171	34	22
MISO2	448	0	55	95	35	25	DFLBOX	332	266	0	0	27	37	DFLBOX	252	346	0	0	21	29

**Table 1:** Tabulated results for `globalMint` for dimensions  $2 \leq n \leq 30$ .

From Figure 4 and Table 1, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on `globalMint`:

- MATRS and NOMAD are the two best solvers.
- The most robust solver MATRS solves slightly more problems than the other solvers.
- The `nf` efficiency of NOMAD is almost 7% higher than that of the other solvers.

## 5.5 Results for bcpMint



**Fig. 5:** Plots for bcpMint for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

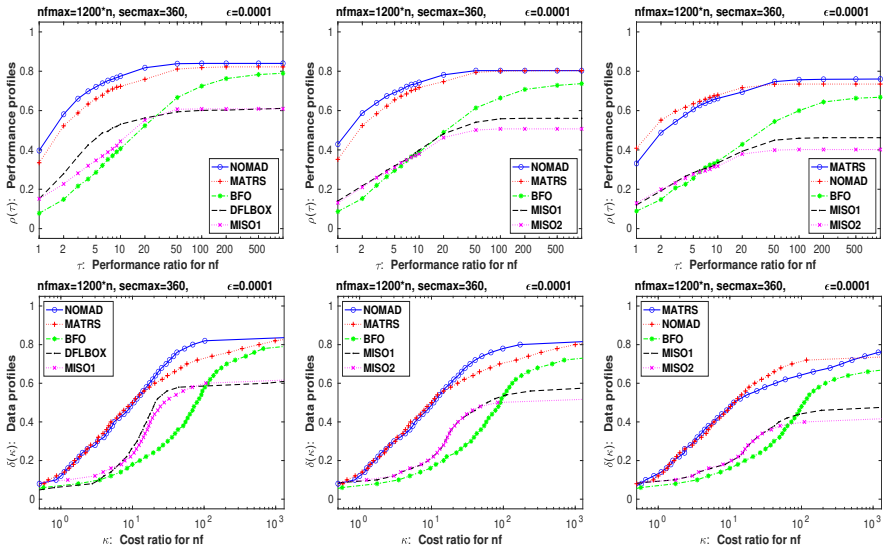
stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
413 of 458 problems solved						386 of 458 problems solved						352 of 458 problems solved								
$\omega = 0.001$						$\omega = 0.01$						$\omega = 0.1$								
dim ∈ [2,30]	# of anomalies					$e_a$ in %	dim ∈ [2,30]	# of anomalies					$e_a$ in %	dim ∈ [2,30]	# of anomalies					$e_a$ in %
solver	solved	#n	#l	#f	nf	sec	solver	solved	#n	#l	#f	nf	sec	solver	solved	#n	#l	#f	nf	sec
MATRS	325	133	0	0	37	40	MATRS	303	155	0	0	34	38	MATRS	285	173	0	0	33	35
NOMAD	323	0	4	131	42	29	NOMAD	302	0	3	153	39	24	NOMAD	252	0	1	205	33	23
BFO	314	72	0	72	21	33	BFO	267	67	0	124	17	27	BFO	231	70	0	157	16	23
DFLBOX	240	218	0	0	30	40	DFLBOX	212	246	0	0	27	36	DFLBOX	173	0	285	0	20	10
MIS01	214	0	244	0	24	14	MIS01	195	0	263	0	22	14	MIS01	162	296	0	0	22	27
MIS02	191	0	57	210	22	13	MIS02	169	0	63	226	20	13	MIS02	159	0	52	247	19	12

**Table 2:** Tabulated results for bcpMint for dimensions  $2 \leq n \leq 30$ .

From Figure 5 and Table 2, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on bcpMint:

- MATRS and NOMAD are the two best solvers.
- The most robust solver MATRS solves slightly more problems than the other solvers.
- The nf efficiency of NOMAD is almost 5% higher than that of the other solvers.

## 5.5.1 Results on princeMint



**Fig. 6:** Plots for princeMint for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
1271 of 1361 problems solved						1252 of 1361 problems solved						1180 of 1361 problems solved								
$\omega = 0.001$						$\omega = 0.01$						$\omega = 0.1$								
dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %		dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %		dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %				
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
NOMAD	1143	0	8	210	57	43	NOMAD	1093	0	3	265	55	35	NOMAD	1035	326	0	0	47	53
MATRS	1119	241	0	1	51	58	MATRS	1093	267	0	1	51	60	NOMAD	1000	0	0	361	52	46
BFO	1079	167	0	115	19	20	BFO	1007	160	0	194	18	23	BFO	912	173	0	276	16	17
DFLBOX	832	529	0	0	30	42	MIS01	763	0	598	0	24	11	MIS01	629	0	732	0	20	9
MIS01	829	0	532	0	25	12	MIS02	690	0	158	513	22	11	MIS02	547	0	139	675	20	10
MIS02	813	0	157	391	26	12	DFLBOX	606	755	0	0	24	33	DFLBOX	474	887	0	0	20	22

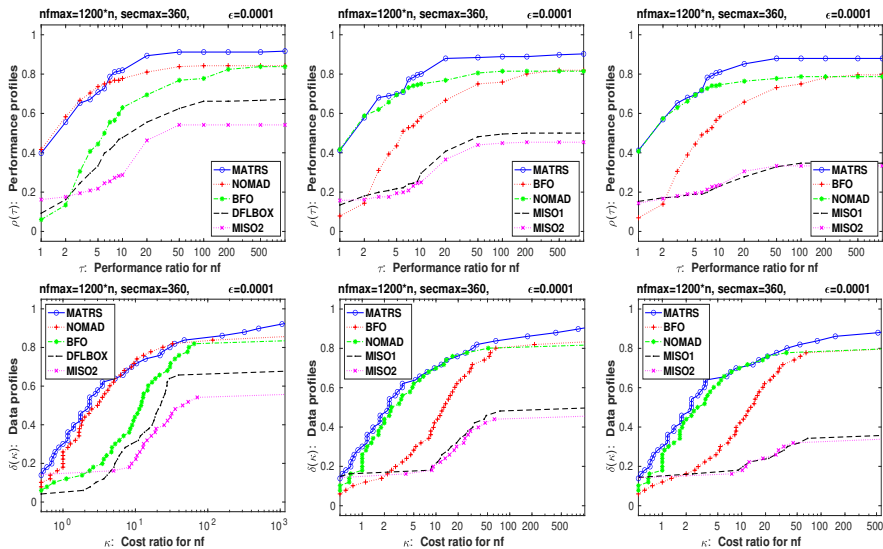
**Table 3:** Tabulated results for princeMint for dimensions  $2 \leq n \leq 30$ .

From Figure 6 and Table 3, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on princeMint:

- MATRS and NOMAD are the two best solvers.
- The most robust solver MATRS solves slightly more problems than the other solvers.
- The nf efficiency of NOMAD is at most 6% higher than that of the other solvers.

## 5.6 A comparison of continuous bound-constrained DFO solvers

### 5.6.1 Results for global



**Fig. 7:** Plots for `global` for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

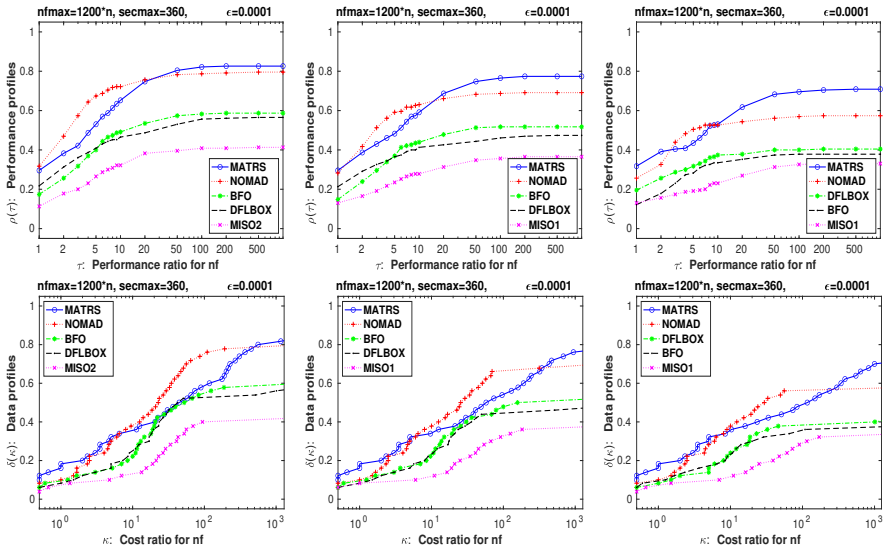
stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
213 of 216 problems solved						210 of 216 problems solved						206 of 216 problems solved								
$\omega = 0.001$						$\omega = 0.01$						$\omega = 0.1$								
dim	# of anomalies			$e_s$ in %		dim	# of anomalies			$e_s$ in %		dim	# of anomalies			$e_s$ in %				
dim	solved	#n	#t	#f	nf	sec	dim	solved	#n	#t	#f	nf	sec	dim	solved	#n	#t	#f	nf	sec
MATRS	199	17	0	0	56	36	MATRS	196	20	0	0	57	38	MATRS	190	26	0	0	57	38
NOMAD	182	0	0	34	57	51	BFO	178	0	0	38	23	54	BFO	172	0	0	44	24	54
BFO	181	0	0	35	23	55	NOMAD	176	0	0	40	54	46	NOMAD	170	0	0	46	54	50
DFLBOX	145	71	0	0	22	53	MISO1	108	0	108	0	19	11	MISO1	75	0	141	0	17	9
MISO2	117	0	34	65	19	12	MISO2	98	0	27	91	18	11	MISO2	72	0	26	118	17	12
MISO1	115	0	101	0	19	11	DFLBOX	73	143	0	0	14	27	DFLBOX	57	159	0	0	11	20

**Table 4:** Tabulated results for `global` for dimensions  $2 \leq n \leq 30$ .

From Figure 7 and Table 4, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on `global`:

- The most robust solver MATRS solves slightly more problems than the other solvers.
- The `nf` efficiency of MATRS is slightly higher than that of the other solvers.

## 5.6.2 Results for bcp



**Fig. 8:** Plots for bcp for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$							stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$							stopping test: $q_s \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						
205 of 230 problems solved							194 of 230 problems solved							180 of 230 problems solved						
$\omega = 0.001$							$\omega = 0.01$							$\omega = 0.1$						
dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %			dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %			dim ∈ [2,30]	# of anomalies			$\epsilon_s$ in %		
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
MATRS	190	40	0	0	41	31	MATRS	178	52	0	0	40	31	MATRS	163	67	0	0	39	32
NOMAD	183	0	1	46	48	35	NOMAD	159	0	2	69	43	35	NOMAD	132	0	0	98	35	28
BFO	136	5	0	89	29	44	BFO	119	0	0	111	25	39	DFLBOX	93	137	0	0	25	34
DFLBOX	130	100	0	0	32	44	DFLBOX	109	121	0	0	29	40	BFO	87	0	0	143	19	25
MISO2	95	0	20	115	18	9	MISO1	84	0	146	0	17	9	MISO1	76	0	154	0	15	9
MISO1	91	0	139	0	18	9	MISO2	83	0	16	131	16	9	MISO2	63	0	13	154	14	8

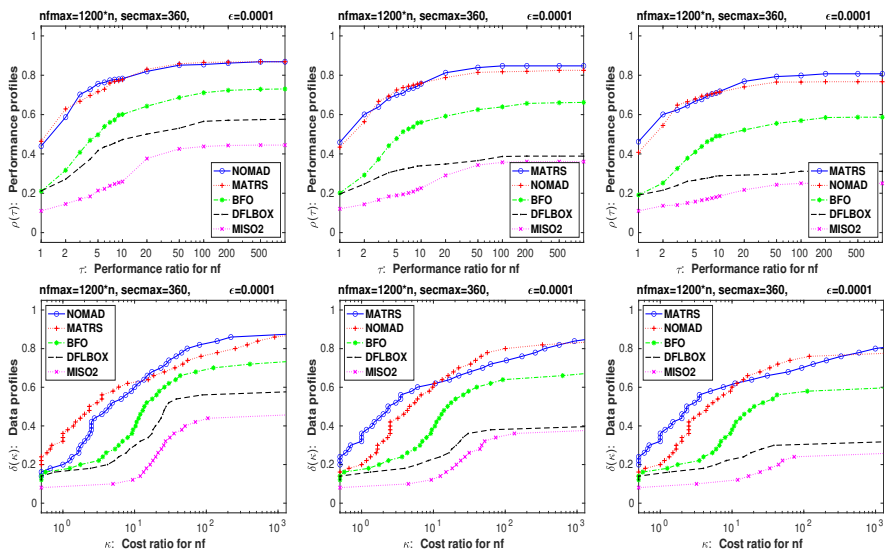
**Table 5:** Tabulated results for bcp for dimensions  $2 \leq n \leq 30$ .

From Figure 8 and Table 5, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on bcp:

- MATRS and NOMAD are the two best solvers.
- The most robust solver MATRS solves slightly more problems than the other solvers.
- The nf efficiency of NOMAD is almost 7% higher than that of the other solvers.



## 5.6.3 Results for prince



**Fig. 9:** Plots for `prince` for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

stopping test: $q_a \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_a \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_a \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
536 of 571 problems solved						519 of 571 problems solved						490 of 571 problems solved								
$\omega = 0.001$						$\omega = 0.01$						$\omega = 0.1$								
dim ∈ [2,30]	# of anomalies			$e_a$ in %		dim ∈ [2,30]	# of anomalies			$e_a$ in %		dim ∈ [2,30]	# of anomalies			$e_a$ in %				
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
NOMAD	496	0	3	72	60	51	MATRS	484	85	0	2	59	41	MATRS	461	110	0	0	58	41
MATRS	496	75	0	0	61	39	NOMAD	471	0	2	98	58	50	NOMAD	438	0	0	133	55	49
BFO	417	4	0	150	36	51	BFO	379	2	0	190	33	46	BFO	335	0	0	236	29	38
DFLBOX	329	242	0	0	30	44	DFLBOX	222	349	0	0	25	30	DFLBOX	178	393	0	0	22	24
MISO2	254	0	54	263	16	6	MISO2	206	0	46	319	15	5	MISO2	143	0	53	375	13	4
MISO1	231	0	340	0	16	7	MISO1	188	0	383	0	14	8	MISO1	135	0	436	0	12	8

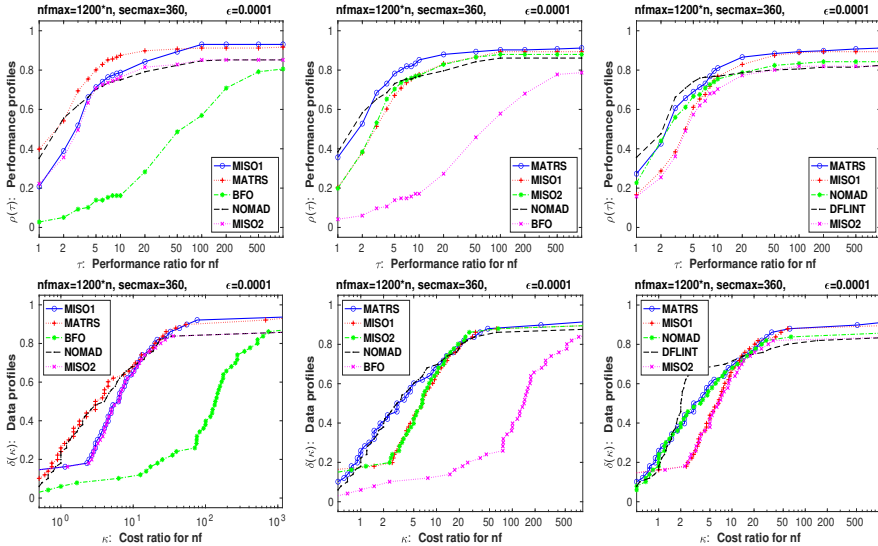
**Table 6:** Tabulated results for `prince` for dimensions  $2 \leq n \leq 30$ .

From Figure 9 and Table 6, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on `prince`:

- MATRS and NOMAD are the two best solvers.
- The most robust solver MATRS solves slightly more problems than the other solvers.
- The `nf` efficiency of MATRS is slightly higher than that of the other solvers.

## 5.7 A comparison of integer bound-constrained DFO solvers

### 5.7.1 Results for globalInt



**Fig. 10:** Plots for `globalInt` for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

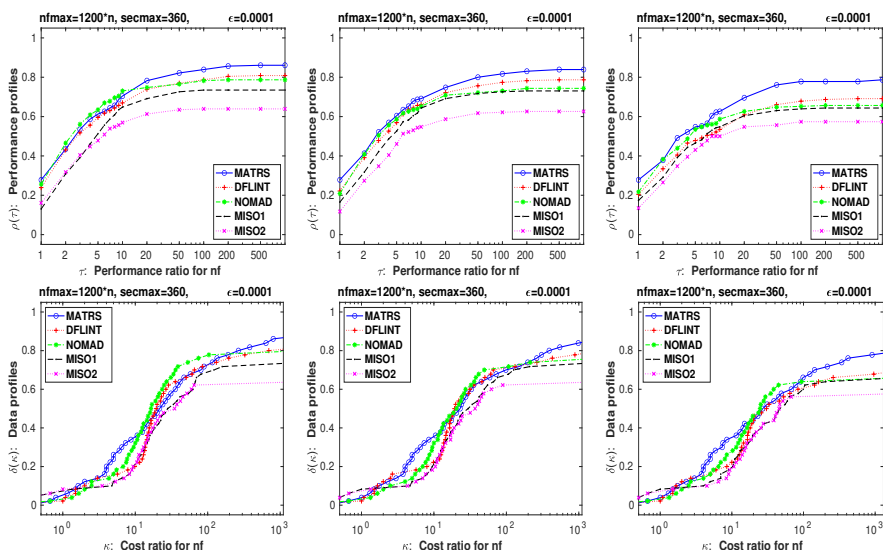
stopping test: $q_e \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_e \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_f \leq 0.0001$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
216 of 216 problems solved			$\omega = 10^{-3}$			215 of 216 problems solved			$\omega = 10^{-2}$			214 of 216 problems solved			$\omega = 10^{-1}$					
dim ∈ [2,30]	# of anomalies			$e_s$ in %		dim ∈ [2,30]	# of anomalies			$e_s$ in %		dim ∈ [2,30]	# of anomalies			$e_s$ in %				
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
MIS01	201	0	15	0	36	24	MATRS	198	15	3	0	48	36	MATRS	198	15	3	0	48	41
MATRS	200	13	3	0	50	37	MIS01	193	0	23	0	36	22	MIS01	193	0	23	0	36	29
BFO	188	0	0	28	8	24	MIS02	190	0	8	18	36	27	NOMAD	182	0	7	27	44	43
NOMAD	184	0	4	28	42	45	NOMAD	186	0	4	26	45	44	DFLINT	178	2	21	15	53	55
MIS02	184	0	7	25	35	27	BFO	183	0	0	33	8	21	MIS02	177	0	8	31	33	30
DFLINT	184	0	21	11	54	61	DFLINT	181	1	22	12	53	58	BFO	167	0	0	49	6	23

**Table 7:** Tabulated results for `globalInt` for dimensions  $2 \leq n \leq 30$ .

From Figure 10 and Table 7, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on `globalInt`:

- The most robust solver MATRS solves slightly more problems than the other solvers.
- The `nf` efficiency of DFLINT is almost 5% higher than that of the other solvers.

## 5.7.2 Results for bcpInt



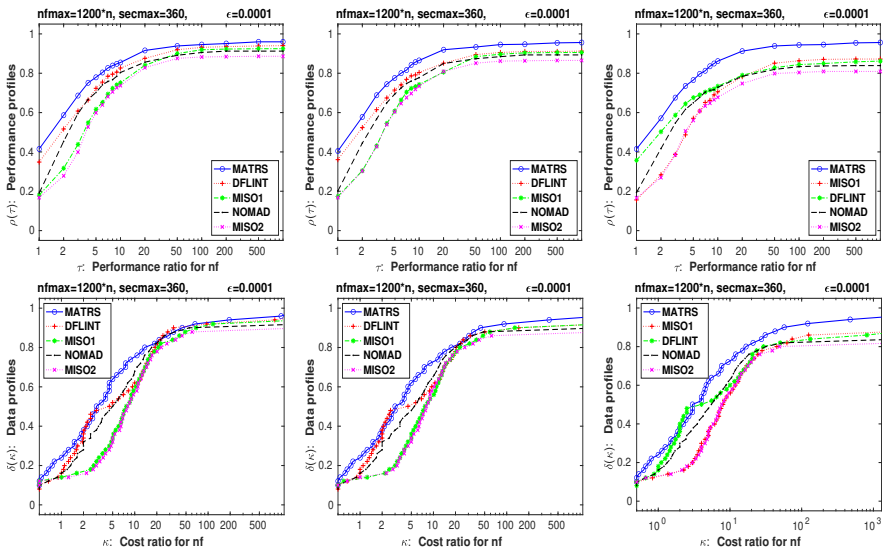
**Fig. 11:** Plots for bcpInt for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

stopping test:							stopping test:							stopping test:						
$q_s \leq 10^{-4}, \text{sec} \leq 360, \text{nf} \leq 1200 * n$							$q_s \leq 10^{-4}, \text{sec} \leq 360, \text{nf} \leq 1200 * n$							$q_s \leq 10^{-4}, \text{sec} \leq 360, \text{nf} \leq 1200 * n$						
226 of 230 problems solved							218 of 230 problems solved							210 of 230 problems solved						
$\omega = 10^{-3}$							$\omega = 10^{-2}$							$\omega = 10^{-1}$						
dim ∈ [2,30]	# of anomalies			$e_s$ in %			dim ∈ [2,30]	# of anomalies			$e_s$ in %			dim ∈ [2,30]	# of anomalies			$e_s$ in %		
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
MATRS	198	30	2	0	42	34	MATRS	194	35	1	0	42	35	MATRS	182	47	1	0	39	34
DFLINT	186	0	1	48	43	39	DFLINT	181	25	4	20	39	54	DFLINT	159	39	3	29	33	46
NOMAD	181	0	1	48	43	39	NOMAD	171	0	1	58	38	35	NOMAD	151	0	5	74	35	33
MIS01	169	0	61	0	32	17	MIS01	168	0	62	0	33	19	MIS01	148	0	82	0	31	19
MIS02	147	0	13	70	31	17	MIS02	144	0	7	79	28	16	MIS02	132	0	10	88	27	17
BFO	105	1	0	124	18	29	BFO	103	1	0	126	18	29	BFO	75	1	0	154	13	20

**Table 8:** Tabulated results for bcpInt for dimensions  $2 \leq n \leq 30$ .

From Figure 11 and Table 8, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on bcpInt:

- The most robust solver MATRS solves slightly more problems than the other solvers.
- The nf efficiency of MATRS is almost 5% higher than that of the other solvers.

5.7.3 Results for `princeInt`

**Fig. 12:** Plots for `princeInt` for dimensions  $2 \leq n \leq 30$  and noise levels  $\omega = 0.001$  (left),  $0.01$  (middle),  $0.1$  (right). Other details as in Figure 4.

stopping test: $q_e \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_e \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$						stopping test: $q_e \leq 10^{-4}$ , $\text{sec} \leq 360$ , $\text{nf} \leq 1200 * n$								
559 of 571 problems solved						558 of 571 problems solved						554 of 571 problems solved								
$\omega = 10^{-3}$						$\omega = 10^{-2}$						$\omega = 10^{-1}$								
dim $\in$ [2,30]	# of anomalies					$e_s$ in %	dim $\in$ [2,30]	# of anomalies					$e_s$ in %	dim $\in$ [2,30]	# of anomalies					$e_s$ in %
solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec	solver	solved	#n	#t	#f	nf	sec
MATRS	548	22	0	1	59	42	MATRS	546	24	0	1	58	45	MATRS	546	24	0	1	59	52
DFLINT	537	11	6	17	53	63	DFLINT	522	20	6	23	54	57	MISO1	498	0	72	1	35	29
MISO1	528	0	42	1	38	27	MISO1	517	0	53	1	37	28	DFLINT	492	36	5	38	52	52
NOMAD	521	0	1	49	46	42	NOMAD	510	0	3	58	45	38	NOMAD	479	0	2	90	43	35
MISO2	506	0	11	54	36	27	MISO2	494	0	11	66	37	30	MISO2	462	0	10	99	34	31
BFO	444	0	0	127	15	33	BFO	428	0	0	143	14	33	BFO	351	0	0	220	12	27

**Table 9:** Tabulated results for `princeInt` for dimensions  $2 \leq n \leq 30$ .

From Figure 12 and Table 9, we conclude that for all noise levels ( $\omega = 10^{-3}, 10^{-2}, 10^{-1}$ ) on `princeInt`:

- The most robust solver MATRS solves slightly more problems than the other solvers.
- The `nf` efficiency of MATRS is almost 7% higher than that of the other solvers.

## 6 Conclusion

This paper describes a new matrix adaptation trust region strategy for bound-constrained DFO problems with mixed-integer variables. This strategy finds the best points by integer and continuous line search methods in the mutation and recombination phases, by integer and continuous trust region methods in the trust region phase, and by mixed-integer line search method in the mixed-integer phase. A new randomized space-filling method was proposed as a good and cheap replacement for the Halton sequences to generate well-distributed mutation points in the integer and continuous mutation phases.

Numerical results show

- on the three mixed-integer collections and the three continuous collections that the two best solvers MATRS and NOMAD are comparable in terms of robustness and efficiency;
- on the three integer collections that the two best solvers MATRS and DFLINT are comparable in terms of robustness and efficiency;
- the increase in noise slightly reduces the efficiency and robustness of MATRS compared to the other solvers.

**Acknowledgment** The first author acknowledges financial support of the Austrian Science Foundation under Project No. P 34317.

## References

- [1] M. A. Abramson, C. Audet, G. Couture, J. E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at <https://www.grad.ca/nomad/>.
- [2] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing (2017).
- [3] C. Audet, S. Le Digabel, and C. Tribes. The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. *SIAM J. Optim.* **29** (2019), 1164–1189.
- [4] H. G. Beyer. Design principles for matrix adaptation evolution strategies (2020).
- [5] J. Blank, K. Deb, Y. Dhebar, S. Bandaru, and H. Seada. Generating well-spaced points on a unit simplex for evolutionary many-objective optimization. *IEEE Trans. Evol.* **25** (2021), 48–60.
- [6] X. W. Chang, X. Yang, and T. Zhou. MLAMBDA: A modified LAMBDA method for integer least-squares estimation. *J. Geod.* **79** (2005), 552–565.

- [7] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics (2009).
- [8] Josef Dick and Friedrich Pillichshammer. *Digital nets and sequences: discrepancy theory and quasi-Monte Carlo integration*. Cambridge University Press (2010).
- [9] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002), 201–213.
- [10] G. Fasano, G. Liuzzi, S. Lucidi, and F. Rinaldi. A linesearch-based derivative-free approach for nonsmooth constrained optimization. *SIAM J. Optim* **24** (2014), 959–992.
- [11] A. Ghasemmehdi and E. Agrell. Faster recursions in sphere decoding. *IEEE Trans. Inf. Theory* **57** (2011), 3530–3536.
- [12] T. Giovannelli, G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for mixed-integer nonsmooth constrained optimization. *Comput. Optim. Appl.* **82** (2022), 293–327.
- [13] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).
- [14] W. Huyer and A. Neumaier. *J. Glob. Optim* **14** (1999), 331–355.
- [15] W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.* **35** (2008), 1–25.
- [16] M. Kimiaei. An active set trust-region method for bound-constrained optimization. *Bull. Iran. Math. Soc.* (2021).
- [17] M. Kimiaei. A developed randomized algorithm with noise level tuning for large-scale noisy unconstrained DFO problems and its real-life applications, Manuscript (2023). Available at <https://optimization-online.org/?p=16687>.
- [18] M. Kimiaei and A. Neumaier. Efficient composite heuristics for integer bound constrained noisy optimization, Manuscript (2022). Available at <https://optimization-online.org/2022/07/8998/>.
- [19] M. Kimiaei and A. Neumaier. Effective matrix adaptation strategy for noisy derivative-free optimization, Manuscript (2023). Available at <https://optimization-online.org/?p=22367>.
- [20] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.* (2022), 365–414.

- [21] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.* **28** (2019), 287–404.
- [22] G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for bound constrained mixed-integer optimization. *Comput. Optim. Appl.* **53** (2011), 505–526.
- [23] G. Liuzzi, S. Lucidi, and F. Rinaldi. An algorithmic framework based on primitive directions and nonmonotone line searches for black-box optimization problems with integer variables. *Math. Program. Comput.* **12** (2020), 673–702.
- [24] G. Liuzzi, S. Lucidi, and F. Rinaldi. TESTINT - a collection of 240 inequality constrained plus 61 bound constrained test problems for black-box integer programming. DFL – derivative-free library, <http://www.iasi.cnr.it/liuzzi/df/> (2022).
- [25] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* **20** (2009), 172–191.
- [26] J. Müller. MISO: mixed-integer surrogate optimization framework. *Optim. Eng.* **17** (2015), 177–203.
- [27] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM (1992).
- [28] N. Ploskas and N. V. Sahinidis. Review and comparison of algorithms and software for mixed-integer derivative-free optimization. *J. Glob. Optim.* **82** (2021), 433–462.
- [29] M. Porcelli and Ph. L. Toint. Exploiting problem structure in derivative free optimization. *ACM. Trans. Math. Softw.* **48** (2022), 1–25.
- [30] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.* **56** (2012), 1247–1293.
- [31] N. V. Sahinidis. *BARON 21.1.13: Global Optimization of Mixed-Integer Nonlinear Programs*, User’s Manual (2017).