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

Morteza Kimiaei[1*] and Arnold Neumaier[1]

[1,2]Fakultät für Mathematik, Universität Wien,
Oskar-Morgenstern-Platz 1, A-1090, Wien, Austria.

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

Contributing authors: 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of MATRS is a development of the bound-constrained derivative-free integer optimization solver IMATRS discussed in the unpublished manuscript by Kimiaei & Neumaier [18].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

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

Here

$$\mathbf{x} := \{x \in \mathbb{R}^n \mid \underline{x} \le x \le \overline{x}\} \ \text{ with } \underline{x}, \overline{x} \in \mathbb{R}^n \ (\underline{x} < \overline{x}) \tag{2}$$

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

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

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 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 covarince 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.

• 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 $x_1, x_2, x_3, \ldots$ 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.

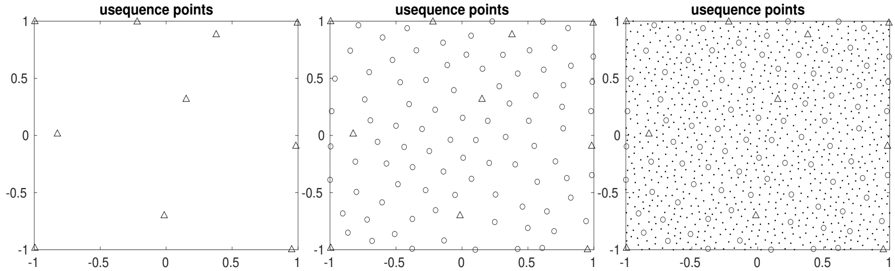

Fig. 1: Plots of 10, 100, and 1000 real points belonging $[-1,1]^2$ generated by `usequence`.

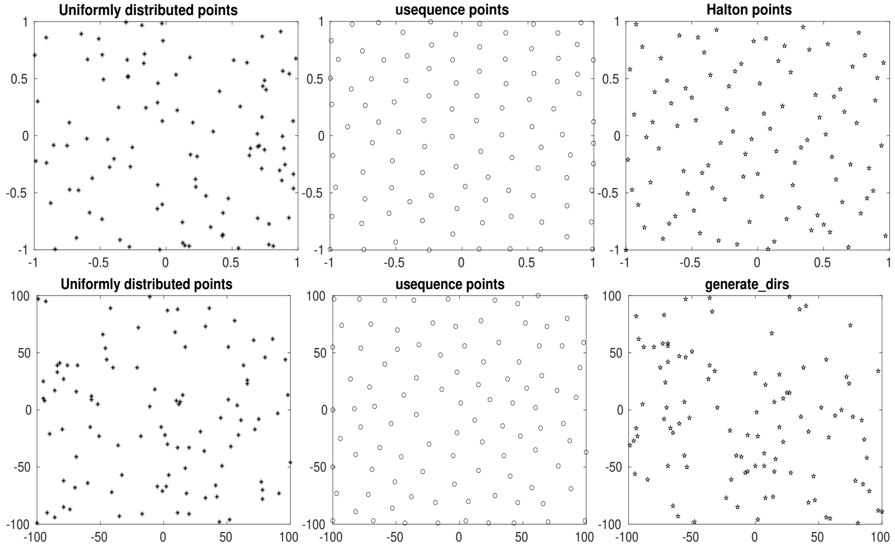

Fig. 2: Plots of points generated by random generators, by `usequence`, and the Halton sequences. First row: The first 100 real points in $[-1,1]^2$. Second row: The first 100 integer points in $[-100:100]^2$.

Algorithm 1 usequence

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

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

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

```
for $i = 1, \dots, n$ do                                  ▷ generate random reservoir matrix
    if $z_i > 0$, $R_{i:} = \text{randi}([-z_i, z_i], 1, m) \in \mathbb{Z}^{n\times m}$.
    else, $R_{i:} = 2\,\text{rand}(1, m) - 1 \in \mathbb{R}^{n\times m}$.
    end if
end for
$R_{:1} = p$ and $j = 1$.
for $k = 1, \dots, m$ do
    $D_{:k} = R_{:j}$.                                     ▷ next point
    for $i = 1, \dots, n$ do                              ▷ generate a new random vector
        if $z_i > 0$, $r_i = \text{randi}([-z_i, z_i], 1) \in \mathbb{Z}$.
        else, $r_i = 2\,\text{rand} - 1 \in \mathbb{R}$.
        end if
    end for
    $R_{:j} = r$.                                          ▷ reservoir update
    $\texttt{on} = \text{ones}(1, k)$; $R' = r_{:,\texttt{on}}$; $D' = D_{:,1:k}$;   ▷ forming matrices $R'$ and $D'$
    $u = \sum_{\ell=1:k} (R'_{:\ell} - D'_{:\ell})^2$; $u_j = \min_{\ell=1:n} u$   ▷ minimum squared distance value
    $\texttt{on} = k + \text{zeros}(1, n)$; $D'' = D_{:,\texttt{on}}$;   ▷ forming the matrix $D''$
    $s = \sum_{\ell=1:m} (R_{:\ell} - D''_{:\ell})^2$;     ▷ distance between $R$ and $D''$
    if $k = 1$, $u = s$;
    else, $u = \min(u, s)$;                               ▷ minimum squared distance vector $u$
    end if
    Find $j = \max_{t=1:n} u_t$ and choose $R_{:j} \in \{R_{:l}\}_{l=1}^{k}$.
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, $\mathrm{rand}(n, 1)$ is a real random vector whose n components are independent and uniformly distributed in $[0, 1]$, $\mathrm{randi}([a_i, b_i], n, 1)$ 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 $\mathrm{ones}(1, n)$ (line 15) is a $1 \times n$ vector whose components are one and $\mathrm{zeros}(1, n)$ (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 λ of integer mutation points by performing iLSS, an integer line search strategy, along λ integer mutation directions. If none of λ integer mutation points can be a new best point, iMATRS performs the selection phase to sort inexact function values at λ 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 λ continuous mutation points by performing cLSS, a continuous line search strategy, along λ continuous mutation directions. If none of λ continuous mutation points can be a new best point, cMATRS performs the selection phase to sort inexact function values at λ continuous mutation points in ascending order and select μ 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.

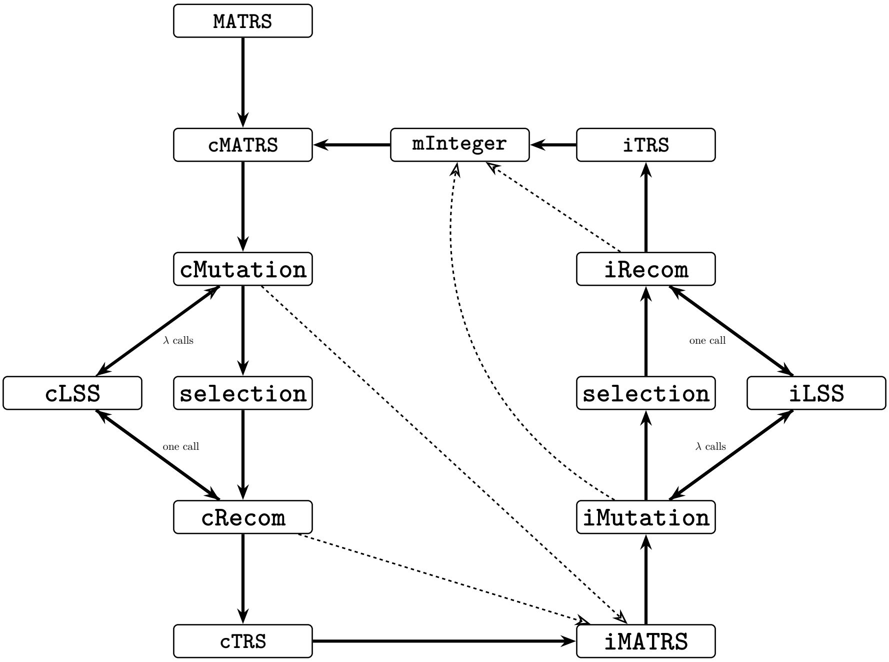

Fig. 3: Flowchart for `MATRS`. Dash arrows indicate conditional jumps described in the main text.

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

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

while nf < nfmax do    ▷ nf denotes the number of function evaluations
    • perform cMATRS with cMutation, selection, cRecom, and cTRS
    on the space of all $x_K$ to find a new best point.
    • perform iMATRS with iMutation, selection, iRecom, and iTRS
    on the space of all $x_I$ to find a new best point.
    • perform mInteger on exactly one of the spaces of all $x$, $x_K$, and $x_I$
    in this order to find a new best point.
end while

cMutation and iMutation
Compute a finite number $\lambda$ of the distribution and mutation directions.
Generate $\lambda$ mutation points and their inexact function values by line
searches (cLSS and iLSS).
selection
Sort directions, points, and their inexact function values obtained from
the mutation phase.
Select some of them with respect to the ascending order of the corresponding
function values for the recombination phase.
cRecom and iRecom
Compute a new recombination mutation direction.
Find a new recombination point and its inexact function value by line
searches (cLSS and iLSS).
Compute a new recombination step size and a new affine scaling matrix
using the selected information from the selection phase.
cTRS and iTRS
If the recombination point cannot be a new best point, perform trust
region strategies (cTRS and iTRS) to find such a best point.
mInteger
Compute a new combination direction with the goal of going into a valley.
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 $x_{\rm best}$ and its inexact function value $\tilde{f}_{\rm best}$.

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 λ 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 λ calls to `cLSS`. Otherwise, if none of the λ 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 $\alpha^{i\ell}_{\rm init}$ $(i = 1, \ldots, \lambda)$ for `cLSS` based on the largest allowed real step sizes $\overline{\alpha}^{i\ell}$ $(i = 1, \ldots, \lambda)$, the real recombination step size σ_ℓ, 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 $i \in \{1, 2, \cdots, \lambda\}$ for the number of mutation points, and counter $\ell \in \{0, 1, 2, \cdots\}$ for the number of iterations, we define ingredients of `cMutation` below.

Continuous distribution directions: These directions $p_{\text{dd}}^{i\ell}$ $(i = 1, \ldots, \lambda)$ in the ℓth 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 ℓth iteration of `MATRS`, the ith 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 ℓth affine scaling matrix (updated in Section 3.5), and $p_{\text{dd}}^{i\ell}$ is the ith continuous distribution direction. Indeed, $p_{\text{md}}^{i\ell}$ $(i = 1, \ldots, \lambda)$ are chosen from $\mathcal{N}(0, M_\ell M_\ell^T)$.

Denote by $A_{:k}$ the kth column of the matrix A. Then, $A_{:,i:k}$ includes the columns between the ith and kth 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 $i \in \{1, 2, \cdots, \lambda\}$, 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^ℓ by our new combination direction

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

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 ℓth 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

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

of real distribution directions. Here κ is updated in each iteration of `cMATRS`. If no best point is found and $\kappa < \kappa_{\max}$, `cMATRS` increases κ 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 κ.

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

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

where $\alpha^{i\ell}$ is initially the initial real step size α_{init} and updated by `cLSS`. It is a scaling factor for the mutation phase in the ℓth 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 $i = 1, 2, \cdots, \lambda$ (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 $i = 1, 2, \cdots, n$) 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 ith continuous mutation point in the ℓth 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 $\mathbf{a}_K^{i\ell} = \mathbf{a}_K^{i\ell}/\nu$ ($\nu > 1$ 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 σ_ℓ 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 $\overline{\alpha}^{i\ell}$ needed for `cLSS`: Before `cLSS` is executed, it requires $\overline{\alpha}^{i\ell} \geq 1$ for each $i = 1, 2, \cdots, \lambda$ at the ℓth iteration of `MATRS`, while maintaining feasibility. To find $\overline{\alpha}^{i\ell}$, we denote $p_j := (p_{\text{md}}^{i\ell})_j$ and compute

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

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

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

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

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

is computed.

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

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

whose goal is to be neither too small nor too large to avoid line search failures. In the ℓth 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 ith 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

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

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

$$\alpha' = \min\left(\overline{\alpha}^{i\ell}, \nu\alpha\right) \tag{8}$$

and computing the new continuous trial point $x_K^{\rm trial} = x_K^{\rm best} + \alpha' p_{\rm md}^{i\ell}$ and its inexact function value $\tilde{f}^{\rm trial}$. 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 ℓth 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 $p_{\rm md}^{i\ell}$ for $i = 1, \ldots, \lambda$ in the ℓth iteration of `MATRS`, λ 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 λ 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 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 λ calls to `iLSS`. Otherwise, none of the λ 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 $\alpha^{i\ell}_{\rm init}$ $(i = 1, \ldots, \lambda)$ for `iLSS` based on the largest allowed integer step sizes $\overline{\alpha}^{i\ell}$ $(i = 1, \ldots, \lambda)$, the integer recombination step size σ_ℓ, and the list $\mathbf{a}^\ell_I$ of integer mutation step sizes that are neither too small nor too large avoiding line search failure;

- updating $\mathbf{a}^\ell_I$ in a new way that affects the determination of $\alpha^{i\ell}_{\rm init}$;

- finding integer mutation points with `iLSS` to leave regions near the saddle point or maximizer.

Integer distribution directions: Integer distribution directions $p^{i\ell}_{\rm dd} \in \mathbb{R}^{|I|}$ 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 $p^{i\ell}_{\rm md}$ (for $i = 1, 2, \cdots, \lambda$) in the ℓth 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

$$p^{{\rm init},\ell}_I = x^{{\rm best},\ell}_I - x^{{\rm rmd},\ell-1}_I, \quad p^{{\rm init},\ell}_I = \texttt{sc}\left\lceil p^{{\rm init},\ell}_I / \|p^{{\rm init},\ell}_I\|_\infty \right\rceil$$

is defined in a new way, where $x^{{\rm rmd},\ell-1}_I$ is the recombination point evaluated in the $(\ell-1)$th iteration of `MATRS` and $x^{{\rm best},\ell}_I$ is the ℓth 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^ℓ by the new combination direction $p^{{\rm init},\ell}_I$. There is a good chance to find decreases in the inexact function value by performing `iLSS` along $p^{{\rm init},\ell}_I$, leaving points with large inexact function values and going into or moving down a valley.

`iMATRS` finds a new set

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

of integer distribution directions. Here $N \geq \lambda$ is a positive tuning parameter, κ 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 κ 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 κ 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 $i = 1, 2, \cdots, \lambda$ (see (9), below). Unlike [23], we do not reduce the ith component of the vector $\mathbf{a}_I^\ell$ in the ℓth iteration of `MATRS` by $\mathbf{a}_I^{i\ell} = \lfloor \mathbf{a}_I^{i\ell}/\nu \rceil$, where $\nu > 1$ is a tuning parameter, if `iLSS` cannot find a new best point along the ith 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 σ_ℓ is the ℓth integer step size, which is computed by (27) below. Otherwise, if `iLSS` finds a new best point (as a result of extrapolation) along the ith 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 $i \in \{1, 2, \cdots, \lambda\}$ in the ℓth iteration of `MATRS` unlike `cMutation`. In this case, we compute the initial step size by the new formula

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

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

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

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

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

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

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

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

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

$$p^{\ell}_{\text{rmd}} := \sum_{i=1}^{\mu} \overline{w}_i p^{\pi i,\ell}_{\text{md}} \in \mathbb{R}^{|K|} \tag{11}$$

and the continuous recombination distribution direction

$$p^{\ell}_{\text{rdd}} := \sum_{i=1}^{\mu} \overline{w}_i p^{\pi i,\ell}_{\text{dd}} \in \mathbb{R}^{|K|} \tag{12}$$

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

$$\sum_{i=1}^{\lambda} w_i = 1 \quad \text{and} \quad w_1 \geq w_2 \geq \cdots \geq w_\mu > 0 = w_{\mu+1} = \ldots = w_\lambda \tag{13}$$

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 $i = 1, \ldots, \lambda$ satisfying (13).

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

$$M_{\ell+1} := \Big(1 - \frac{c_1 + c_\mu}{2}\Big) M_\ell + \frac{c_1}{2} M_\ell P^\sigma_\ell (P^\sigma_\ell)^T + \frac{c_\mu}{2} \sum_{i=1}^{\mu} w_i p^{\pi i,\ell}_{\text{md}} (p^{\pi i,\ell}_{\text{dd}})^T, \tag{14}$$

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

$$P^\sigma_0 := 0, \; P^\sigma_\ell := (1 - c_\sigma) P^\sigma_{\ell-1} + \overline{c}_\sigma p^\ell_{\text{rdd}}, \quad \text{for } \ell \geq 1 \tag{15}$$

is defined, where the normalization constant

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

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_μ, and c_σ 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-μ 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-μ update.

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

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

and project it into $[\sigma_{\min}, \sigma_{\max}]$. Here, e_σ 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 σ_t, $0 < \sigma_{\min} < 1$ is a minimum value for σ_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_σ, c_σ, and e_σ.

After the computation of the continuous recombination mutation direction and the updates of the real recombination step size σ_ℓ and the affine scaling matrix M_ℓ, 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 ℓth initial real step size $\alpha_{\text{init}}^\ell$ and the ℓth largest allowed real step size $\overline{\alpha}^\ell$ 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

$$\alpha^{\ell}_{\text{init}} := \min\Big(\sigma_\ell, \overline{\alpha}^{\ell}\Big), \tag{17}$$

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

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

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

Here α^{ℓ} 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^{ℓ}_{rmd}, 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 $0 < \Delta_{\min} < 1$. 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 σ_ℓ is form (16) and $0 < \Delta_{\max} < \infty$ is a tuning parameter) and repeatedly performs the following steps:

(CT$_1$) **Trust region subproblem:** We define the trust region subproblem

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

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$_2$) **Solving trust region subproblem:** We define the continuous Cauchy step

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

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

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

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

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

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

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

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

(CT$_3$) **Trust region trial point:** We compute the continuous trial point

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

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

(CT$_4$) **Trust region condition:** Given the tuning parameter $0 < \eta < \frac{1}{4}$, if the sufficient descent condition

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

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

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

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

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

As the iterations of `cTRS` are unsuccessful and $\Delta > \Delta_{\min}$, Δ 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

$$p^{\ell}_{\mathrm{rmd}} := \left\lfloor \sum_{i=1}^{\mu} \overline{w}_i p^{\pi i,\ell}_{\mathrm{md}} \right\rceil \in \mathbb{R}^{|I|}, \tag{26}$$

which is our integer recombination mutation direction. In this phase, $p^{\ell}_{\mathrm{rdd}} \in \mathbb{R}^{|I|}$ 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 $p^{i\ell}_{\mathrm{md}} = M_\ell p^{i\ell}_{\mathrm{dd}}$ 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

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

where τ_ℓ is from (16).

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

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

Here α^ℓ 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^{ℓ}_{rmd}, 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 σ_ℓ is from (27) and $0 < \Delta_{\min} < \Delta_{\max} < \infty$ are tuning parameters) and then repeatedly performs the following steps:

(IT$_1$) **Trust region subproblem:** By defining $r := -M^{-T}\tilde{g}_{\text{best}}$ and since

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

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

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

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$_2$) **Solving trust region subproblem:** We solve the bound-constrained integer least squares problem (29) by a variant of Schnorr–Euchner search [6, 11].

(IT$_3$) **Trust region trial point:** We compute the integer trial point

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

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

(IT$_4$) **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_{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 $\overline{\Delta} > 1$ and $1 < c < \infty$, as the iterations of iTRS are unsuccessful, if $\Delta > \overline{\Delta}$, Δ is reduced to

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

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

$$\Delta = \Delta - 1 \tag{31}$$

is used; because Δ is reduced faster by (30) than Δ is reduced by (31), (30) may ignore some small values for Δ. 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 α' and α'' 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 $m_{\max}$.

• 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

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

or

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

at most stuckmax until $\Delta \geq 1$. Here stuckmax ≥ 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

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

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 σ and the affince scaling matrix M in each iteration of MATRS:

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

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 form 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

and chooses the following default values for its tuning parameters:
`iMATRS`:

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

`cMATRS`:

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

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

$$\texttt{opts} = \texttt{nomadset}(\text{`}\texttt{max\_eval}\text{'},\texttt{nfmax},\text{`max\_iterations'},$$
$$2{*}\texttt{nfmax},\text{`model\_search'},\text{`1'}).$$

and that `DFLBOX` uses $\texttt{alfa\_stop} = -\infty$.

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

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

and generate the continuous bound-constrained optimization problem

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

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

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

We refer to three resulting mixed-integer problem collections: `globalMint` (598 problems), `bcpMint` (458 problems), and `princeMint` (1361 problems). With dimensions $2 \le n \le 30$ and the three noise levels $\omega = 10^{-3}, 10^{-2}, 10^{-1}$, this gives a total of 3 × 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 $2 \le n \le 30$ and the three noise levels $\omega = 10^{-3}, 10^{-2}, 10^{-1}$, this gives a total of 3 × 3180 = 1017 integer test problems.

The type of noise is absolute uniform noise, i.e., $\tilde{f} = f + (2 * \text{rand} - 1)\omega$ with 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_{init} denote the function value of the starting point (common to all solvers), f_{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

n if the target accuracy

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

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

$$\rho_s(\tau) := \frac{1}{|\mathcal{P}|}\Big|\Big\{p \in \mathcal{P} \;\Big|\; pr_{p,s} \le \tau\Big\}\Big| \tag{32}$$

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

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

$$\delta_s(\kappa) := \frac{1}{|\mathcal{P}|}\Big|\Big\{p \in \mathcal{P} \;\Big|\; cr_{p,s} \le \kappa\Big\}\Big| \tag{33}$$

is the fraction of problems solved by the solver s with κ groups of $n_p + 1$ function evaluations such that κ is the upper bound of the *cost ratio* $cr_{p,s} := \dfrac{c_{p,s}}{n_p + 1}$. 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

$$e_s := \begin{cases} \big(\min\limits_{\overline{s}\in S} c_{\overline{s}}\big)/c_s, & \text{if the solver } s \text{ solves the problem,}\\ 0, & \text{otherwise,} \end{cases}$$

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 `nf` $\geq$ `nfmax` $= 1200n$ *was reached.*
- t indicates that `sec` $\geq$ `secmax` $= 360$ seconds *was reached.*
- f indicates that the solver s *failed* for other reasons, such as bugs or algorithmic terminations. In particular, `MISO2` terminated in some cases with the message 'index in position 2 exceeds array bounds' and `MISO1` terminated because `secmax` was reached much more often than for the other solvers, even for problems with dimension 2. So changing `secmax` does not help `MISO1`.

`nf` is used as the cost measure for the data and performance profiles, and both `nf` and `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`

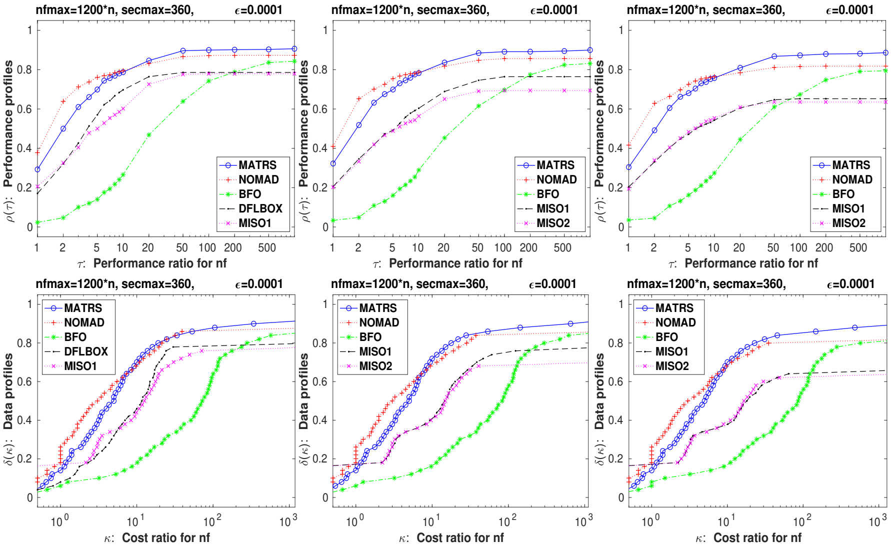

Fig. 4: Plots for `globalMint` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Performance profiles $\rho(\tau)$ (first row) are in dependence of a bound τ on the performance ratio (see (32)), while data profiles $\delta(\kappa)$ (second row) are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

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
591 of 598 problems solved						$\omega = 0.001$
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	549	49	0	0	51	50
`NOMAD`	522	0	0	76	58	41
`BFO`	513	60	0	25	10	17
`DFLBOX`	470	128	0	0	37	51
`MISO1`	466	0	132	0	35	22
`MISO2`	448	0	55	95	35	25

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
583 of 598 problems solved						$\omega = 0.01$
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	543	55	0	0	52	49
`NOMAD`	512	0	0	86	57	48
`BFO`	504	65	0	29	10	15
`MISO1`	457	0	141	0	36	22
`MISO2`	415	0	44	139	34	22
`DFLBOX`	332	266	0	0	27	37

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
574 of 598 problems solved						$\omega = 0.1$
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	535	63	0	0	50	57
`NOMAD`	489	0	0	109	57	43
`BFO`	488	71	0	39	10	16
`MISO1`	390	0	208	0	33	20
`MISO2`	380	0	47	171	34	22
`DFLBOX`	252	346	0	0	21	29

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

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`

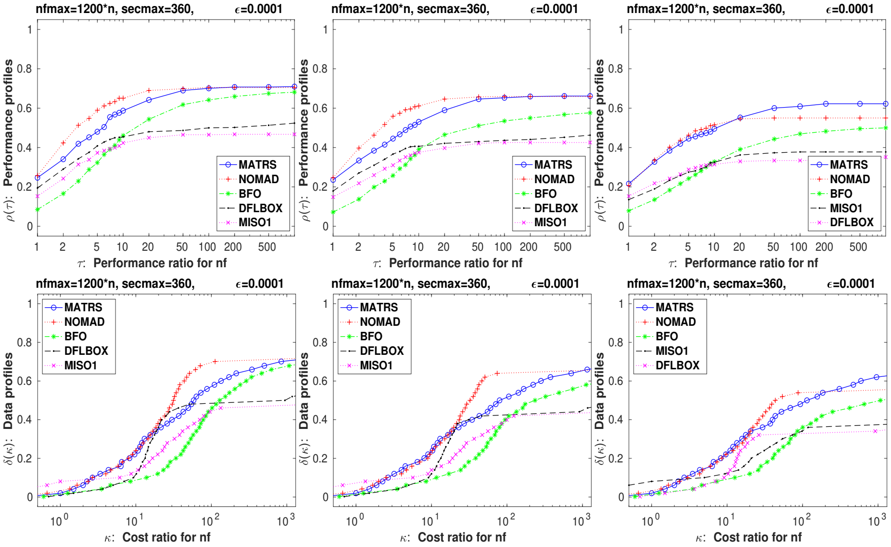

Fig. 5: Plots for `bcpMint` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200*n$						
413 of 458 problems solved					$\omega = 0.001$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	325	133	0	0	37	40
`NOMAD`	323	0	4	131	42	29
`BFO`	314	72	0	72	21	33
`DFLBOX`	240	218	0	0	30	40
`MISO1`	214	0	244	0	24	14
`MISO2`	191	0	57	210	22	13

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200*n$						
386 of 458 problems solved					$\omega = 0.01$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	303	155	0	0	34	38
`NOMAD`	302	0	3	153	39	24
`BFO`	267	67	0	124	17	27
`DFLBOX`	212	246	0	0	27	36
`MISO1`	195	0	263	0	22	14
`MISO2`	169	0	63	226	20	13

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200*n$						
352 of 458 problems solved					$\omega = 0.1$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	285	173	0	0	33	35
`NOMAD`	252	0	1	205	33	23
`BFO`	231	70	0	157	16	23
`MISO1`	173	0	285	0	20	10
`DFLBOX`	162	296	0	0	22	27
`MISO2`	159	0	52	247	19	12

Table 2: Tabulated results for `bcpMint` for dimensions $2 \le n \le 30$.

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`

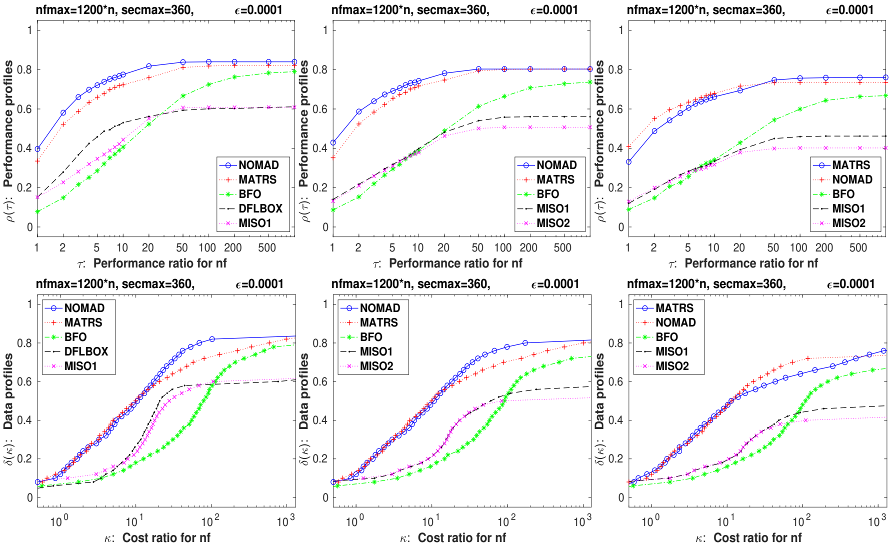

Fig. 6: Plots for `princeMint` for dimensions $2 \leq n \leq 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \leq 10^{-4}$, $\texttt{sec} \leq 360$, $\texttt{nf} \leq 1200 * n$						
1271 of 1361 problems solved					$\omega = 0.001$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`NOMAD`	1143	0	8	210	57	43
`MATRS`	1119	241	0	1	51	58
`BFO`	1079	167	0	115	19	20
`DFLBOX`	832	529	0	0	30	42
`MISO1`	829	0	532	0	25	12
`MISO2`	813	0	157	391	26	12

stopping test: $q_s \leq 10^{-4}$, $\texttt{sec} \leq 360$, $\texttt{nf} \leq 1200 * n$						
1252 of 1361 problems solved					$\omega = 0.01$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`NOMAD`	1093	0	3	265	55	35
`MATRS`	1093	267	0	1	51	60
`BFO`	1007	160	0	194	18	23
`MISO1`	763	0	598	0	24	11
`MISO2`	690	0	158	513	22	11
`DFLBOX`	606	755	0	0	24	33

stopping test: $q_s \leq 10^{-4}$, $\texttt{sec} \leq 360$, $\texttt{nf} \leq 1200 * n$						
1180 of 1361 problems solved					$\omega = 0.1$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	1035	326	0	0	47	53
`NOMAD`	1000	0	0	361	52	46
`BFO`	912	173	0	276	16	17
`MISO1`	629	0	732	0	20	9
`MISO2`	547	0	139	675	20	10
`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`

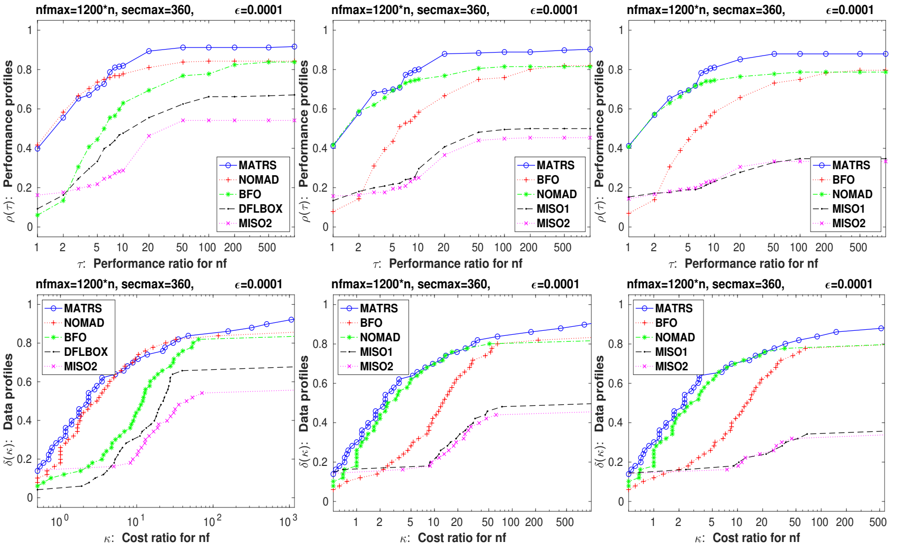

Fig. 7: Plots for `global` for dimensions $2 \leq n \leq 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
213 of 216 problems solved					$\omega = 0.001$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	199	17	0	0	56	36
`NOMAD`	182	0	0	34	57	51
`BFO`	181	0	0	35	23	55
`DFLBOX`	145	71	0	0	22	53
`MISO2`	117	0	34	65	19	12
`MISO1`	115	0	101	0	19	11

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
210 of 216 problems solved					$\omega = 0.01$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	196	20	0	0	57	38
`BFO`	178	0	0	38	23	54
`NOMAD`	176	0	0	40	54	46
`MISO1`	108	0	108	0	19	11
`MISO2`	98	0	27	91	18	11
`DFLBOX`	73	143	0	0	14	27

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
206 of 216 problems solved					$\omega = 0.1$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	190	26	0	0	57	38
`BFO`	172	0	0	44	24	54
`NOMAD`	170	0	0	46	54	50
`MISO1`	75	0	141	0	17	9
`MISO2`	72	0	26	118	17	12
`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`

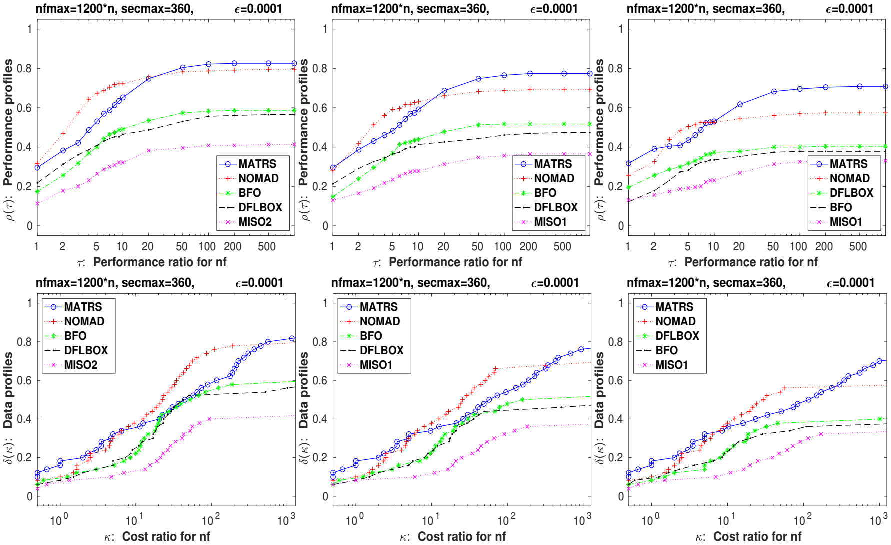

Fig. 8: Plots for `bcp` for dimensions $2 \leq n \leq 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
205 of 230 problems solved					$\omega = 0.001$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	**`nf`**	**`sec`**
`MATRS`	190	40	0	0	41	31
`NOMAD`	183	0	1	46	48	35
`BFO`	136	5	0	89	29	44
`DFLBOX`	130	100	0	0	32	44
`MISO2`	95	0	20	115	18	9
`MISO1`	91	0	139	0	18	9

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
194 of 230 problems solved					$\omega = 0.01$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	**`nf`**	**`sec`**
`MATRS`	178	52	0	0	40	31
`NOMAD`	159	0	2	69	43	35
`BFO`	119	0	0	111	25	39
`DFLBOX`	109	121	0	0	29	40
`MISO1`	84	0	146	0	17	9
`MISO2`	83	0	16	131	16	9

stopping test: $q_s \leq 10^{-4}$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
180 of 230 problems solved					$\omega = 0.1$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	**`nf`**	**`sec`**
`MATRS`	163	67	0	0	39	32
`NOMAD`	132	0	0	98	35	28
`DFLBOX`	93	137	0	0	25	34
`BFO`	87	0	0	143	19	25
`MISO1`	76	0	154	0	15	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`

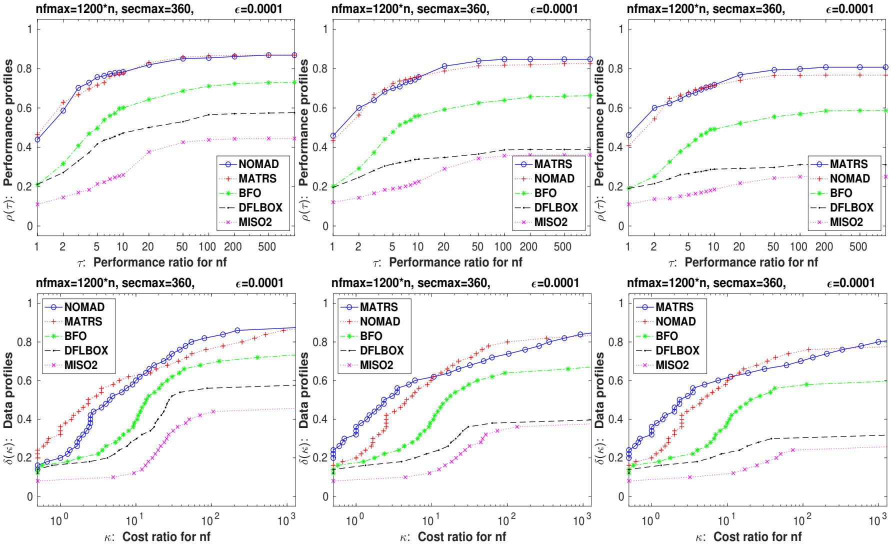

Fig. 9: Plots for `prince` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
536 of 571 problems solved					$\omega = 0.001$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`NOMAD`	496	0	3	72	60	51
`MATRS`	496	75	0	0	61	39
`BFO`	417	4	0	150	36	51
`DFLBOX`	329	242	0	0	30	44
`MISO2`	254	0	54	263	16	6
`MISO1`	231	0	340	0	16	7

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
519 of 571 problems solved					$\omega = 0.01$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	484	85	0	2	59	41
`NOMAD`	471	0	2	98	58	50
`BFO`	379	2	0	190	33	46
`DFLBOX`	222	349	0	0	25	30
`MISO2`	206	0	46	319	15	5
`MISO1`	188	0	383	0	14	8

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
490 of 571 problems solved					$\omega = 0.1$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	461	110	0	0	58	41
`NOMAD`	438	0	0	133	55	49
`BFO`	335	0	0	236	29	38
`DFLBOX`	178	393	0	0	22	24
`MISO2`	143	0	53	375	13	4
`MISO1`	135	0	436	0	12	8

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

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`

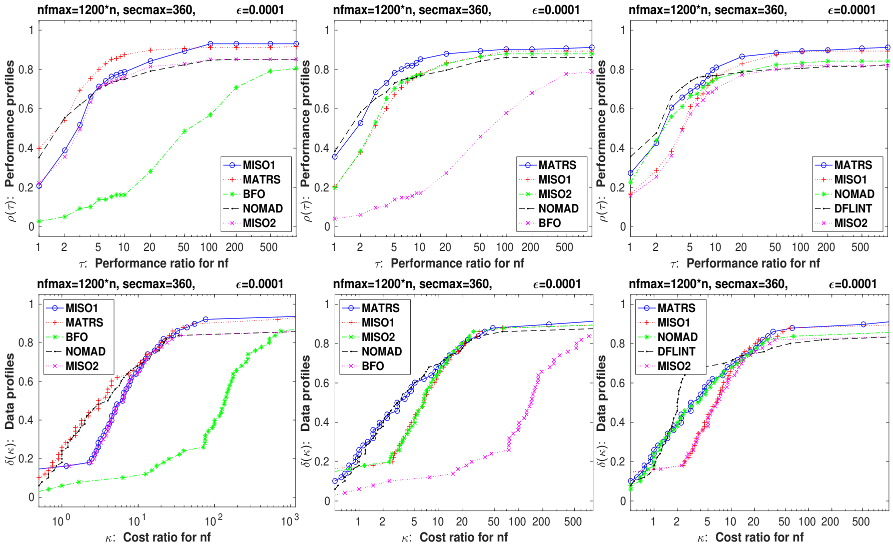

Fig. 10: Plots for `globalInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
216 of 216 problems solved					$\omega = 10^{-3}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MISO1`	201	0	15	0	36	24
`MATRS`	200	13	3	0	50	37
`BFO`	188	0	0	28	8	24
`NOMAD`	184	0	4	28	42	45
`MISO2`	184	0	7	25	35	27
`DFLINT`	184	0	21	11	54	61

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
215 of 216 problems solved					$\omega = 10^{-2}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	198	15	3	0	48	36
`MISO1`	193	0	23	0	36	22
`MISO2`	190	0	8	18	36	27
`NOMAD`	186	0	4	26	45	44
`BFO`	183	0	0	33	8	21
`DFLINT`	181	1	22	12	53	58

stopping test: $q_f \le 0.0001$, `sec` ≤ 360, `nf` $\le 1200 * n$						
214 of 216 problems solved					$\omega = 10^{-2}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	198	15	3	0	48	41
`MISO1`	193	0	23	0	36	29
`NOMAD`	182	0	7	27	44	43
`DFLINT`	178	2	21	15	53	55
`MISO2`	177	0	8	31	33	30
`BFO`	167	0	0	49	6	23

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

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`

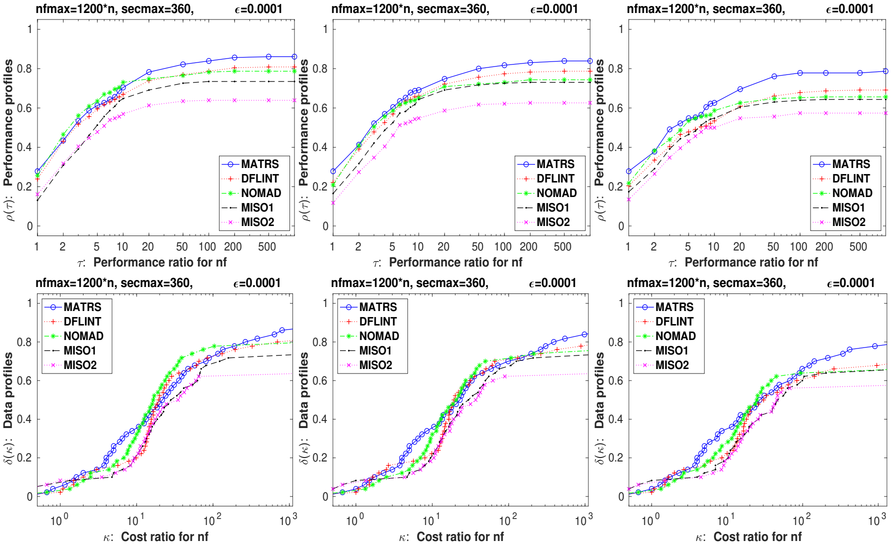

Fig. 11: Plots for `bcpInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
226 of 230 problems solved					$\omega = 10^{-3}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	198	30	2	0	42	34
`DFLINT`	186	30	3	11	41	56
`NOMAD`	181	0	1	48	43	39
`MISO1`	169	0	61	0	32	17
`MISO2`	147	0	13	70	31	17
`BFO`	105	1	0	124	18	29

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
218 of 230 problems solved					$\omega = 10^{-2}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	194	35	1	0	42	35
`DFLINT`	181	25	4	20	39	54
`NOMAD`	171	0	1	58	38	35
`MISO1`	168	0	62	0	33	19
`MISO2`	144	0	7	79	28	16
`BFO`	103	1	0	126	18	29

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
210 of 230 problems solved					$\omega = 10^{-1}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	182	47	1	0	39	34
`DFLINT`	159	39	3	29	33	46
`NOMAD`	151	0	5	74	35	33
`MISO1`	148	0	82	0	31	19
`MISO2`	132	0	10	88	27	17
`BFO`	75	1	0	154	13	20

Table 8: Tabulated results for `bcpInt` for dimensions $2 \le n \le 30$.

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`

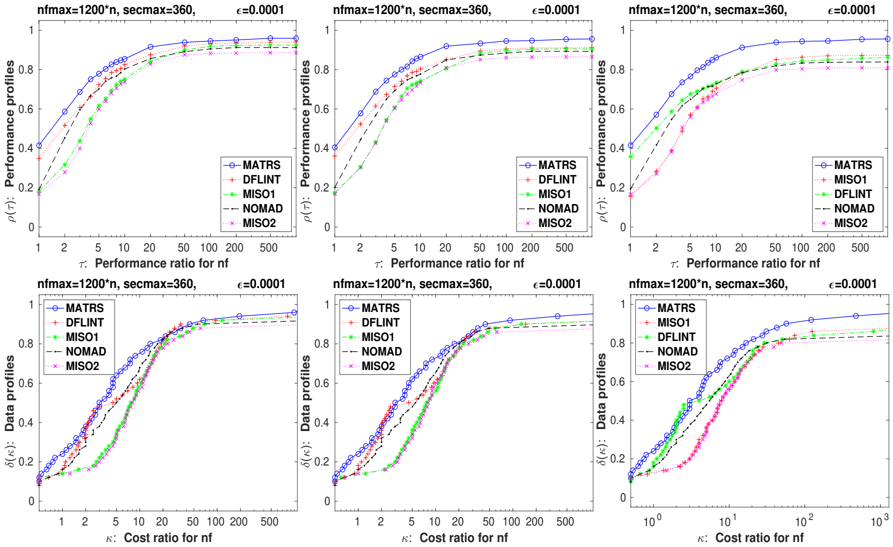

Fig. 12: Plots for `princeInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.001$ (left), 0.01 (middle), 0.1 (right). Other details as in Figure 4.

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
559 of 571 problems solved					$\omega = 10^{-3}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	548	22	0	1	59	42
`DFLINT`	537	11	6	17	53	63
`MISO1`	528	0	42	1	38	27
`NOMAD`	521	0	1	49	46	42
`MISO2`	506	0	11	54	36	27
`BFO`	444	0	0	127	15	33

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
558 of 571 problems solved					$\omega = 10^{-2}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	546	24	0	1	58	45
`DFLINT`	522	20	6	23	54	57
`MISO1`	517	0	53	1	37	28
`NOMAD`	510	0	3	58	45	38
`MISO2`	494	0	11	66	37	30
`BFO`	428	0	0	143	14	33

stopping test: $q_s \le 10^{-4}$, `sec` ≤ 360, `nf` $\le 1200 * n$						
554 of 571 problems solved					$\omega = 10^{-1}$	
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	546	24	0	1	59	52
`MISO1`	498	0	72	1	35	29
`DFLINT`	492	36	5	38	52	52
`NOMAD`	479	0	2	90	43	35
`MISO2`	462	0	10	99	34	31
`BFO`	351	0	0	220	12	27

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

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.gerad.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/dfl/ (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).