Efficient composite heuristics for integer bound constrained noisy optimization

Morteza Kimiaei

Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~kimiaei`

Arnold Neumaier

Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: `https://arnold-neumaier.at`

July 27, 2022

Abstract. This paper discusses a composite algorithm for bound constrained noisy derivative-free optimization problems with integer variables. This algorithm is an integer variant of the matrix adaptation evolution strategy. An integer derivative-free line search strategy along affine scaling matrix directions is used to generate candidate points. Each affine scaling matrix direction is a product of the corresponding affine scaling matrix (adaptively determined in a heuristic way) and a direction from a positive spanning set. If an integer derivative-free line search strategy cannot reduce the inexact function value along each weighted average of affine scaling matrix directions and its opposite direction, an integer trust region strategy is applied. The gradient of the model function is estimated by fitting and the Hessian of the model function is a symmetric matrix computed from an affine scaling matrix. The trust region subproblems are transformed into bound constrained integer triangular least squares problems, solved by a variant of Schnorr–Euchner search. Numerical results show that, compared to the state-of-the-art derivative-free integer solvers `NOMAD`, `DFLint`, and `BFO`, our algorithm is competitive on several collections of test problems.

Keywords. Integer nonlinear programming, derivative-free optimization, stochastic optimization, integer trust region method, integer evolution strategy

2000 AMS Subject Classification. 90C56

Acknowledgment The author acknowledges the financial support of the university of Vienna funded by the Austrian Science Foundation under Project No. P 34317.

1 Introduction

In this paper, we consider the problem of solving the bound constrained noisy derivative-free optimization (DFO) problem

$$\begin{array}{ll} \min & f(z) \\ \text{s.t.} & z \in \mathbf{z}, \quad z \text{ integral}, \end{array} \tag{1}$$

where

$$\mathbf{z} := \{z \in \mathbb{R}^n \mid \underline{z} \le z \le \overline{z}\} \quad \text{with } \underline{z}, \overline{z} \in \mathbb{Z}^n \ (\underline{z} < \overline{z})$$

is a *bounded box.* We assume that the smooth real-valued function

$$f : \mathbf{z} \subseteq \mathbb{Z}^n \to \mathbb{R}$$

is available only through a noisy oracle that takes z from $\mathbf{z}$ and gives an approximation $\tilde{f}(z)$ to the true function value $f(z)$. We do not assume any knowledge of the true gradients or its Lipschitz constant, of structure of the objective function, or of the statistical properties of the noise $\omega := \tilde{f}(z) - f(z)$, which may be *deterministic* or *stochastic.* Sources of deterministic noise may be modeling, truncation, and/or discretization errors, and sources of stochastic noise may be inaccurate measurements and rounding errors.

Because of a finite termination, the sequence generated by our algorithm provides an *approximate minimizer* of the problem (1), although theoretically it is not guaranteed to be a local (global) minimizer of the problem (1).

1.1 DFO methods

DFO has a long history. Comprehensive references for DFO algorithms are the books by Audet & Hare [5] and Conn et al. [12] and the paper by

LARSON et al. [25]. For investigating the behaviour of DFO algorithms for continuous problems, see the papers by MORÉ & WILD [31], RIOS & SAHINIDIS [37], KIMIAEI [22], and KIMIAEI & NEUMAIER [24]. Our new solver is a composite algorithm of discrete line search strategy, discrete evolution strategy, and discrete trust region strategy. Hence, we give an overview of these strategies in this section.

1.1.1 Line search methods

Deterministic and randomized derivative-free line search methods have been applied to DFO problems with continuous variables, see e.g., KIMIAEI & NEUMAIER [24] (randomized) and LUCIDI & SCIANDRONE [30] (deterministic). These methods choose a coordinate or random direction in each iteration and check whether or not a reduction of the inexact function value is found. As long as the inexact function values decrease along a given direction, step sizes are expanded and then trial points and their inexact function values are computed. This procedure is called *extrapolation*, which speeds up reaching an approximate minimizer. The line search methods ([30, 24]) use instead of an approximate directional derivative in the line search condition a forcing function (with non-decreasing and positive values), e.g., $\gamma\alpha^t$ with a constant factor $0 < \gamma < 1$ and a constant exponent $t > 0$, since such an approximation is poorly when noise is larger than rounding error. Inspired by continuous searches, LIUZZI et al. [27, 26] proposed a discrete line search method and continuous/discrete line search methods for bound constrained DFO problems with integer and mixed-integer variables, respectively, using extrapolation and ignoring approximate directional derivatives and forcing functions in the line search condition.

1.1.2 Matrix adaptation evolution strategies

Matrix adaptation evolution strategies (MAES) form a class of continuous derivative-free optimization algorithms (e.g., see AUGER & HANSEN [6], LOSHCHILOV et al. [29], BEYER [8], BEYER & SENDHOFF [9]). These algorithms include three phases, mutation, selection, and recombination:

• (*Mutation*) This phase is a sampling of search (candidate) points. An *affine scaling matrix* $M \in \mathbb{R}^{n\times m}$ $(m \leq n)$ determines adaptively the shape $C := MM^T$ of the distribution ellipsoid. *Search directions p of the distribution* have independent normally distributed components with zero mean and variance one, i.e., $p \sim \mathcal{N}(0, I)$ with zero mean and variance I (the identity matrix). After computing each *affine scaling matrix direction* $p_{\rm sm} := Mp$ with zero mean and variance C, each candidate point

$$y \sim x + \sigma p_{\rm sm} \sim \mathcal{N}(x, \sigma^2 C)$$

with mean x and variance $\sigma^2 C$ is computed, which is a perturbation of the vector x with zero mean. Here σ is a scaling factor.

• (*Selection*) A finite number of candidate points with the sorted inexact function values (called *selected points*) in ascending order are selected to make them the parents of the next phase.

• (*Recombination*) $x_{\rm new} := x + \sigma \overline{p}_{\rm sm}$ is selected as a new offspring for the distribution by taking a weighted average $\overline{p}_{\rm sm}$ of the directions $p_{\rm sm}$. The selected points are used to update M (or C) and compute the directions $p_{\rm sm}$.

To compute suitable matrices C and M from a sample of search points, Auger & Hansen [6] and Beyer & Sendhoff [9] use $\mathcal{O}(n^3)$ operations ($m = n$), Beyer [8] uses $\mathcal{O}(n^2)$ operations ($m = n$), Loshchilov et al. [29] use $\mathcal{O}(mn)$ operations (limited memory version). These techniques can be classified into two classes:

(i) *Unreliable estimators for covariance matrix.* Since the population size is small to have a fast search, the three following estimators are not reliable estimators for a good M (or C) :

(i-1) *Empirical covariance matrix* $C_{\rm emp}$ is the mean of the *actual realized sample*, which is an unbiased estimator of C.

(i-2) *True mean value* of the sampled distribution (the second estimator) estimates variances of sampled steps, while $C_{\rm emp}$ estimates the distribution variance within sampled points.

(i-3) *Weighted selection mechanism* is a better estimator for the distribution of selected steps, unlike the second estimator, which is an estimator of the original distribution of steps before selection.

(ii) *Reliable estimators* for M (or C). These estimators have two or three terms.

(ii-1) *Rank-μ-update* uses old information adaptively and after a sufficient number of generations it is a reliable estimator for the selected steps with a weighted selection mechanism. Indeed, this estimator is a convex combination of two terms. The first term encodes prior information, and the second term encodes learning (new information). The parameter of this convex combination, called *learning rate* for the rank-μ-update, is in $(0, 1]$. If this value is one, prior information is ignored.

(ii-2) *Rank-one-update* estimates M (or C) in a single selected step only. This estimator is a convex combination of two terms. The first term encodes prior

information (the old covariance matrix) and the second term is the maximum likelihood for a single selected step into the old covariance matrix. The parameter of this convex combination, called learning rate for the rank-one-update, is in $(0, 1]$. In the second term, the sign of the selected steps is lost for the update of M (or C). To overcome this drawback, in the second term of the rank-one-update, the single selected step is replaced by an *evaluation path* (defined by (8) below), a sequence of successive steps over a number of generations.
(ii-3) The last estimator for M (or C) is a combination of rank-μ-update and rank-one-update to take advantages of them. It includes three terms. The first term is prior information (the old covariance matrix), the second term is rank-μ-update, and the third term is rank-one-update. This estimator can be reduced to rank-μ-update or rank-one-update.

Recently, Kimiaei & Neumaier [23] proposed an efficient continuous `MAES` (called `MADFO`) for unconstrained noisy DFO problems, whose recombination phase is enriched by continuous extrapolations and many practical improvements. `MADFO` is a second-order method, while preserving its robustness and efficiency when noise is not too large. It uses an affine scaling matrix instead of quasi-Newton approximations, very useful when noise is slightly large.

1.1.3 Trust region methods

Derivative-free trust region methods have been applied to DFO problems with continuous variables, in the deterministic case, e.g., by Gratton et al. [17] and Powell [36], and in the randomized case, e.g., by Bandeira [7] and Huyer & Neumaier [20]. These methods are iterative methods in which, at each iteration, *trust region subproblems* (linear model or quadratic model) are constructed whose objective model functions are approximations to the nonlinear objective functions and whose constraint defines an upper bound (called *trust region radius*) on the norm of their feasible solutions. By restricting the trust region radius, the accuracy of these models is increased and too large steps are avoided. The gradient and the symmetric Hessian matrix of these models can be approximated by fitting or finite difference method. The *trust region ratio* measures an agreement between the objective function and the model function. The numerator of this ratio is the difference between the actual inexact function values at the previous accepted trial point and at the current trial point and its denominator is the difference between the model function values at the previous accepted trial point and at the current trial point. If this ratio is positive, the model function is a good approximation to the objective function and the iteration is said to be *successful*, the current trial point is accepted as a new point, and the trust region radius remains fixed or can be extended. Otherwise, the agreement is not good and the iteration is called *unsuccessful* and the trust region radius is reduced. In trust region methods, quadratic models are constructed, avoiding too large steps that could cause the approximate gradients of the model to be orthogonal to the approximate solution of the model. After calculating the trust region ratio, one can see whether or not the function is nearly quadratic (any effective method can be applied to solve the trust region subproblems). Two papers by Exler & Schittkowski [14] and Newby & Ali [33] propose trust region algorithms for mixed integer programming.

1.2 Software currently available

Ploskas & Sahinidis [34] investigated the behaviour of several solvers for bound constrained mixed/pure integer DFO problems. They used 176 pure-integer problems and 91 mixed-integer problems from the MIPLIP2 collection of Vigerske [38]. The solvers compared (shown in [34, Table 3]) for the bound constrained DFO problems with integer or mixed-integer variables are NOMAD [1] (an implemented version of the mesh adaptive direct search algorithms by Audet & Dennis [4] and the mesh adaptive direct search using orthogonal directions Abramson et al. [2]), BFO (an implemented version of a direct search method by Porcelli & Toint [35]), SNOBFIT (an implemented version of a branch-and-fit algorithm proposed by Huyer & Neumaier [20]), MISO (an implemented version of a mixed-integer surrogate optimization method by Müller [32]), DFLBOX (a mixed integer derivative-free line search algorithm by Liuzzi et al. [26]), TOMLAB* [19] (with several derivative-free solvers; for more details see [34, Section 3.8]), DAKOTA* [3] (with two mixed integer derivative-free optimization solvers MADS and SOGA; see for more details [34, Section 3.2]).

In a comparison among all solvers mentioned in [34], it was found that:

- MISO and NOMAD were two best solvers in terms of obtaining the best solutions.
- MISO had the best behaviour on large and binary problems.
- NOMAD had the better performance on mixed-integer, non-binary discrete, small, and medium scale problems.
- DAKOTA/MADS and SNOBFIT also performed well on most types of problems.
- BFO, DFLBOX, MISO, NOMAD, and SNOBFIT collectively solve 93% of the problems.

Another integer derivative-free line search algorithm DFLint was proposed by Liuzzi et al. [27]. DFLint was compared with NOMAD and BFO on a collection TESTINT of test problems, available at [28]. The results show that DFLint outperforms both BFO and NOMAD in terms of efficiency and robustness, especially at higher accuracies and in the presence of noise.

1.3 Our contribution

This paper discusses an *integer matrix adaptation trust region strategy* (IMATRS) for the bound constrained noisy DFO problems with integer variables in Section 2. IMATRS is an integer variant of MAES whose step sizes and directions are forced to be integral. The continuous MAES solvers are robust in the noiseless case, in the presence of rounding error, and for the medium noise. They find not necessarily better candidate points (points with lowest inexact function values) in the traditional mutation phase. In the traditional recombination phase, a new offspring is accepted, not necessarily a better point.

To generate better candidate points in our integer mutation phase (discussed in Section 2.2), an integer line search strategy intLSS (described in Section 2.1.1) is used along each affine scaling matrix direction, unlike the traditional mutation. Our selection phase (discussed in Section 2.3) is equivalent to the continuous mutation, but feasible candidate points have integer components. In the case of large noise, the sorting process in the selection phase may be incorrect, leading to a poor weighted average of the affine scaling matrix directions. If feasible candidate points can be found (but not better points), a new variant of the traditional weighted average of affine scaling matrix directions or possibly its opposite direction with randomized weights is calculated along which intLSS tries to find a new better offspring in our integer recombination phase (discussed in Section 2.4). Otherwise, if at least a better candidate point is found, the selection and recombination phases are skipped and a better candidate point among all better candidate points generated in our mutation phase is accepted as a new better offspring. If the number of feasible candidate points is less than two or such a weighted average direction is a zero step, then our integer recombination phase cannot be performed. In this case, an integer trust region strategy intTRS (discussed in Section 2.1.2) is used around the previous better offspring; see Figure 1.

To investigate *efficiency* (lowest relative cost of function evaluations) and *robustness* (highest number of solved problems), Section 3 provides exten-

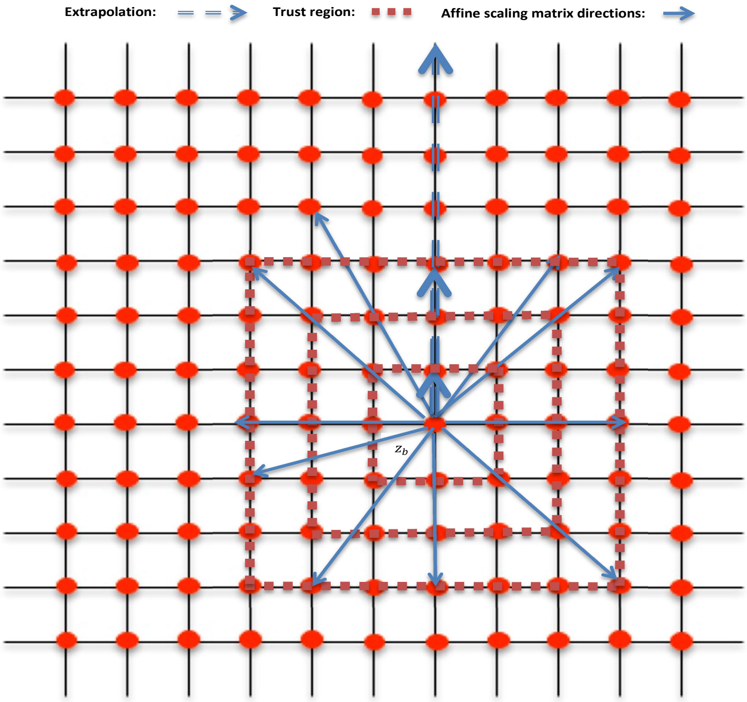

Figure 1: Some feasible candidate points and trust regions around the better offspring $z_b := z_{\text{best}}$, possibly with extrapolation steps

sive numerical experiments on several collections of bound constrained DFO problems with integer variables and summarizes them in the form of performance profiles (Dolan & Moré [13]) and data profiles (Moré & Wild [31]).

1.3.1 `intLSS`, an integer line search strategy

Integer line search strategies are effective methods for noiseless and noisy DFO problems (cf. Liuzzi et al. [27]). These strategies perform interpolation and extrapolation by reducing and expanding integer step sizes to attempt decreasing the inexact function values and finding better points (candidate and offspring). Extrapolation steps (if any) speed up reaching an approximate minimizer. This is our motivation to construct our improved integer line search `intLSS`, which can be used to enrich our integer mutation phase, possibly resulting in better feasible candidate points, and our integer recombination phase, possibly resulting in a new better offspring.

intLSS is a variant of the integer line search algorithm (NM-BBOA) by Liuzzi et al. [27, Algorithm 1] preserving its main structure, but with the three differences:

- **No non-monotone term**: intLSS does not use any non-monotone term in the line search condition; it uses a monotone line search condition.
- **A new initial step size**: After finding an initial largest step size that does not violate feasibility, we heuristically find (unlike NM-BBOA) an initial step size for line search that is smaller than the largest step size (if possible). This choice increases the chance of performing extrapolation steps.
- **New directions**: First, we choose the search directions of the distribution from a positive spanning set that can be enriched with [27, Algorithm 4]. Then, in the presence of noise, we use affine scaling matrix directions in our integer mutation phase and its weighted average in our integer recombination phase, as new directions along which intLSS is performed to get better candidate points (in our mutation phase) and better offspring (in our recombination phase).

1.3.2 intTRS, an integer trust region strategy

Continuous trust region strategies (TRS) are other effective methods for noiseless and noisy DFO problems (e.g., see Kimiaei & Neumaier [24] and Kimiaei [22]). These strategies restrict steps within a trust region to increase the accuracy of the surrogate models and avoid large steps that might be orthogonal to approximated gradients (leading an approximate maximizer or saddle point). This motivates us to generate intTRS as an integer TRS. If our integer recombination phase cannot be performed (no feasible candidate point or the zero weighted average of affine scaling matrix directions), intTRS is tried to hopefully find a decrease in the inexact function value, resulting in a new better offspring.

intTRS has several important ingredients:

- **Quadratic model estimation:** As in Huyer & Neumaier [20], the gradients of the model are estimated by fitting, while the model's symmetric Hessian matrices are symmetric variants of affine scaling matrices. This is an improvement over other model-based methods that require $\frac{1}{2}n(n+1)$ sample points to estimate the symmetric Hessian matrix.
- **Trust region subproblem:** The trust region subproblems (defined by (3) below) are transformed into integer bound constrained least squares problems (defined by (4) below) and then solved by the sils solver of Chang et al. [10].

• **A replacement for the trust region ratio:** As defined in the second paragraph of Section 1.1, the trust region ratio evaluates the amount of agreement between the function and the model function. In the presence of noise, this ratio may be very inaccurate. Instead, we use the sufficient descent condition (defined by (5) below) proposed by Kimiaei [21], which makes the trust region algorithm efficient for ill-conditioned problems with continuous variables (see [21, Section 7]). In addition, we regularize the approximate directional derivative, used in the the sufficient descent condition, to force it to be negative in the spite of possible rounding errors.

• **Update of the trust region radius:** The initial trust region radius (given as a positive integer tuning parameter) is chosen fairly large to have a global character. Later the trust region radius is decreased when a decrease in the inexact function value is not found. The trust region radius is updated based on its old value. To increase the accuracy of the model functions, we introduce a new way to take small steps by decreasing in unsuccessful iterations the trust region radius by one whenever it is below an integer tuning parameter $\overline{\Delta}$.

• **Update of the sample points:** The sets of sample points used to estimate the gradients is updated in the mutation phase and during the trust region algorithm by replacing an oldest point and its inexact function value with a new trial point and its inexact function value.

• **Termination:** In each call to `intTRS` by `IMATRS`, once the trust region radius is below one or iteration is successful, `IMATRS` terminates `intTRS`. Once the maximum number of inexact function evaluations is reached, both `intTRS` and `IMATRS` are terminated.

2 `IMATRS`, an integer matrix adaptation trust region strategy

Continuous matrix adaptation evolution strategies (`MAES`) are robust for noisy DFO problems (see, e.g., Kimiaei [22], Kimiaei & Neumaier [23], Auger & Hansen [6], Loshchilov et al. [29], Beyer [8], Beyer & Sendhoff [9]) compared to other DFO methods. These strategies compute affine scaling matrix directions inspired by quasi-Newton methods, but do not require computing gradients; hence they are second-order methods. `IMATRS` is an integer version of `MAES`, which is enriched by derivative-free integer line search and trust region strategies. As discussed in the previous section, traditional line search and trust region strategies have advantages. Their disadvantages arise when problems are ill-conditioned or when noise is high. In such cases, both traditional trust region methods and line search methods may generate zero steps, since no reduction in the inexact function value is found and trust region radii and line search step sizes become too small or even zero. Moreover, the estimates for the gradient and the Hessian are numerically poor. The reason that our solver uses a composite algorithm is that we retain the advantages of integer `MAES`, line search, and trust region strategies and overcome their disadvantages as much as possible.

`IMATRS` counts by `nf` the number of inexact function evaluations. It alternates three phases until an approximate minimizer has been found. Input of `IMATRS` are the initial point z^0 with integer components, the integer feasible set $\mathbf{z}$, the maximum number `nfmax` of inexact function evaluations, and all tuning parameters (given in Section 3). Output of `IMATRS` are the overall better offspring z_{best} and its inexact function value $\tilde{f}_{\text{best}}$.

Algorithm 1 `IMATRS`, *an integer matrix adaptation trust region strategy*

```
goal: IMATRS performs an integer composite algorithm

while 1 do
    intMutation (integer mutation phase)
    generate a finite number of candidate points if nf is below nfmax;
    if no better candidate point is found then
        selection
        perform a selection phase as in the traditional selection of MAES;
        intRecombination (integer recombination phase)
        generate a new better offspring if nf is below nfmax;
    else
        intTRS (integer trust region strategy)
        try to find a new better offspring if nf is below nfmax;
    end if
end while
```

Both `intMutation` and `intRecombination` are enriched by applying `intLSS` (integer line search strategy). To generate better feasible candidate points (points with lowest inexact function value), `intMutation` performs `intLSS` along each affine scaling matrix direction or its opposite direction, while `intRecombination` performs `intLSS` along each weighted average of affine scaling matrix directions or its opposite direction.

2.1 Two main ingredients of IMATRS

This section discusses intLSS and intTRS whose goals are to improve the traditional mutation, resulting in intMutation, and the traditional recombination, resulting in intRecombination.

2.1.1 intLSS

This section discusses the intLSS algorithm. This algorithm tries to find better candidate points and new better offspring. The main advantage of this algorithm is to perform extrapolation steps to speed up reaching an approximate minimizer if the initial step size is chosen not to be large. As discussed earlier, intLSS preserves the main structure of [27, Algorithm 1] but with some new improvements, discussed below.

$\lfloor s \rceil$ denotes the integer (or later the vector with integer components) closest to the real number (or vector) s, breaking ties in an arbitrary but fixed way (e.g., by taking the absolutely smaller value). But $\lfloor s \rfloor$ ($\lceil s \rceil$) denotes the largest (smallest) integer smaller (larger) than or equal to a real number s.

intLSS uses the $n \times 1$ vector $\mathbf{a}$ to update step sizes (only in our integer mutation phase), which are used to find step sizes used by intMutation. The vector $\mathbf{a}$ is initially a tuning vector with integer components and later is updated according to whether or not decreases in the inexact function values are found ($\mathbf{a}_i \geq 1$ for $i = 1, 2, \cdots, n$). intLSS to be started always needs an initial integer step size α_{init}. We compute α_{init} in two different ways:
• If intMutation calls intLSS to generate the ℓth candidate point ($\ell = 1, 2, \cdots, \lambda$), in this case if $\mathbf{a}_\ell < \overline{\alpha}_\ell$ holds, $(\alpha_{\text{init}})_\ell := \min(\mathbf{a}_\ell, \lfloor \sqrt{\mathbf{a}_\ell \overline{\alpha}_\ell} \rceil)$ is computed in a new way in hope of performing extrapolation steps along a given direction or its opposite direction. Otherwise, the traditional choice $(\alpha_{\text{init}})_\ell := \min(\mathbf{a}_\ell, \overline{\alpha}_\ell)$ is used as in [27]. Here $\overline{\alpha}_\ell \geq 1$ is the ℓth integer largest step size, which does not violate the feasibility, and $\sigma \geq 1$ is an integer step size found by intRecombination (see (11) below). In fact, if both $\mathbf{a}_\ell$ and $\overline{\alpha}_\ell$ are large, the traditional choice reduces the chance of applying extrapolation steps.
• In the tth iteration of IMATRS ($t = 1, 2, \cdots$), if intRecombination calls intLSS to generate a new better offspring, $(\alpha_{\text{init}})_t := \min(\sigma_t, \overline{\alpha}_t)$ is found, without using the components of $\mathbf{a}$ in the update of $(\alpha_{\text{init}})_t$. Here $\overline{\alpha}_t$ is the tth integer largest step size, which does not violate the feasibility of $\mathbf{z}$ and computed in the same way as $\overline{\alpha}_\ell$ in intMutation with the difference that

`intMutation` uses affine scaling matrix directions, while `intRecombination` uses a weighted average of these directions.

Just to simplify our notation, we ignore here the counter ℓ and t in α_{init} and $\overline{\alpha}$. After finding the initial integer step size $\alpha = \alpha_{\text{init}} \geq 1$, given the search direction $p \in \{\pm p^{\text{sm}}, \pm\overline{p}^{\text{sm}}\}$, `intLSS` computes the initial trial point $z^0 = z_{\text{trial}} = z_{\text{best}} + \alpha p$ and its inexact function value $\tilde{f}_{\text{trial}} := \tilde{f}(z_{\text{trial}})$. If $\tilde{f}_{\text{trial}} < \tilde{f}_{\text{best}}$ and $\alpha < \overline{\alpha}$ hold, then the new trial point $z_{\text{trial}} = z_{\text{best}} + \min(\overline{\alpha}, \nu\alpha)p$ ($\nu > 1$ is tuning parameter) and its inexact function value $\tilde{f}_{\text{trial}}$ are computed. As long as $\alpha < \overline{\alpha}$ and $\tilde{f}_{\text{trial}} < \tilde{f}_{\text{best}}$ hold, an extrapolation step is performed by expanding step size to $\alpha = \min(\overline{\alpha}, \nu\alpha)$ and computing the new trial point $z_{\text{trial}} = z_{\text{best}} + \min(\overline{\alpha}, \nu\alpha)p$ and its inexact function value $\tilde{f}_{\text{trial}}$. If `intLSS` uses `intMutation` to generate the ℓth candidate point, after the termination of `intLSS`, α is saved in $\mathbf{a}_\ell$ if a better offspring z_{best} is updated. Otherwise, $\mathbf{a}_\ell = \max(1, \lfloor \mathbf{a}_\ell/\nu \rfloor)$ is reduced. During `intRecombination`, $\mathbf{a}_\ell$ does not update since it does not use to find α_{init} as discussed above in the second way.

2.1.2 `intTRS`

This section describes `intTRS` to find reductions in the inexact function values when `intLSS` cannot find a reduction in the inexact function value. `intTRS` constructs bound constrained quadratic models with integer variables restricted to the intersect of the trust regions and the integer box $\mathbf{z}$ (the trust region subproblems), which is *against too large step sizes*, leading to *slow convergence* or even *failure*. The traditional trust region ratio is replaced by a regularized condition to know whether or not decrease in the inexact function value is found in the presence of noise. The initial trust region is chosen to slightly be large to have a global character. Trust region radii are reduced based on their old values in two ways to increase the accuracy of the models when radii are small. The gradients of the models are approximated by fitting and the Hessian matrices of the models are symmetric variants of affine scaling matrices.

`intTRS` has calls to the following steps as long as the condition $\Delta \geq 1$ holds, trust region iterations are unsuccessful, and `nf` is below `nfmax`:
- **A computation of the initial trust region radius:** In the first step, the initial trust region radius $\Delta = \Delta_{\text{init}}$ is chosen, where $1 < \Delta_{\text{init}} < \infty$ is an integer tuning parameter. Δ_{init} should be slightly large to have a global character.

• **A computation of the gradient of the quadratic model:** In the second step, the gradient $\tilde{g}_{\mathrm{best}} = \tilde{g}(z_{\mathrm{best}})$ at z_{best} is a solution of $\min_{\tilde{g}\in\mathbb{R}^n} \sum_{i=1}^n \phi_i(\tilde{g})$, where

$$\phi_i(\tilde{g}) := (\tilde{f}_i - \tilde{f}_{\mathrm{best}} - \tilde{g}^T s_i)/\mathtt{sc}_i \tag{2}$$

are the model errors. In (2), $s_i := z_i - z_{\mathrm{best}}$ for $i = 1, 2, \cdots, n$, z_i are the candidate points found by `intMutation`, and `sc` is an appropriate scaling vector (cf. Huyer & Neumaier [20]). The idea is to scale the estimated gradient by `sc` to remain reasonably stable in the presence of noise, since the function values can be so inaccurate.

• **The trust region subproblem:** In the third step, setting $r := -M^{-T}\tilde{g}_{\mathrm{best}}$, we transform the *trust region subproblem*

$$\begin{array}{ll} \min & \mathcal{Q}(p) = \tilde{g}_{\mathrm{best}}^T p + \dfrac{1}{2} p^T M^T M p \\ \text{s.t.} & \|p\| \le \Delta, \quad p \text{ integral}, \\ & z_{\mathrm{best}} + p \in \mathbf{z} \end{array} \tag{3}$$

into the bound constrained integer least squares problem

$$\begin{array}{ll} \min & \frac{1}{2}\|Mp - r\|_2^2 \\ \text{s.t.} & \|p\| \le \Delta, \quad p \text{ integral}, \\ & z_{\mathrm{best}} + p \in \mathbf{z}; \end{array} \tag{4}$$

because of $\tilde{g}_{\mathrm{best}}^T p + \dfrac{1}{2} p^T M^T M p = \dfrac{1}{2}\|Mp - r\|_2^2 - \dfrac{1}{2}\|r\|^2$ (the term $-\frac{1}{2}\|r\|^2$ is constant). Here M is an affine scaling matrix (estimated by (10) in Section 2.4). Now, any bound constrained integer least squares solver can be used to solve (4).

• **Computation of a new trial point and its inexact function value:** In the fourth step, `intTRS` computes the new trial point $z_{\mathrm{trial}} := z_{\mathrm{best}} + p$ and its function value $\tilde{f}_{\mathrm{trial}} := \tilde{f}(z_{\mathrm{trial}})$ and checks whether or not the maximum `nfmax` of the function evaluations is reached.

• **Updating the sample points:** In the fifth step, `intTRS` updates the sample points used to estimate the gradient for the next iterations by replacing an oldest saved point from the sample points and its inexact function value with the current trial point and its inexact function value.

• **A sufficient descent condition as a trust region condition:** In the sixth step, instead of computing the traditional trust region ratio

$$(\tilde{f}_{\mathrm{trial}} - \tilde{f}_{\mathrm{best}})/(\mathcal{Q}(0) - \mathcal{Q}(p)) \text{ with } \tilde{f}_{\mathrm{trial}} := \tilde{f}(z_{\mathrm{best}} + p) \text{ and } \tilde{f}_{\mathrm{best}} := \tilde{f}(z_{\mathrm{best}}),$$

`intTRS` computes the ratio $\rho_k := (\tilde{f}_{\mathrm{trial}} - \tilde{f}_{\mathrm{best}})/\tilde{g}^T p$. Then, if the *sufficient descent condition*

$$\rho_k|\rho_k - 1| > \eta \tag{5}$$

is satisfies, where $0 < \eta < \frac{1}{4}$ is a tuning parameter, the Boolean variable

$$\texttt{success} := (\rho_k|\rho_k - 1| > \eta)$$

is true and otherwise false. In fact, `intTRS` indicates `success` to know whether or not a reduction of the inexact function value is found.

• **Successful and unsuccessful iterations:** In the seventh step, given the tuning parameter $1 < \theta < \infty$ for reducing the trust region radius, if `success` is true, the current iteration is called *successful.* In this case, `intLSS` along p or its opposite direction is tried to possibly perform extrapolation steps and then `intTRS` ends. Otherwise, the current iteration is *unsuccessful.* In this case, we have two formulas to reduce Δ. If $\Delta > \overline{\Delta}$, $\Delta := \lfloor \Delta/\theta \rceil$ (the traditional formula) is reduced. Otherwise, $\Delta := \Delta - 1$ (the new formula) is reduced. In fact, the traditional formula reduces Δ faster than the new formula. The reason for truncating Δ is to take advantage of small steps to increase the accuracy of the model functions. Here, the integer tuning parameter $1 \leq \overline{\Delta} < \Delta_{\text{init}}$ must not be small or large.

2.2 `intMutation`

In contrasts to the continuous mutation, instead of using $p_i^t \sim \mathcal{N}(0, I)$ for $i = 1, 2, \cdots, \lambda$, we choose the search directions from a positive spanning set D, possibly enriched by [27, Algorithm 4], where $\lambda \geq 2$ is a sample size. All components of these directions are integer. Then, each affine scaling matrix direction is a multiplication of affine scaling matrix and a direction from D. All components of these affine scaling matrix directions are rounded to integer since all entries of the affine scaling matrices are not necessarily integer. We generate a population of *candidate points* (a perturbation of the old better offspring z_{best})

$$z_i^t = z_{\text{best}} + \alpha_i^t p_i^{\text{sm},t} \quad \text{for } i = 1, 2, \cdots, \lambda. \tag{6}$$

As defined earlier, t counts the number of iterations of `IMATRS`, α_i^t is initially α_{init} and updated by `intLSS`. It is a scaling factor for the mutation at the iteration t, $M_t \in \mathbb{R}^{n \times n}$ is the affine scaling matrix, p_i^t for $i = 1, 2, \cdots, \lambda$ are chosen from the enriched set D at the tth iteration, and $p_i^{\text{sm},t}$ for $i = 1, 2, \cdots, \lambda$ are affine scaling matrix directions at the tth iteration. Initially, M_0 is an identity matrix.

2.3 selection

`selection` is the same as in the continuous case. As described in Section 1.1, in this phase the inexact function values $\{\tilde{f}(z_i^t)\}_{i=1}^{\lambda} = \{\tilde{f}(z_{i:\lambda}^t)\}_{i=1}^{\lambda}$ of candidate points $\{z_{i:\lambda}^t\}_{i=1}^{\lambda} = \{z_i^t\}_{i=1}^{\lambda}$ with integer components are sorted in ascending order

$$\tilde{f}(z_{1:\lambda}^t) \leq \tilde{f}(z_{2:\lambda}^t) \leq \tilde{f}(z_{\mu:\lambda}^t) \leq \tilde{f}(z_{\mu+1:\lambda}^t) \leq \cdots \leq \tilde{f}(z_{\lambda:\lambda}^t).$$

Then, accordingly the directions $\{p_{i:\lambda}^t\}_{i=1}^{\lambda} = \{p_i^t\}_{i=1}^{\lambda}$ and affine scaling matrix directions $\{p_{i:\lambda}^{\mathrm{sm},t}\}_{i=1}^{\lambda} = \{p_i^{\mathrm{sm},t}\}_{i=1}^{\lambda}$ are sorted. All sorted information are used to generate new parents in the next iteration. Here, μ denotes the *parent number* or the *number of selected search points in the populations*. In our integer recombination, μ selected points from $\{z_{i:\lambda}^t\}_{i=1}^{\lambda}$ are used to find a new better offspring.

2.4 intRecombination

In this phase, a new variant of the traditional weighted average of affine scaling matrix directions with integer components is used as a given direction for `intLSS`. This direction in contrast to the traditional version scales the weights by a randomized vector. The reason of this scaling is that sorting process in the selection phase may be wrong due to high noise. After finding an initial step size, `intLSS` is performed along this direction or possibly its opposite direction to get a sensible reduction in the inexact function value, resulting in a new better offspring. Another task of this phase is to update the affine scaling matrix M and the step size σ, discussed below.

Computation of the recombination direction: To compute the traditional weighted average $\lceil \sum_{i=1}^{\mu} w_i p_{i:\lambda}^{\mathrm{sm},t} \rceil$ of affine scaling matrix directions $\{p_{i:\lambda}^{\mathrm{sm},t}\}_{i=1}^{\mu}$ with the integer components, the deterministic recombination weights w_i satisfy

$$\sum_j w_j = \sum_{i=1}^{\lambda} w_i \quad \text{and} \quad w_1 \geq w_2 \geq \cdots \geq w_\mu > 0 \geq w_{\mu+1} \geq w_\lambda.$$

Because of high noise, the sorting of inexact function values at the candidate points may be incorrect in the selection phase. To increase the chance of a fair sort, we product the weight vector w component-wise in the random vector

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

i.e., $\overline{w} := \beta \odot w$. Hence, the new weighted average $\overline{p}^{\mathrm{sm},t} := \lceil \sum_{i=1}^{\mu} \overline{w}_i p_{i:\lambda}^{\mathrm{sm},t} \rceil$ of $\{p_{i:\lambda}^{\mathrm{sm},t}\}_{i=1}^{\mu}$ is computed.

Updates of M and σ: Defining the *normalization constant*

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

the *variance effective selection mass*

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

and the *selected steps*

$$\overline{y}_{i:\lambda}^{t+1} := \frac{z_{i:\lambda}^{t+1} - y^t}{\sigma_t} \quad \text{for } i = 1, \cdots, \mu, \tag{7}$$

we update the *evolution path*

$$P_0^\sigma := 0, \;\; P_t^\sigma := (1 - c_\sigma) P_{t-1}^\sigma + \overline{c}_\sigma \sum_{i=1}^{\mu} w_i \overline{y}_{i:\lambda}^t, \quad \text{for } t \geq 1. \tag{8}$$

Then, because of precomputing $d_\sigma = M_t P_t^\sigma$ and reusing $p_{i:\lambda}^{\mathrm{sm},t} = M_t \overline{y}_{i:\lambda}^t$ (computed in the mutation phase before), we update the affine scaling matrix

$$M_{t+1} := \Big(1 - \frac{c_1 + c_\mu}{2}\Big) M_t + \frac{c_1}{2} d_t^\sigma (P_t^\sigma)^T + \frac{c_\mu}{2} \sum_{i=1}^{\mu} w_i p_{i:\lambda}^{\mathrm{sm},t} (\overline{y}_{i:\lambda}^t)^T \tag{9}$$

with $\mathcal{O}(n^2)$ operations (due to the vector-matrix products and sums, for more details see [8]); called fast version `fMAES` [8]. As discussed in the introduction, `MAES` estimates

$$M_{t+1} := M_t \left(I + \frac{c_1}{2} \Big(P_t^\sigma (P_t^\sigma)^T - I \Big) + \frac{c_\mu}{2} \Big(\sum_{i=1}^{\mu} w_i \overline{y}_{i:\lambda}^t (\overline{y}_{i:\lambda}^t)^T - I \Big) \right) \tag{10}$$

with $\mathcal{O}(n^3)$ operations (due to the matrix-matrix products), which is a combination of rank-μ-update (the third term) and rank-one-update (the second term). The estimator (9) is an reordering of (10). The integer step size

$$\sigma_{t+1} := \left\lfloor \sigma_t \exp\Big(\frac{c_\sigma}{d_\sigma} \Big(\frac{\|P_{t+1}^\sigma\|}{\mathbf{E}(\|u\|)} - 1 \Big) \Big) \right\rceil \tag{11}$$

is projected into $[1,\sigma_{\max}]$. Here, $\mathbf{E}(\|u\|)$ is the expected value 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, $c_\sigma \leq 1$ is a *learning rate* for the cumulation for the step size, $d_\sigma \approx 1$ is a damping parameter (cf. [18, Section 4]), $0<c_\mu \leq 1$ is a *learning rate for updating* M_{t+1}, and $c_1 \leq 1-c_\mu$ is a *learning rate for the rank-one-update* of M_{t+1}. In (9), as discussed in the previous section, both $\{p^{\text{sm},t}_{i:\lambda}\}_{i=1}^{\mu} = \{p^{\text{sm},t}_{i}\}_{i=1}^{\mu}$ and $\{\overline{y}^t_{i:\lambda}\}_{i=1}^{\mu} = \{\overline{y}^t_i\}_{i=1}^{\mu}$ were sorted by `selection`.

The entries of the matrix M_t are not necessarily integers or rounded to integers. When we compute each affine scaling matrix $p^t_{\text{sm}} := \lfloor M_t p_t \rceil$, all non-integer entries (if any) are rounded to integer values. The reason for this choice is that if non-integer entries of the matrix M_t are rounded to integer values, M_t may remain an identity matrix or some of its entries may become too large.

Requirements for `intLSS`: As discussed in Section 2.1, in the tth iteration of `IMATRS`,

$$(\alpha_{\text{init}})_t := \min(\sigma_t, \overline{\alpha}_t)$$

is computed and used as an initial step size by `intLSS` finding a better offspring, where σ_t is computed by (11) and $\overline{\alpha}_t$ is a largest step size such that feasibility preserves.

New recombination offspring: By performing `intLSS` along $\overline{p}^{\text{sm},t}$ or possibly $-\overline{p}^{\text{sm},t}$, a new better recombination offspring z_{best} can be obtained in the $(t+1)$th iteration of `IMATRS`.

3 Numerical results

In this section, we compare our solver `IMATRS` with `BFO` of Porcelli & Toint [35], `NOMAD` of Abramson et al. [1], and `DFLint` of Liuzzi et al. [27] on the five collections of test problems, `CUTEst` by Gould et al. [16] and the collections `global`, `bcp`, `inf`, and `minlp` of NLP and MINLP test problems available at

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

Denote by $x^0 \in \mathbb{R}^n$ the standard initial points of unconstrained test problems from all above five collections and let $x \in \mathbb{R}^n$. As in [28, (16)], defining

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

we generated the continuous bound constrained optimization problem

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

and then transformed this problem into the discrete bound constrained test problem

$$\begin{array}{ll} \min & f(z) := \Phi(\underline{x} + 0.01(\overline{x} - \underline{x})z) \\ \text{s.t.} & 0 \leq z_i \leq 100, \quad \text{for } i = 1, \cdots, n, \\ & z \text{ integral.} \end{array}$$

We refer to five resulting integer problem collections `CUTESTint` (285 problems), `globalInt` (216 problems), `bcpInt` (230 problems), `infInt` (207 problems), and `minlpInt` (88 problems). This gives a total of 1026 discrete test problems with dimensions $2 \leq n \leq 30$. For all test problems the initial points are $x_i^0 := 50$ for $i = 1, 2, \cdots, n$ as in [27, Section 4], except in the cases where the initial points are minimum points; in this case, the initial points are changed to $x_i^0 := 40$ for $i = 1, 2, \cdots, n$. The type of noise is absolute uniform noise, i.e., $\tilde{f} = f + (2 * \text{rand} - 1)\omega$ with the noise levels $\omega = 0.01, 0.1, 1$ and rand $\sim \mathcal{N}(0, 1)$.

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` $:= 180$ seconds of run time and at most `nfmax` $:= 200n$ function evaluations for a problem with n variables.

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

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

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

Denote by $\mathcal{S}$ the list of compared solvers and by $\mathcal{P}$ the list of problems. We say that the solver s is *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 [13]. The performance profile of the solver s

$$\rho_s(\tau) := \frac{1}{|\mathcal{P}|} \Big| \Big\{ p \in \mathcal{P} \;\Big|\; pr_{p,s} \leq \tau \Big\} \Big| \tag{12}$$

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 *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 [31]. We say that the solver s is *competetive* if it is efficient and robust. The data profile of the solver s

$$\delta_s(\kappa) := \frac{1}{|\mathcal{P}|}\Big|\Big\{p \in \mathcal{P} \;\Big|\; cr_{p,s} \leq \kappa\Big\}\Big| \tag{13}$$

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

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

called the *efficiency* (denoted by `eff` in Table 1 below) of the solver s with respect to this cost measure, which is the inverse of the performance ratio of the solver s. In Table 1, 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 $\texttt{nf} \geq \texttt{nfmax} = 200n$ *was reached.*
- t indicates that $\texttt{sec} \geq \texttt{secmax} = 180$ seconds *was reached.*
- f indicates that the solver s *failed* for other reasons.

The number `nf` of function evaluations is used as a cost measure for the data and performance profiles and all tables.

3.1 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]. NOMAD uses the following option set (model-based version)

opts = nomadset('max_eval',nfmax,'max_iterations',2*nfmax,
'min_mesh_size','1e-008','initial_mesh_size','10').

• BFO, obtained from https://github.com/m01marpor/BFO is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli & Toint [35].

• 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. [27].

These solver were used with the default parameters.

IMATRS, our implementation of techniques presented in this paper, uses the following default tuning parameters:
(i) **Evolution strategy parameters**. As in [8], $\lambda := n$, $\mu := \lfloor \frac{\lambda}{2} \rfloor$, $w_i^0 := \ln(\mu + \frac{1}{2}) - \ln i$, $w_i := \frac{w_i^0}{\sum_{j=1}^{\mu} w_j^0}$ for $i = 1, \cdots, \mu$, $\mu_w := \dfrac{1}{\sum_{j=1}^{\mu} w_j^2}$, $c_\sigma := \min\Big(1.999, \dfrac{\mu_w + 2}{n + \mu_w + 5}\Big)$, $\overline{c}_\sigma := \sqrt{c_\sigma(2 - c_\sigma)\mu_w}$, $c_1 := 2/((n+1.3)^2 + \mu_w)$, $e_\sigma := \sqrt{n}(1 - 1/(4n) - 1/(21n^2))$,

$$c_\mu := \min\left\{1 - c_1, \frac{2\Big(\mu_w - 2 + \dfrac{1}{\mu_w}\Big)}{(n+2)^2 + \mu_w}\right\}, \ d_\sigma := 1 + c_\sigma + 2\max\Big\{0, \sqrt{\frac{\mu_w - 1}{n+1}} - 1\Big\}.$$

Other parameters are $\sigma_0 := 1$ and $\sigma_{\max} := 10^{10}$.
(ii) **Trust region parameters**. $\Delta_{\rm init} := 10$, $\overline{\Delta} := 3$, $\eta := 10^{-20}$, and $\theta := 2$.
(iii) **Line search parameters**. $\nu := 2$ and $\mathbf{a}_i := 1$ for $i = 1, \cdots, \lambda$.

Matlab code can be downloaded from https://github.com/GS1400/IMATRS. We provide results for the other collections CUTESTint, bcpInt, infInt, and minlpInt and the implementation details of IMATRS in Appendix A.

3.2 Results on `globalInt`

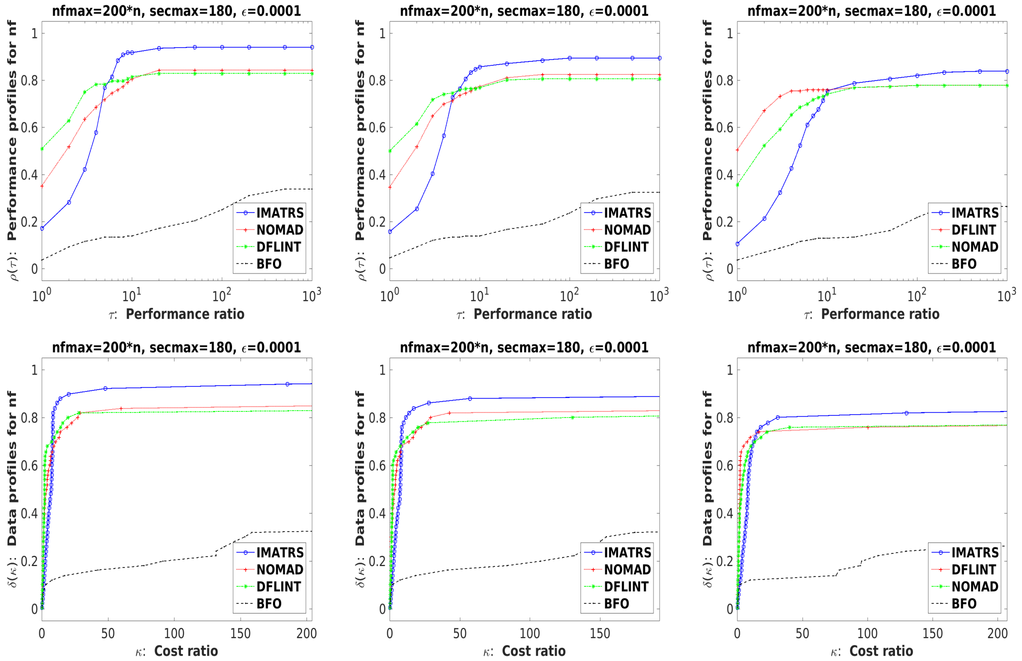

Figure 2: Results for `globalInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right). Performance profiles $\rho(\tau)$ (first row) are in dependence of a bound τ on the performance ratio (see (12)), while data profiles $\delta(\kappa)$ (second row) are in dependence of a bound κ on the cost ratio (see (13)). Problems solved by no solver are ignored.

stopping tests:						stopping tests:						stopping tests:					
$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`						$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`						$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`					
212 of 216 problems solved						209 of 216 problems solved						198 of 216 problems solved					
n∈[2,30]		# of anomalies			`eff`%	n∈[2,30]		# of anomalies			`eff`%	n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`	solver	solved	#n	#t	#f	`nf`	solver	solved	#n	#t	#f	`nf`
`IMATRS`	203	11	2	0	40	`IMATRS`	193	21	2	0	38	`IMATRS`	181	34	1	0	30
`NOMAD`	182	0	0	34	54	`NOMAD`	178	0	0	38	54	`NOMAD`	168	0	3	45	53
`DFLINT`	179	3	16	18	65	`DFLINT`	174	9	12	21	63	`DFLINT`	168	11	15	22	65
`BFO`	73	143	0	0	9	`BFO`	70	145	0	1	9	`BFO`	57	159	0	0	7

Table 1: Results for `globalInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right).

From Figure 2 and Table 1, we conclude that on the collection `globalInt` for all ω:

- `IMATRS` is more robust than the other solvers.

• DFLINT is more efficient than the other solvers.
• NOMAD is the second best robust solver.

3.3 Recommendations

Summarizing our comparison on the five collections of discrete test problems, we conclude that, mostly,
• IMATRS is more robust than the other solvers,
• DFLint is more efficient than the other solvers,
• NOMAD is the second best robust and efficient solver,
• BFO has the lowest rank in efficiency and robustness.

A Appendix: Supplementary materials

This supplementary material provides implementation details of the IMATRS solver and results for the four collections CUTESTint, bcpInt, infInt, and minlpInt.

A.1 Implementation details

In this section, we discuss some implementation details of IMATRS.

Implementation details of intTRS:
• The numerator of $\phi_i(\tilde{g})$ in (2) is $\mathcal{O}(\|s_i\|^2)$. For $i = 1, \cdots, n$, we apply a reduced QR factorization of the matrix $S := (\, s_1, \cdots, s_n \,)^T$ to compute the matrix $H := (\sum_l s_l s_l^T)^{-1}$ and then compute the scaling vector $\mathtt{sc}_i := s_i^T H s_i$.
• The affine scaling matrix M is adjusted if it is contaminated by NaN or $\pm\infty$.
• In place of solving the integer bound constrained least squares problem (4), the current implementation first ignores the bound constraints ($z_{\text{best}} + p \in \mathbf{z}$) and finds an approximate solution of the unconstrained problem by the sils solver of Chang et al. [10]. It performs the partial LLL reduction of Xie et al. [39] with sorted QR factorization and applies a variant of Schnorr–Euchner search [11, 15]. Given the solution p of sils, we project $z_{\text{best}} + p$

into the feasible set $\mathbf{z}$.

• The regularized approximate directional derivative

$$\delta := \max\{-\tilde{g}^T p, \varepsilon_m |p|^T |\tilde{g}|\}$$

is computed, where ε_m is ε-machine, and the ratio $\rho_k := (\tilde{f}_{\text{best}} - \tilde{f}_{\text{trial}})/\delta$ is adjusted. As mentioned earlier, the reason for this choice is that the traditional trust region ratio is very inaccurate in the presence of noise.

• After finding an approximated solution of the trust region subproblem if the solution is a zero solution or if the trust region subproblem is infeasible, then we terminate `intTRS`.

• After solving the trust region subproblem, whenever `intTRS` cannot generate feasible trial points that have not already been evaluated, we reduce Δ by -2 and return to solving the trust region subproblem. In this case, the chance of finding new feasible trial points increases.

Implementation details of `intRecombination`:

• After computing the weighted average of affine scaling directions, if this direction is a zero direction, we choose $\texttt{alp} \sim \mathcal{N}(0, 2I-1)$ and scale it by $\texttt{alp} = \texttt{alp}/\|\texttt{alp}\|$. Then, for $i = 1, 2, \cdots, \lambda$, we compute $\texttt{dX}_{:i} = \texttt{alp}_i(Z_{:i} - z_{\text{best}})$ and the new random subspace direction $p^{\text{sm}} = \left\lceil \sum_i \texttt{dX}_{:i} \right\rfloor$ with integer components. Here the columns of the matrix Z are the candidate points found by `intMutation`. This direction is a good replacement for the weighted average of affine scaling matrix directions when it is a zero step.

• If both `intLSS` and `intTRS` could not find a better offspring due to high noise, we accept the trial point found by `intLSS` and its inaccurate function value as the new offspring and its inexact function value. This point has a good chance to be a better offspring, because this point is a kind of subspace points and is in a neighborhood of the previous better offspring.

Implementation detail of `intLSS`: Given a direction p, to find the largest integer step size $\overline{\alpha}$, as in [27], we find the first break point

$$\tau_1 := \min\{|(\underline{z} - z_{\text{best}})_i / p_i| \mid i \in \underline{I}\}$$

if the set $\underline{I} := \{i \mid p_i < 0, \quad (z_{\text{best}})_i > \underline{z}_i\}$ is not empty. Otherwise, $\tau_1 = \infty$ is chosen. Then, we find the second break point

$$\tau_2 := \min\{|(\overline{z} - z_{\text{best}})_i / p_i| \mid i \in \underline{I}\}$$

if the set $\overline{I} := \{i \mid p_i > 0, \quad (z_{\text{best}})_i < \overline{z}_i\}$ is not empty. Otherwise, $\tau_2 = \infty$ is chosen. Next, we choose the smallest break point $\overline{\alpha} := \lfloor \min\{\tau_1, \tau_2\} \rceil$.

Implementation of σ: As in [9], to compute σ by (11),

$$\mathbf{E}(\|u\|) = \sqrt{2}\Gamma\Big(\frac{n+1}{2}\Big)/\Gamma\Big(\frac{n}{2}\Big) \approx \sqrt{n}\Big(1 - 1/(4n) - 1/(21n^2)\Big)$$

is heuristically approximated, where $u \sim \mathcal{N}(0, I)$.

Restart process: To avoid affine scaling matrix directions are not being zero steps when at least one of these directions is a zero step, we restart the matrix M and the evolution path P^σ. In this case, M is chosen as the identity matrix and P^σ is a zero vector as in the first iteration of `IMATRS`.

A.2 Results for CUTESTint

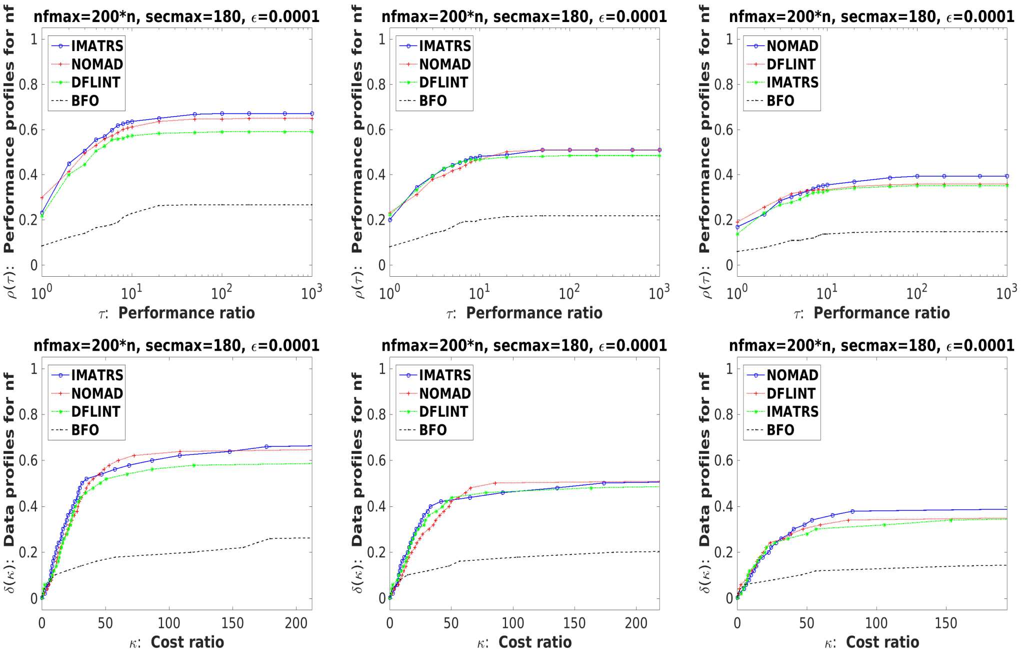

Figure 3: Results for CUTESTint for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right). Other details as in Figure 2.

stopping tests:						stopping tests:						stopping tests:					
$q_s \le 10^{-4}$, sec ≤ 180, nf ≤ 200*n						$q_s \le 10^{-4}$, sec ≤ 180, nf ≤ 200*n						$q_s \le 10^{-4}$, sec ≤ 180, nf ≤ 200*n					
230 of 285 problems solved						202 of 285 problems solved						153 of 285 problems solved					
n∈[2,30]		# of anomalies			eff%	n∈[2,30]		# of anomalies			eff%	n∈[2,30]		# of anomalies			eff%
solver	solved	#n	#t	#f	nf	solver	solved	#n	#t	#f	nf	solver	solved	#n	#t	#f	nf
IMATRS	191	93	1	0	45	IMATRS	145	138	2	0	34	NOMAD	112	0	1	172	24
NOMAD	185	0	1	99	44	NOMAD	145	0	2	138	33	DFLINT	102	125	4	54	26
DFLINT	168	72	4	41	38	DFLINT	138	83	3	61	33	IMATRS	100	184	1	0	23
BFO	76	207	0	2	13	BFO	62	221	0	2	12	BFO	42	241	0	2	8

Table 2: Results for CUTESTint for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right).

From Figure 3 and Table 2, we see that on the collection CUTESTint:

- IMATRS is more robust and efficient than the other solvers for $\omega = 0.01, 0.1$.
- NOMAD is more robust than the other solvers for $\omega = 1$.
- DFLINT is more efficient than the other solvers for $\omega = 1$.

A.3 Results for `bcpInt`

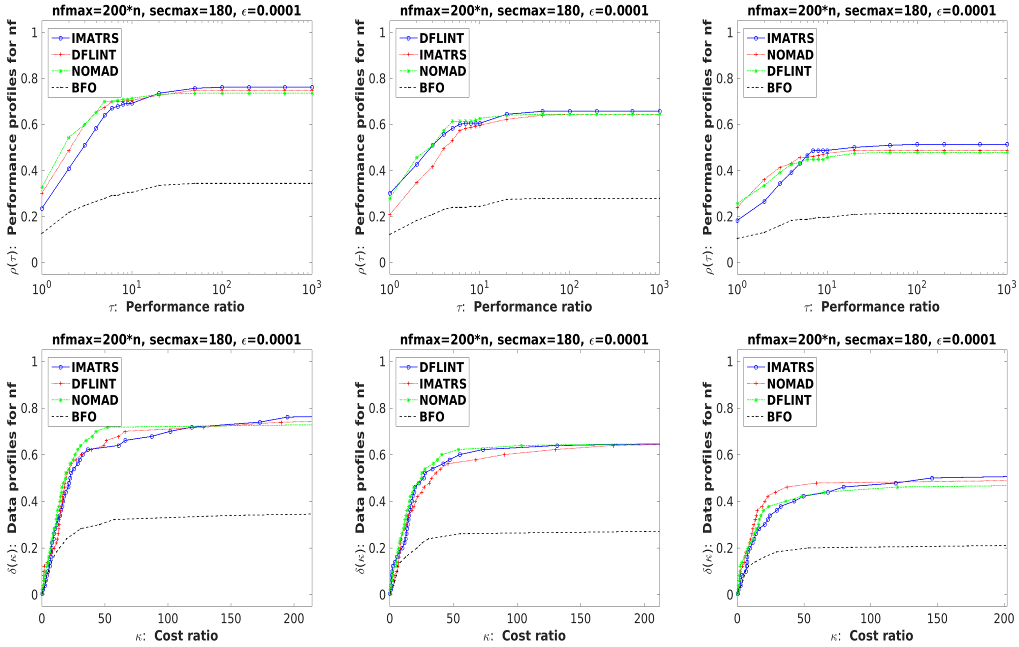

Figure 4: Results for `bcpInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right). Other details as in Figure 2.

stopping tests: $q_s \le 10^{-4}$, `sec` ≤ `180`, `nf` ≤ `200*n`					
203 of 230 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`IMATRS`	175	54	1	0	44
`DFLINT`	172	34	1	23	51
`NOMAD`	169	0	0	61	53
`BFO`	79	151	0	0	21

stopping tests: $q_s \le 10^{-4}$, `sec` ≤ `180`, `nf` ≤ `200*n`					
160 of 230 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`DFLINT`	151	52	1	26	44
`IMATRS`	148	81	1	0	38
`NOMAD`	148	0	2	80	45
`BFO`	64	166	0	0	18

stopping tests: $q_s \le 10^{-4}$, `sec` ≤ `180`, `nf` ≤ `200*n`					
167 of 230 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`IMATRS`	118	109	3	0	30
`NOMAD`	112	0	3	115	35
`DFLINT`	110	67	1	52	35
`BFO`	49	181	0	0	14

Table 3: Results for `bcpInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right).

From Figure 4 and Table 3 we see that on the collection `bcpInt`:

- `IMATRS` is more robust than the other solvers for $\omega = 0.01, 1$.
- `NOMAD` is more efficient than the other solvers for all ω.
- `DFLINT` is more robust ($\omega = 0.01$) and efficient ($\omega = 1$) than the other solvers.

A.4 Results on `infInt`

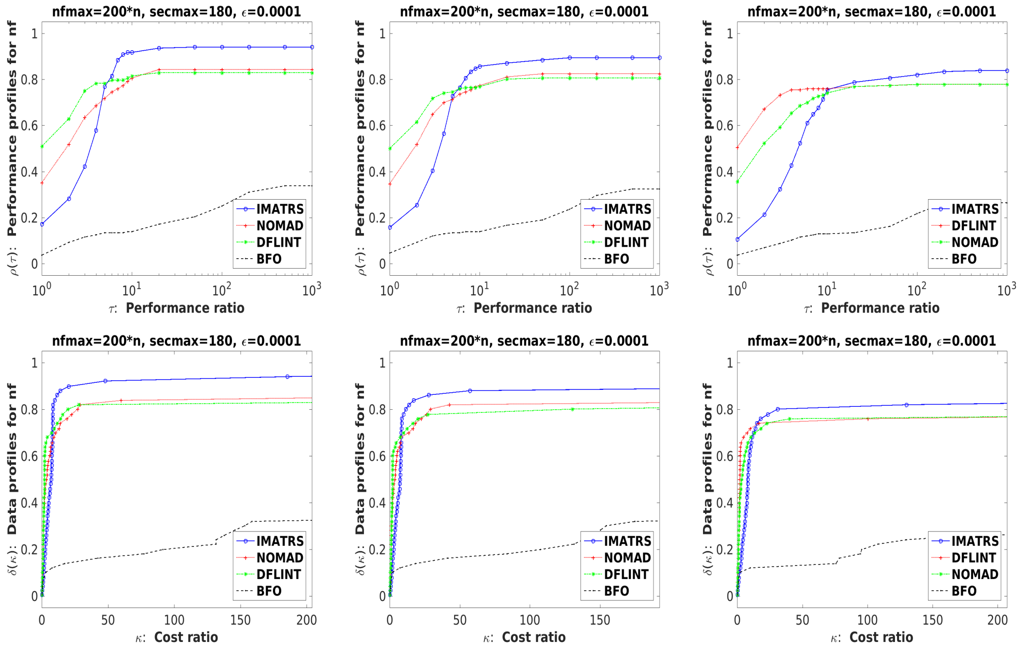

Figure 5: Results for `infInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right). Other details as in Figure 2.

stopping tests:						stopping tests:						stopping tests:					
$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`						$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`						$q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`					
181 of 207 problems solved						180 of 207 problems solved						178 of 207 problems solved					
n∈[2,30]		# of anomalies			`eff`%	n∈[2,30]		# of anomalies			`eff`%	n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`	solver	solved	#n	#t	#f	`nf`	solver	solved	#n	#t	#f	`nf`
`NOMAD`	177	0	0	30	53	`NOMAD`	173	0	0	34	53	`NOMAD`	168	0	0	39	48
`IMATRS`	175	32	0	0	45	`IMATRS`	170	37	0	0	42	`IMATRS`	162	45	0	0	32
`DFLINT`	147	4	25	31	58	`DFLINT`	147	24	25	11	60	`DFLINT`	144	23	26	14	57
`BFO`	48	159	0	0	2	`BFO`	48	159	0	0	2	`BFO`	24	183	0	0	1

Table 4: Results for `infInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right).

From Figure 5 and Table 4, we see that on the collection `infInt` for all ω:

- `NOMAD` is more robust than the other solvers.
- `IMATRS` is the second best robust.
- `DFLINT` is more efficient than the other solvers.

A.5 Results on `minlpInt`

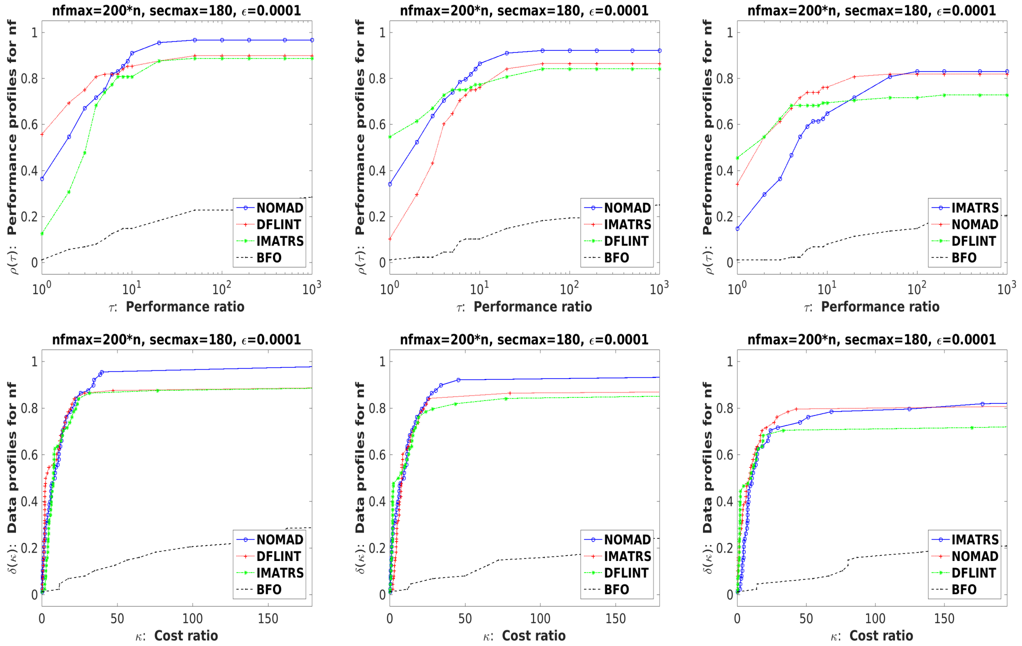

Figure 6: Results for `minlpInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right). Other details as in Figure 2.

stopping tests: $q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`					
88 of 88 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`NOMAD`	85	0	0	3	58
`DFLINT`	79	3	0	6	70
`IMATRS`	78	9	1	0	40
`BFO`	25	63	0	0	6

stopping tests: $q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`					
85 of 88 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`NOMAD`	81	0	0	7	56
`IMATRS`	76	12	0	0	38
`DFLINT`	74	5	0	9	65
`BFO`	22	66	0	0	4

stopping tests: $q_s \le 10^{-4}$, `sec` $\le$ `180`, `nf` $\le$ `200*n`					
81 of 88 problems solved					
n∈[2,30]		# of anomalies			`eff`%
solver	solved	#n	#t	#f	`nf`
`NOMAD`	72	0	0	16	54
`IMATRS`	67	20	1	0	32
`DFLINT`	64	11	0	13	59
`BFO`	18	70	0	0	2

Table 5: Results for `minlpInt` for dimensions $2 \le n \le 30$ and noise levels $\omega = 0.01$ (left), 0.1 (middle), 1 (right).

From Figure 6 and Table 5, we see that on the collection `minlpInt`:

- `NOMAD` is more robust than the other solvers for all ω.
- `DFLINT` is more efficient than the other solvers for all ω.
- `IMATRS` is the second best robust solver for $\omega = 0.1, 1$.

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] M. A. Abramson, C. Audet, J. E. Dennis, and S. Le Digabel. OrthoMADS: A deterministic MADS instance with orthogonal directions. *SIAM J. Optim* **20** (January 2009), 948–966.

[3] B. M. Adams, M. S. Ebeida, M. S. Eldred, G. Geraci, J. D. Jakeman, and etc. DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.5 user's manual. sandia national laboratories, albuquerque, NM and livermore, CA. (2016).

[4] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim* **17** (January 2006), 188–217.

[5] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization.* Springer International Publishing (2017).

[6] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation.* IEEE (2005).

[7] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim* **24** (January 2014), 1238–1264.

[8] H. G. Beyer. Design principles for matrix adaptation evolution strategies. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion.* ACM (July 2020).

[9] H. G. Beyer and B. Sendhoff. Simplify your covariance matrix adaptation evolution strategy. *IEEE Trans. Evol. Comput.* **21** (October 2017), 746–759.

[10] X. W. Chang, X. Ren, X. Xie, and T. Zhou. sils–routines for solving the standard integer least squares problem. `https://www.cs.mcgill.ca/~chang/MILES_routine1.php` (June 2016).

[11] X. W. Chang, X. Yang, and T. Zhou. MLAMBDA: A modified LAMBDA method for integer least-squares estimation. *J. Geod.* **79** (2005), 552–565.

[12] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization.* Society for Industrial and Applied Mathematics (January 2009).

[13] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (January 2002), 201–213.

[14] O. Exler and K. Schittkowski. A trust region SQP algorithm for mixed-integer nonlinear programming. *Optim. Lett.* **1** (September 2006), 269–280.

[15] A. Ghasemmehdi and E. Agrell. Faster recursions in sphere decoding. *IEEE Trans. Inf. Theory* **57** (2011), 3530–3536.

[16] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* **60** (2015), 545–557.

[17] S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.* **26** (October 2011), 873–894.

[18] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[19] K. Holmström and A. Göran. User's guide for Tomlab v3.2.1. (2002).

[20] W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.* **35** (July 2008), 1–25.

[21] M. Kimiaei. An active set trust-region method for bound-constrained optimization. *Bull. Iran. Math. Soc.* (July 2021).

[22] M. Kimiaei. VRDFON – line search in noisy unconstrained derivative-free optimization (2022).

[23] M. Kimiaei and A. Neumaier. Effective matrix adaptation strategy for noisy derivative-free optimization (2022).

[24] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.* (February 2022), 365–414.

[25] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.* **28** (May 2019), 287–404.

[26] G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for bound constrained mixed-integer optimization. *Comput. Optim. Appl.* **53** (April 2011), 505–526.

[27] 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** (February 2020), 673–702.

[28] 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).

[29] I. Loshchilov, T. Glasmachers, and H. G. Beyer. Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Trans. Evol. Comput.* **23** (April 2019), 353–358.

[30] S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.* **21** (2002), 119–142.

[31] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* **20** (January 2009), 172–191.

[32] J. Müller. MISO: mixed-integer surrogate optimization framework. *Optim. Eng.* **17** (June 2015), 177–203.

[33] E. Newby and M. M. Ali. A trust-region-based derivative free algorithm for mixed integer programming. *Comput. Optim. Appl.* **60** (April 2014), 199–229.

[34] N. Ploskas and N. V. Sahinidis. Review and comparison of algorithms and software for mixed-integer derivative-free optimization. *J. Glob. Optim.* **82** (September 2021), 433–462.

[35] M. Porcelli and P. Toint. Global and local information in structured derivative free optimization with BFO. *arXiv: Optimization and Control* (2020).

[36] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* **92** (May 2002), 555–582.

[37] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.* **56** (July 2012), 1247–1293.

[38] S. Vigerske. Proceedings of the xii global optimization workshop, mathematical and applied global optimization, mago 2014. pp. 137–140. Universidad de Almería (2014).

[39] X. Xie, X. W. Chang, and M. Al Borno. Partial LLL reduction. *arXiv preprint arXiv:1204.1398* (2012).