DGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization

Linas Stripinis[a], Remigijus Paulavičius[a,*]

[a]Vilnius University Institute of Data Science and Digital Technologies, Akademijos 4, LT-08663 Vilnius, Lithuania

ARTICLE INFO

Keywords:
DIRECT-type algorithms
Derivative-free optimization
Global optimization
Hidden constraints
Parallel computing
Benchmarking
Optimization software

ABSTRACT

In this work, we introduce DGO, a new MATLAB toolbox for derivative-free global optimization. DGO collects various deterministic derivative-free DIRECT-type algorithms for box-constrained, generally-constrained, and problems with hidden constraints. Each sequential algorithm is implemented in two different ways: using static and dynamic data structures for more efficient information storage and organization. Furthermore, parallel schemes are applied to some promising algorithms within DGO. The toolbox is equipped with a graphical user interface (GUI), which ensures the user-friendly use of all functionalities available in DGO. Available features are demonstrated in detailed computational studies using a created comprehensive library of global optimization problems. Additionally, eleven classical engineering design problems are used to illustrate the potential of DGO to solve challenging real-world problems.

1. Introduction

The DIRECT (DIviding RECTangles) algorithm [35] is a well-known and widely used solution technique for derivative-free global optimization problems. The DIRECT algorithm is an extension to classical Lipschitz optimization [57, 58, 64, 65, 71, 74], where the Lipschitz constant is not assumed to be known. This property makes DIRECT-type methods especially attractive for the solution of various real-world optimization problems (see, e.g., [1, 2, 7, 13, 15, 23, 46, 56, 60, 79] and the references given therein). Moreover, a recent review and comparison in [68] revealed that, on average, DIRECT-type algorithms' performance is one of the best among all tested state-of-the-art derivative-free global optimization approaches. The DIRECT-type algorithms often outperform algorithms belonging to other well-known classes, such as Genetic [32], Simulated annealing [38], and Particle swarm optimization [37].

While the original DIRECT addresses only box-constrained optimization problems, various DIRECT-type modifications and extensions have been proposed. Based on the type of constraints, DIRECT-type algorithms can be classified into four main categories:

- Box-constrained (see, e.g., [20, 21, 33, 23, 35, 44, 45, 46, 47, 60, 69] and the references given therein);
- Linearly-constrained/symmetric (see, e.g., [25, 55, 59, 60, 61] and the references given therein);
- Generally-constrained (see, e.g., [12, 19, 33, 40, 79] and the references given therein);
- Containing hidden constraints (see, e.g., [22, 53, 77] and the references given therein).

MATLAB [49] is one of the most broadly used mathematical computing environments in scientific and technical computing. Many widely used implementations of the original DIRECT algorithm (see, e.g., [5, 18, 22]) as well as various later introduced DIRECT-type extensions (see, e.g., [40, 43, 44, 60]), were developed using MATLAB. Motivated by this, we developed a DIRECT-type global optimization toolbox (DGO) within the MATLAB environment. The toolbox is equipped with a graphical user interface (GUI), which links to a DIRECTlib library and ensures the user-friendly use of all functionalities available in DGO.

The first publicly available DIRECT implementations and many others introduced later are typically using static data structures for data storage and organization. Our recent work [81] showed that the MATLAB implementation of the same DIRECT-type algorithm based on dynamic data structures often has a significant advantage over the performance based on more straightforward static data structures. Therefore, each algorithm in DGO is implemented using both static and dynamic data structures. As various applications can benefit from parallel computing, therefore, the SPMD (Single Program Multiple Data) parallel scheme (see [81] for more information on this) is used for the parallel implementation of some approaches.

1.1. Contributions and structure

We summarize our main contributions below:

- We develop a new MATLAB toolbox for derivative-free global optimization, consisting of 36 different DIRECT-type algorithms (see Table 1 for the details).
- We implement each DIRECT-type algorithm using two types of data structures, static and dynamic [28, 81].
- We design a user-friendly application with a graphical user interface (GUI).
- We adapt the SPMD parallel scheme [81] for the parallel implementation of selected DIRECT-type algorithms.

*Corresponding author
linas.stripinis@mif.vu.lt (L. Stripinis); remigijus.paulavicius@mif.vu.lt (R. Paulavičius)
ORCID(s): 0000-0001-9680-5847 (L. Stripinis); 0000-0003-2057-2922 (R. Paulavičius)

Problem type	Algorithm name	Description and References
Box constrained	DIRECT v4.0	Finkel's implementation [18] of the original DIRECT [35] algorithm
	DIRECT-restart	Our implementation of the algorithm from [20] (based on Finkel's DIRECT [18] implementation)
	DIRECT-m	Our implementation of the algorithm from [21] (based on Finkel's DIRECT [18] implementation)
	DIRECT-l	Our implementation of the algorithm from [23] (based on Finkel's DIRECT [18] implementation)
	DIRECT-rev	Our implementation of the algorithm from [33] (based on Finkel's DIRECT [18] implementation)
	DIRECT-a	Our implementation of the algorithm from [42] (based on Finkel's DIRECT [18] implementation)
	DIRMIN	Our implementation of the algorithm from [46] (based on Finkel's DIRECT [18] implementation)
	PLOR	Our implementation of the algorithm from [52] (based on Finkel's DIRECT [18] implementation)
	glbSolve	Björkman's implementation [5] of the original DIRECT [35] algorithm
	glbSolve-sym, glbSolve-sym2	Our implementation of algorithms from [25] (based on Björkman's glbSolve [5] implementation)
	MrDIRECT, MrDIRECT$_{075}$	Our implementation of algorithms from [44, 45] (based on Björkman's glbSolve [5] implementation)
	BIRECT	Our implementation of the algorithm from [54] (based on Björkman's glbSolve [5] implementation)
	GB-DISIMPL-C, GB-DISIMPL-V	Our implementation of algorithms from [55] (based on Björkman's glbSolve [5] implementation)
	Gb-BIRECT, BIRMIN, Gb-glbSolve	Our implementation of algorithms from [56] (based on Björkman's glbSolve [5] implementation)
	DISIMPL-C, DISIMPL-V	Our implementation of algorithms from [59] (based on Björkman's glbSolve [5] implementation)
	ADC	Our implementation of the algorithm from [69] (based on Björkman's glbSolve [5] implementation)
	Aggressive DIRECT	Our implementation of the algorithm from [1]
	DIRECT-G, DIRECT-L, DIRECT-GL	Our implementation of algorithms from [78]
Linearly constrained	Lc-DISIMPL-C, Lc-DISIMPL-V	Our implementation of algorithms from [60, 61] (based on Björkman's glbSolve [5] implementation)
Generally constrained	DIRECT-L1	Finkel's implementation of the algorithm from [18]
	DIRECT-GLc, DIRECT-GLce, DIRECT-GLce-min	Our implementation of algorithms from [79] (based on our DIRECT-GL [78] implementation)
Hidden constraints	DIRECT-NAS	Finkel's implementation of the algorithm from [22]
	DIRECT-Barrier	Our implementation of the algorithm from [22] (based on Finkel's DIRECT [18] implementation)
	subDIRECT-Barrier	Our implementation of the algorithm from [53] (based on Finkel's DIRECT [18] implementation)
	DIRECT-GLh	Our implementation of the algorithm from [77] (based on our DIRECT-GL [78] implementation)

Table 1: Classification of DIRECT-type implementations (within the DGO toolbox) based on the type of problem constraints.

- We expand a library of test and engineering global optimization problems (DIRECTlib) [80].
- We make MATLAB toolbox (DGO) open-source, i.e., freely available to anyone [76].

The rest of the paper is organized as follows. Section 2 provides the classification of existing DIRECT-type algorithms and describes in more detail algorithms implemented within our toolbox. The parallel scheme used in the implementation of some parallel algorithms is also discussed here. DGO toolbox is introduced and described in Section 3. The detailed computational study of the DGO toolbox using classical global optimization test and engineering design problems are presented in Sections 4 and 5, respectively. Finally, in Section 6, we conclude the work and discuss the possible future directions.

2. Theoretical and algorithmic backgrounds

This section provides the classification of existing DIRECT-type algorithms and describes in more detail algorithms implemented within our DGO toolbox. For a thorough review, we refer the reader to a recent survey [34].

The derivative-free DIRECT algorithm [35] is an efficient deterministic technique to solve global optimization [31, 72, 82] problems subject to simple box-constraints

$$\min_{\mathbf{x}\in D} \quad f(\mathbf{x}), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ denotes the objective function and the feasible region is an n-dimensional hyper-rectangle $D = [\mathbf{a}, \mathbf{b}] = \{\mathbf{x} \in \mathbb{R}^n : a_j \leq x_j \leq b_j, j = 1, \dots, n\}$. The objective function $f(\mathbf{x})$ is supposed to be Lipschitz-continuous (with an unknown Lipschitz constant) but can be nonlinear, non-differentiable, non-convex, and multimodal.

At the initial iteration, the DIRECT algorithm normalizes the feasible region D to be the unit hyper-cube $\bar{D}$, and refers to original space D only when evaluating the objective function. Regardless of the dimension, the first evaluation of the objective function is done at the midpoint $\mathbf{c}_1$ of $\bar{D}$. Then, the initial hyper-rectangle $\bar{D}_1$ is trisected along all of its longest dimensions, as shown in Fig. 1. New points are sampled, and the objective function is evaluated at $\mathbf{c}_j = 1/3 I^n \pm \mathbf{c}_1$, where I^n is identity matrix, and $j = 2, \dots, 2n+1$. Furthermore, $\bar{D}$ trisected into $2n+1$ smaller non-overlapping hyper-rectangles, such that the lowest function values are placed in the largest measured hyper-rectangles.

The essential step in DIRECT is the identification of potentially optimal (the most promising) hyper-rectangles (POH) of the current partition, which at the iteration k is defined as

$$\mathcal{P}^k = \{\bar{D}^k_i : i \in \mathbb{I}^k\},$$

where $\bar{D}^k_i = [\mathbf{a}^k_i, \mathbf{b}^k_i] = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq a^k_{i_j} \leq x_j \leq b^k_{i_j} \leq 1, j = 1, \dots, n, \forall i \in \mathbb{I}^k\}$ and $\mathbb{I}^k$ is the index set identifying the current partition $\mathcal{P}^k$. The next partition $\mathcal{P}^{k+1}$ is obtained after the subdivision of the selected potentially optimal hyper-rectangles from the current partition $\mathcal{P}^k$.

The selection procedure at the initial step is trivial as we have only one candidate $\bar{D}$. To make the selection of potentially optimal hyper-rectangles in the future iterations, DIRECT assesses the goodness based on the lower bound estimates for the objective function f over each hyper-rectangle $\bar{D}^k_i$. The requirement of potential optimality is stated formally in Definition 1.

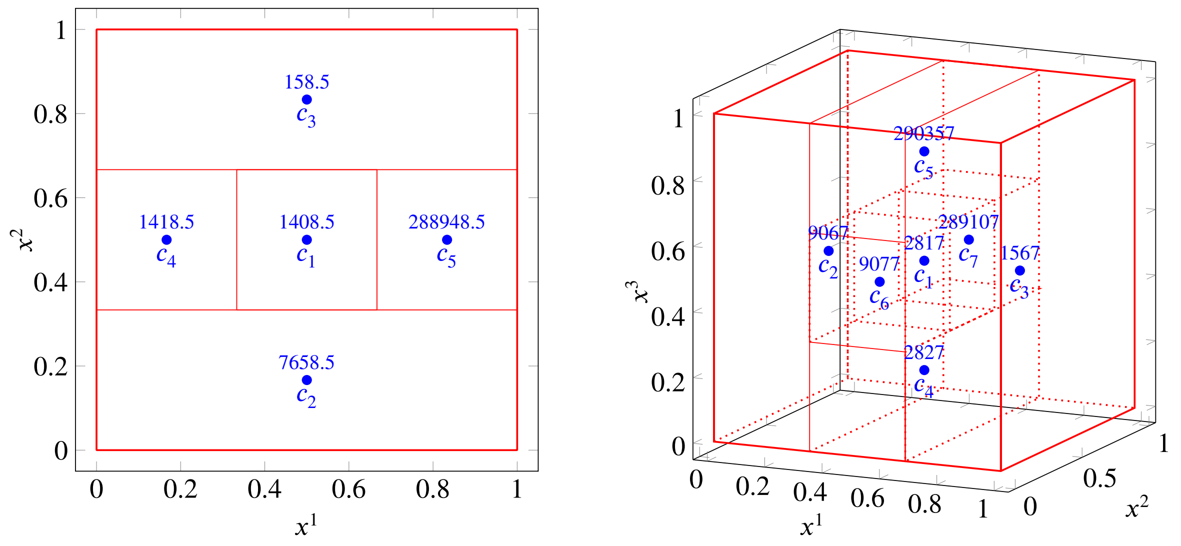


Figure 1: Visual representation of sampling and trisection techniques in DIRECT while solving two and three-dimensional *Rosenbrock* test problem.

Definition 1. *(Potentially optimal hyper-rectangle) Let* $\mathbf{c}_i^k$ *denote the center sampling point and* δ_i^k *be a measure (equivalently, sometimes called distance or size) of the hyper-rectangle* $\bar{D}_i^k$*. Let* $\varepsilon > 0$ *be a positive constant and* $\hat{f}(\hat{\mathbf{c}})$ *be the best currently found value of the objective function. A hyper-rectangle* $\bar{D}_j^k, j \in \mathbb{I}^k$ *(where* $\mathbb{I}^k$ *is the index set identifying the current partition) is said to be potentially optimal if there exists some rate-of-change (Lipschitz) constant* $\tilde{L} > 0$ *such that*

$$f(\mathbf{c}_j^k) - \tilde{L}\delta_j^k \leq f(\mathbf{c}_i^k) - \tilde{L}\delta_i^k, \quad \forall i \in \mathbb{I}^k, \qquad (2)$$

$$f(\mathbf{c}_j^k) - \tilde{L}\delta_j^k \leq \hat{f}(\hat{\mathbf{c}}) - \varepsilon|\hat{f}(\hat{\mathbf{c}})|, \qquad (3)$$

where the measure of the hyper-rectangle $\bar{D}_i^k$ *is*

$$\delta_i^k = \frac{1}{2}\|\mathbf{b}_i^k - \mathbf{a}_i^k\|_2. \qquad (4)$$

The hyper-rectangle D_j^k is potentially optimal if the lower Lipschitz bound for the objective function computed by the left-hand side of (2) is the smallest one with some positive constant $\tilde{L}$ among the hyper-rectangles of the current partition $\mathcal{P}^k$. In (3), the parameter ε is used to protect from an excessive refinement of the local minima [35, 55]. Authors obtained good results for ε values ranging from 10^{-3} to 10^{-7} in [35]. A geometrical interpretation of the selection procedure is shown in Fig. 2. Each subsequent iteration DIRECT performs a selection of POHs, which are sampled, evaluated, and trisected. Almost all DIRECT-type extensions and modifications follow the same algorithmic framework, which is summarized in Algorithm 1.

Algorithm 1: Main steps of DIRECT-type algorithms

```
1 Initialization. Normalize the search space $D$ to be
    the unit hyper-rectangle $\bar{D}$, but refer to the original
    space $D$ when making function calls. Evaluate the
    objective $f$ at the center point $\mathbf{c}_1$. Set $\hat{f} = f(\mathbf{c}_1)$,
    $\hat{\mathbf{c}} = \mathbf{c}_1$. Initialize algorithmic performance
    measures, and stopping criteria.
2 while stopping criteria are not satisfied do
3     Selection. Identify the sets $S$ of potentially
        optimal hyper-rectangles (subregions of $\bar{D}$).
4     Sampling. For each POH ($\bar{D}_j \in S$) sample and
        evaluate the objective function at new domain
        points. Update $\hat{f}, \hat{\mathbf{c}}$, and algorithmic
        performance measures.
5     Subdivision. Each POH ($\bar{D}_j \in S$) subdivide
        (trisect) and update the partitioned search
        space information.
6 end
7 Return $\hat{f}, \hat{\mathbf{c}}$, and performance measures.
```

2.1. DIRECT-type algorithms for box-constrained global optimization

Two broadly used public MATLAB implementations of the original DIRECT algorithm are DIRECT v4.0 [18] and glbSolve [5] (see Table 1). Both implementations use vector-based static data memory management to store information, contrary to a tree-structure used in the original paper [35]. Interestingly, the performance (based on the number of function evaluations) of different DIRECT implementations using the same nine standard test problems slightly differs [5]. The most significant difference is how the selection of potentially optimal hyper-rectangles (the lower convex hull, as shown in Fig. 2) is implemented. Modified Graham's scan algorithm [66] is used for this in [5], but Lemma 2.3 [22] is used in [18]. Also, the numerical tolerances used in the implementations play an essential role [5].

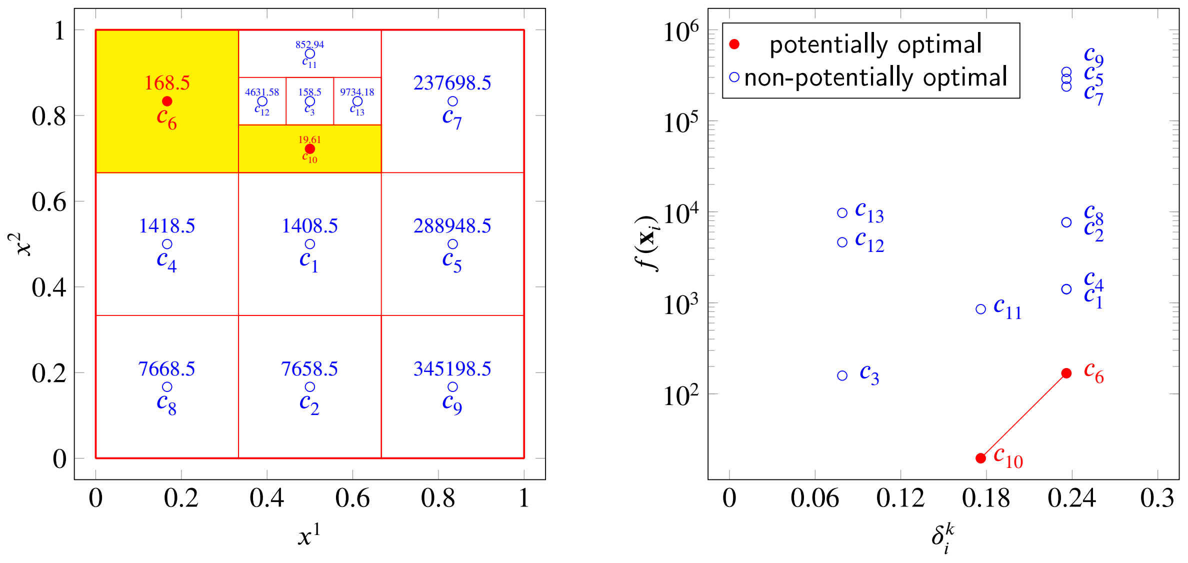


Figure 2: Visualization of selected potentially optimal rectangles in the fourth iteration of the DIRECT algorithm solving two-dimensional *Rosenbrock* test problem.

Many different DIRECT extensions have been suggested. Most of them focused on improving the selection of POHs, while others introduced new partitioning and sampling strategies. The summary of all box-constrained proposals considered in the DGO toolbox is given in Table 2. Most algorithms are based on the trisection of n-dimensional POHs, and just ADC, BIRECT, and both DISIMPL versions use different partitioning strategies. Below we briefly review the DIRECT-type approaches for box-constrained global optimization implemented in the current release of the DGO toolbox.

Adaptive diagonal curves (ADC) based algorithm was introduced in [69]. Independently of the problem dimension, the ADC algorithm evaluates the objective function $f(\mathbf{x})$ only at two vertices $\mathbf{a}_i^k$ and $\mathbf{b}_i^k$ of the main diagonals of each hyper-rectangles $\bar{D}_i^k$. Additionally, a new two-phase technique balancing local and global information usage has also been incorporated. When the ADC algorithm performs a sufficient number of hyper-rectangle subdivisions near the current best point, the two-phase approach forces the new algorithm to switch to exploring a larger hyper-rectangle that could contain better solutions.

BIRECT (BIsecting RECTangles) [54] is motivated by the diagonal partitioning strategy [69, 70, 72]. The bisection is more appropriate to the trisection because of the shape of hyper-rectangles. Still, usual sampling strategies at the center or the diagonal's endpoints are not appealing for bisection. In BIRECT, the objective function is evaluated at two points on the diagonal equidistant between themselves and a diagonal's vertices. Such a sampling strategy entitles the reuse of the sampling points in descendant hyper-rectangles. Additionally, more comprehensive information about the objective function is considered compared to the central sampling strategy used in most DIRECT-type algorithms.

In DISIMPL [59], instead of hyper-rectangles, simplicial partitions are considered. At the first iteration, the hyper-cube $\bar{D}$ is partitioned into $n!$ simplices by the standard face-to-face simplicial division based on the combinatorial vertex triangulation. After this, all simplices share the diagonal of the feasible region and have equal hyper-volume. In [59], we proposed two different sampling strategies. Both are included in the DGO toolbox: i) DISIMPL-C evaluating the objective function at the geometric center of the simplex; ii) DISIMPL-V evaluating the objective function on all unique vertices of the simplex. For box-constrained problems, the number of initial simplices increases speedily. Therefore, DISIMPL effectively can be used only for small box-constrained problems. However, the DISIMPL approach is auspicious (among all DIRECT-type methods) for symmetric optimization problems [59, 60] and problems with linear constraints [61].

In the DIRECT-restart algorithm [20], the authors introduced an adaptive scheme for the ε parameter. Condition (3) is needed to stop the DIRECT from wasting function evaluations on minor hyper-rectangles where only a negligible improvement can be expected. The DIRECT-restart algorithm starts with $\varepsilon = 0$, and the same value for ε is maintained while improvement is achieved. However, if five consecutive iterations have no improvement in the best function value, the search may be stagnated around a local optimum. Therefore, the algorithm switches to $\varepsilon = 0.01$ value, seeking to prevent an excessive local search.

The authors of MrDIRECT [45] and MrDIRECT$_{075}$ [41] algorithms introduced three different levels to perform the selection procedure:

1. At level 2, DIRECT is run as usual, with $\varepsilon = 10^{-5}$.

2. At level 1, the selection is limited to only 90% of $\bar{D}_i^k$

in the partition $\mathcal{P}^k$; 10% of the largest measure hyper-rectangles, where the objective function f values are worst, are ignored. A value of $\varepsilon = 10^{-7}$ is used.

3. At level 0, the selection is limited to only 10% of the best hyper-rectangles from level 1. At this level, a value of $\varepsilon = 0$ is used.

Both algorithms cycle through these levels using "*W-cycle*": 21011012. The main difference between the proposed algorithms is that MrDIRECT uses fixed $\varepsilon = 10^{-4}$ value at all levels, while MrDIRECT$_{075}$ follows the rules described above.

In [1], authors relaxed the criteria of selection of POHs and proposed an aggressive version of the DIRECT algorithm. Aggressive DIRECT's main idea is to select and divide at least one hyper-rectangle from each group of different diameters (δ_i^k) having the lowest function value. Therefore, using Aggressive DIRECT in the situation presented in Fig. 2, a hyper-rectangle with the center point c_3 would also be selected and divided. The aggressive version performs much more function evaluations per iteration compared to other DIRECT-type methods. From the optimization point of view, such an approach seems less favorable since it "wastes" function evaluations by exploring unnecessary (non-potentially optimal) hyper-rectangles. However, such a strategy is much more appealing in a parallel environment, as was shown in [26, 27, 29, 86].

In [23] the algorithm named DIRECT-l was proposed. In most of DIRECT-type algorithms, the size of the hyper-rectangle is measured by a half-length of a diagonal (see (4)). In DIRECT-l, the measure of a hyper-rectangle is evaluated by the length of its longest side . Such a measure corresponds to the infinity norm and allows the DIRECT-l algorithm to group more hyper-rectangles with the same measure. Thus, there are fewer distinct measures, and therefore, fewer potentially optimal hyper-rectangles are selected. Moreover, with DIRECT-l at most one hyper-rectangle selected from each group, even if there is more than one potentially optimal hyper-rectangle in the same group. Such a strategy allows a reduction in the number of divisions within a group.

In [21], the authors concluded that the original DIRECT algorithm is sensitive to the objective function's additive scaling. Additionally, the algorithm does not operate well when the values of the objective function are large enough. To overcome this, the authors proposed a scaling of function values by subtracting the median of the collected function values. DIRECT-m replaces the equation (3) in Definition 1 to:

$$f(\mathbf{c}_j) - \tilde{L}\delta_j \leq \hat{f}(\hat{\mathbf{c}}) - \varepsilon|\hat{f}(\hat{\mathbf{c}}) - \hat{f}^{median}|. \qquad (5)$$

Similarly, in [42], the authors extended the same idea in DIRECT-a to reduce the objective function's additive scaling. At each iteration, instead of the median value ($\hat{f}^{median}$), authors proposed to use the average value ($\hat{f}^{average}$):

$$f(\mathbf{c}_j) - \tilde{L}\delta_j \leq \hat{f}(\hat{\mathbf{c}}) - \varepsilon|\hat{f}(\hat{\mathbf{c}}) - \hat{f}^{average}|. \qquad (6)$$

Another extension of the DIRECT algorithm was proposed in [25]. Here, the authors introduced glbSolve-sym (glbSolve-sym2) as DIRECT extensions for symmetric Lipschitz continuous functions. When solving symmetric optimization problems, there exist equivalent subregions in the hyper-rectangle. To avoid exploration over equivalent subregions, for all newly generated hyper-rectangles, the algorithm determines which hyper-rectangles can be safely discarded (because of the problem's symmetrical nature) from the further search.

In the PLOR algorithm [52], the set of all Lipschitz constants (here with the set of potentially optimal hyper-rectangles) is reduced to just two: with the maximal diameter ($\delta^k_{\max}$) and the minimal one ($\delta^k_{\min}$). In such a way, the PLOR approach is independent of any user-defined parameters and balances equally local and global searches during the optimization process.

In our recent DIRECT extension, DIRECT-GL [78], we introduced a new approach to identifying the extended set of potentially optimal hyper-rectangles. Here, using a novel two-step-based strategy, the set of the best hyper-rectangles is enlarged by adding more medium-measured hyper-rectangles with the smallest function value at their centers and, additionally, closest to the current minimum point. The first step of the selection procedure forces the DIRECT-GL algorithm to work more globally (compared to the selection used in DIRECT [35]). In contrast, the second step assures a faster and broader examination around the current minimum point. The original DIRECT-GL version performs a selection of potentially optimal hyper-rectangles in each iteration twice [78], and the algorithm separately handles the found independent sets G (using Definition 2 from [78] - DIRECT-G) and L (using Definition 3 from [78] - DIRECT-L). Following the same trend from [81], the version used in this paper slightly differs compared to [78]. In the current version of DIRECT-GL, identifying these two sets is performed in succession, and the unique union of these two sets ($S = G \cup L$) is used in Algorithm 1, Line 3. This modification was introduced, aiming to reduce the data dependency in the algorithm seeking a better parallelization.

Several different globally biased (Gb-) versions of DIRECT-type algorithms were introduced and investigated [55, 56]. Proposed approaches are primarily oriented for solving multidimensional optimization problems and contain a phase that constrains itself to large subregions. The introduced step performs until a sufficient number of divisions of hyper-rectangles near the current best point is done. Once those subdivisions around the current best minima point are performed, the neighborhood contains only small measure hyper-rectangles and all larger ones located far away from it. Therefore, the two-phase strategy makes the DIRECT-type algorithms examine larger hyper-rectangles and return to the general phase only when an improved minimum is obtained. Within this toolbox, the proposed globally biased strategy is combined with glbSolve, BIRECT, DISIMPL-C, and DISIMPL-V algorithmic frameworks.

Finally, three different hybridized DIRECT-type algorithms are proposed (DIRECT-rev [33], DIRMIN [46], BIRMIN [56]). In our implementation, all algorithms are combined with the same local search routine – *fmincon*. The DIRMIN algorithm suggested running a local search starting from the midpoint of every potentially optimal rectangle. However, such an approach likely generates more local searches than necessary, as many of the starting points will converge to the same local optimum. The other authors of the DIRECT-rev and BIRMIN algorithms tried to minimize the usage of local searches. They suggested using *fmincon* only when some improvement in the best current solution is obtained. The authors in the [33] additionally incorporated the following two enhancements. First, in the DIRECT-rev algorithm, selected hyper-rectangles are trisected only on one longest side. Second, only one potentially optimal hyper-rectangle is selected if several equally good exist (the same diameter and objective values) in Definition 1.

2.2. DIRECT-type algorithms for generally constrained global optimization

The original DIRECT algorithm [35] solves optimization problems only with bounds on the variables. In this subsection, we consider a generally constrained global optimization problem of the form:

$$\begin{aligned} &\min_{\mathbf{x}\in D} f(\mathbf{x}) \\ &\text{s.t. } \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \\ &\qquad \mathbf{h}(\mathbf{x}) = \mathbf{0}, \end{aligned} \tag{7}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^r$ are (possibly nonlinear) continuous functions. The feasible region is a nonempty set, consisting of points that satisfy all constraints, i.e., $D^{\text{feas}} = D \cap \Omega \neq \emptyset$, where $\Omega = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \mathbf{h}(\mathbf{x}) = \mathbf{0}\}$. As for the box-constrained problems, it is also assumed that the objective and all constraint functions are Lipschitz-continuous (with unknown Lipschitz constants) but can be nonlinear, non-differentiable, non-convex, and multimodal.

The first DIRECT-type algorithm for problems with general constraints was introduced in [33]. Finkel in [19] investigated three different constraint handling schemes within the DIRECT framework. The comparison revealed various disadvantages of the initial proposals. Recently, various new promising extensions for general global optimization problems were introduced (see, e.g., [3, 12, 40, 62, 63, 79] and the references given therein). Below we briefly review the approaches implemented in the current release of the DGO toolbox (see Table 1).

An exact L1 penalty approach DIRECT-L1 [18] is transforming the original constrained problem (7) in the form:

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) + \sum_{i=1}^{m} \max\{\gamma_i g_i(\mathbf{x}), 0\} + \sum_{i=1}^{r} \gamma_{i+m}|h_i(\mathbf{x})|, \tag{8}$$

where γ_i are penalty parameters. Experiments in [19] showed promising results of this approach. Nevertheless, the biggest drawback is the users' requirement to set penalty parameters for each constraint function manually. In practice, choosing penalty parameters is an essential task and can significantly impact the performance of the algorithm [19, 40, 60, 61, 79].

In [79], we have introduced a new DIRECT-type extension based on the DIRECT-GL [78] algorithm. The new DIRECT-GLce algorithm uses an auxiliary function approach that combines information on the objective and constraint functions and does not require any penalty parameters. The DIRECT-GLce algorithm works in two phases, where during the first phase, the algorithm finds feasible points and in the second phase improves a feasible solution. A separate step for handling infeasible initial points is beneficial when the feasible region is small compared to the entire search space. In the first phase, DIRECT-GLce samples the search space and minimizes the sum of constraint violations, i.e.:

$$\min_{\mathbf{x}\in D} \varphi(\mathbf{x}), \tag{9}$$

where

$$\varphi(\mathbf{x}) = \sum_{i=1}^{m} \max\{g_i(\mathbf{x}), 0\} + \sum_{i=1}^{r} |h_i(\mathbf{x})|. \tag{10}$$

The algorithm works in this phase until at least one feasible point ($\mathbf{x} \in D^{\text{feas}}_{\varepsilon_\varphi}$) is found, where

$$D^{\text{feas}}_{\varepsilon_\varphi} = \{\mathbf{x} : 0 \leq \varphi(\mathbf{x}) \leq \varepsilon_\varphi, \mathbf{x} \in D\}. \tag{11}$$

When feasible points are located, the effort is switched for improving the feasible solutions. In the second phase, DIRECT-GLce uses the transformed problem (7):

$$\begin{aligned} &\min_{\mathbf{x}\in D} f(\mathbf{x}) + \tilde{\xi}(\mathbf{x}, \hat{f}^{\text{feas}}), \\ &\tilde{\xi}(\mathbf{x}, \hat{f}^{\text{feas}}) = \begin{cases} 0, & \mathbf{x} \in D^{\text{feas}}_{\varepsilon_\varphi} \\ 0, & \mathbf{x} \in D^{\text{inf}}_{\varepsilon_{\text{cons}}} \\ \varphi(\mathbf{x}) + \Delta, & \text{otherwise,} \end{cases} \end{aligned} \tag{12}$$

where

$$D^{\text{inf}}_{\varepsilon_{\text{cons}}} = \{\mathbf{x} : f(\mathbf{x}) \leq \hat{f}^{\text{feas}}, \varepsilon_\varphi < \varphi(\mathbf{x}) \leq \varepsilon_{\text{cons}}, \mathbf{x} \in D\}, \tag{13}$$

and $\varepsilon_{\text{cons}}$ is a small tolerance for constraint function sum, which automatically varies during the optimization process. An auxiliary function $\xi(\mathbf{x}, \hat{f}^{\text{feas}})$ depends on the sum of the constraint functions and the parameter $\Delta = |f(\mathbf{x}) - \hat{f}^{\text{feas}}|$, which is equal to the absolute value of the difference between the best feasible function value found so far ($\hat{f}^{\text{feas}}$)

Step / Alg.	*Partitioning scheme*	*Sampling scheme*	*Selection of POH & comments on additional steps, if any*			*Input param.*
Aggressive DIRECT	Hyper-rectangular partitions based on n-dimensional trisection	Samples midpoints of the hyper-rectangles	Relaxed criteria of POH selection. At each iteration, the algorithm selects and divides the best hyper-rectangles of each size (δ_i^k).			No input parameters
DIRECT-G			Uses enhanced global selection (Definition 2 in [78]).			
DIRECT-L			Uses enhanced local selection (Definition 3 in [78]).			
DIRECT-GL			Uses the unique union of two sets obtained using enhanced global and local selection (Definitions 2 and 3 [78]).			
DIRECT			Uses Definition 1			Balance parameter ϵ
DIRECT-restart					Uses two different ε values during the selection: 0 if there is an improvement in the solution, and 0.01 otherwise.	
DIRECT-m					Uses the median value $\hat{f}_{median}$ in (5).	
DIRECT-l					Uses the infinity norm in (4) and selects at most one POH from each group having the same measure (δ_i^k)	
DIRECT-rev	Hyper-rectangular partitions based on 1-dimensional trisection				Selects at the most one POH from each group of (δ_i^k). Additional minimization procedure fmincon is employed.	
DIRECT-a	Hyper-rectangular partitions based on n-dimensional trisection				Uses the average value $\hat{f}_{average}$ in Eq. (6).	
DIRMIN					fmincon is performed from each selected POH.	
PLOR					The set of POH is reduced to just two: with the maximal ($\delta_{\max}^k$) and the minimal ($\delta_{\min}^k$) measures.	
glbSolve				Uses the function *conhull* to return all points on the convex hull, even redundant ones. *conhull* is based on the extended version of the GRAHAMSHULL [66] algorithm.		
glbSolve-sym					Discards unnecessary hyper-rectangles for symmetric functions.	
glbSolve-sym2						
MrDIRECT					Performs the selection of POH on three different sets ("levels"). MrDIRECT$_{075}$ uses different ε values at each level.	
MrDIRECT$_{075}$						
Gb-glbSolve					Uses an adaptive scheme for balancing the local and global search. Local minimization procedure fmincon is embedded into the BIRMIN algorithm.	
Gb-BIRECT	Hyper-rectangular partitions based on 1-dimensional bisection	Samples hyper-rectangle at two points lying on diagonals				
BIRMIN						
BIRECT						
ADC		Sample hyper-rectangle at two vertices			Uses an adaptive scheme for balancing the local and global search.	
DISIMPL-C	Simplicial partitions based on n-dimensional trisection	Samples midpoints of the simplices				
GB-DISIMPL-C					Uses an adaptive scheme for balancing the local and global search.	
GB-DISIMPL-V		Samples at vertices of the simplices				
DISIMPL-V						

Table 2: Summary of the main algorithmic characteristics of DIRECT-type methods for box constrained global optimization

and the objective value at an infeasible center point. The primary purpose of the parameter Δ is to forbid the convergence of the algorithm to infeasible regions by penalizing the objective value obtained at infeasible points. In such a way, the formulation (12) does not require any penalty parameters and determines the convergence of the algorithm to a feasible solution. The value of $\xi(\mathbf{x}, \hat{f}^{\text{feas}})$ is updated when a smaller value of $\hat{f}^{\text{feas}}$ is found. This way, the new DIRECT-GLce algorithm divides more hyper-rectangles with center points lying close to the boundaries of the feasible region, i.e., the potential solution.

The proposed DIRECT-GLce algorithm has two extensions: The first one is DIRECT-GLc (see Table 1), which is a simplified version of DIRECT-GLce and instead of (12) minimizing the transformed problem:

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) + \xi(\mathbf{x}, \hat{f}^{\text{feas}}),$$
$$\xi(\mathbf{x}, \hat{f}^{\text{feas}}) = \begin{cases} 0, & \mathbf{x} \in D^{\text{feas}}_{\varepsilon_\varphi} \\ \varphi(\mathbf{x}) + \Delta, & \text{otherwise}, \end{cases} \tag{14}$$

Experimental investigation in [79] showed that the algorithm has the most wins among the algorithms used in the comparison, and it can solve about 50% of the problems with the highest efficiency. Unfortunately, DIRECT-GLc algorithm's efficiency decreasing, solving more challenging problems (with nonlinear constraints and $n \geq 4$), where DIRECT-GLce is significantly better, so DIRECT-GLc should be used only for simpler optimization problems (with linear constraints and $n \leq 4$). The second extension of the DIRECT-GLce algorithm is DIRECT-GLce-min, where the algorithm is incorporated with MATLAB optimization solver fmincon. In [79] we observed, that by embedding a local minimization procedure into DIRECT-GLce-min (see Table 1) significantly reduces the total number of function evaluations compared to DIRECT-GLce and can significantly improve the quality of the final solution.

2.2.1. DIRECT-*type algorithms for linearly constrained global optimization*

Let us note that all previously described algorithms for a generally constrained problem can be directly applied to solve linearly constrained problems. This section emphasizes only applicable to problems with linear constraints, but not general ones.

In [61], we have extended the original simplicial partitioning based DISIMPL algorithm [59, 60] for such problems with linear constraints. Simplices may cover a search space defined by linear constraints. Therefore a simplicial approach may tackle such linear constraints in a very subtle way. In such a way, the new algorithms (Lc-DISIMPL-C and Lc-DISIMPL-V) [61] perform the search only in the feasible region, in contrast to other DIRECT-type approaches. Nevertheless, the authors in [61] showed that the feasible region's calculation requires solving $2n + m$ linear n-dimensional systems, and such operation is exponential in complexity. Therefore, the proposed algorithm can be effectively used for relatively small n and m values.

2.3. DIRECT-type algorithms for problems with hidden constraints

In this subsection, we consider the solution to the constrained global optimization problem:

$$\min_{\mathbf{x}\in D^{\text{feas}}} \quad f(\mathbf{x}), \tag{15}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ denotes an extended real-valued, most likely "black-box" objective function. A priori an unknown feasible region D^{feas} is defined as a non-empty set

$$D^{\text{feas}} = D \cap D^{\text{hidden}} \neq \emptyset,$$

and D^{hidden} are not given by explicit formulae hidden constraints. Such a problem formulation leads to a complex and analytically undefined feasible region.

Problems with hidden constraints often occur when the objective function is not everywhere defined [9]. Such situations arise, e.g., when computations within the objective function fail [7, 10, 14, 75], or when the evaluation of the objective function itself tests for feasibility conditions that are not given by closed-form inequalities [8, 39]. In the same vein as in [9], we call these internal to f constraints as "hidden constraints," and assume that f fails to return a value when evaluated at $\mathbf{x} \notin D^{\text{feas}}$.

One of the first proposed modifications for such problems was the barrier method (DIRECT-Barrier) [22]. The DIRECT-Barrier algorithm is relatively straightforward and assigns a predefined high value to infeasible hyper-rectangles. However, such an approach produces other well-known problems discussed and reviewed by a few authors [19, 60, 79]. The main issue is that the barrier approach makes exploration around the edges of feasibility very slow. Significant penalties used by the barrier method ensure that no infeasible hyper-rectangle can be potentially optimal as long as there is the same measure hyper-rectangle with the feasible center midpoint. For DIRECT-Barrier, the priority is the examination of regions where feasible points are found already. Another critical issue concluded in [19] is that hyper-rectangles, even with the sizeable feasible region, will not be explored in a reasonable number of function evaluations. To sum up, the barrier approach is not the best fit for the problem (15).

The second DIRECT-type approach for hidden constraints is based on Neighbourhood Assignment Strategy (NAS) [22]. DIRECT-NAS's main idea is to assign the value at infeasible point $\mathbf{x}^{\text{inf}} \notin D^{\text{feas}}$ relative to the objective values attained in the feasible points from the neighborhood of $\mathbf{x}^{\text{inf}}$. DIRECT-NAS iterates over all infeasible midpoints by creating surrounding hyper-rectangles around them by keeping the same center points in every iteration. These hyper-rectangles are increased by doubling the length of each dimension. If inside the enlarged region exists more than one feasible center point DIRECT-NAS, assigns the smallest function value to the infeasible midpoint plus a small epsilon $f(\mathbf{x}^{\text{feas}}) + \epsilon f(\mathbf{x}^{\text{feas}})$, where $\epsilon = 10^{-6}$ was proposed to use. If inside the enlarged region does not exist any feasible points, DIRECT-NAS assigns the largest objective function value found so far $f^{\max} + \lambda$, where $\lambda = 1$ was proposed to use. Such a strategy does not allow the DIRECT-NAS algorithm to move beyond the feasible region by punishing infeasible midpoints with large values. However, the algorithm's principal concern is the slow convergence, which is caused by many additional calculations.

Another recent idea to handle hidden constraints within the DIRECT framework is to use a subdividing step for infeasible hyper-rectangles. The proposed subDIRECT-Barrier [53] incorporates techniques from the previously mentioned barrier approach. Specifically, if the center point is identified as infeasible, then subDIRECT-Barrier assigns a considerable penalty value to it. An extra subdividing step is performed only in specific iterations, during which all infeasible hyper-rectangles are identified as potentially optimal and subdivided together with others POHs. The sub-dividing step can decompose the boundaries of the hidden constraints quite efficiently. Still, subDIRECT-Barrier has several apparent drawbacks. The algorithm performance strongly depends on when (how often) the subdividing step is performed. Therefore the number of new subdivisions can grow drastically, especially for higher dimensionality problems.

The most recent version for problems with hidden constraints DIRECT-GLh [77] is based on our previous DIRECT-GL [78] algorithm. For hyper-rectangles with infeasible midpoints, DIRECT-GLh assigns a value depending on how far the center is from the current best minima $\mathbf{x}^{\min}$. Such a technique does not require any additional computation. Simultaneously, distances from the $\mathbf{x}^{\min}$ point are already known, as they are used to selecting potential optimal hyper-rectangle schemes adapted from DIRECT-GL [78]. In such a way, DIRECT-GLh does not punish infeasible hyper-rectangles with large values (as was suggested by previous proposals), which are close to the $\mathbf{x}^{\min}$ and assures a faster and more comprehensive examination of hidden regions. Moreover, this approach employs additional procedures, which use an independent phase to efficiently handle infeasible initial points (see [77] for experimental justification).

2.4. Implementation of parallel DIRECT-type algorithms

We use the MathWorks official extension to the MATLAB language – the Parallel Computing Toolbox [49]. The Parallel Computing Toolbox provides several parallel programming paradigms [48], like threads, parallel for-loops and SPMD (Single Program Multiple Data). In [81], we introduced three parallel implementations of the DIRECT-GLce algorithm within MATLAB. We concluded that the version based on MATLAB spmd is the most efficient and significantly outperforms other implementations based on parfor-loops.

Therefore, in the DGO toolbox for parallel implementation, we used the most efficient parallel scheme, based on [81]. Our implementations are based on the SPMD functionality within the Parallel Computing Toolbox, which is used to allocate the work across multiple labs in the MATLAB software environment. Each lab stores information on its main memory block and data is exchanged through the message passing over the interconnection network [49]. The master-slave paradigm is used to implement dynamic load balancing. The flowchart of the parallel algorithmic framework is illustrated in Fig. 3. One lab is the master, denoted by M, and the other labs are slaves $W_i, i = 1, \dots, \rho - 1$. The master also acts as a slave. The master lab decides which hyper-rectangles will be sampled and subdivided and how these tasks will be distributed among all available slave labs. Additionally, the master lab is responsible for stopping the algorithm. The master lab also performs load balancing by distributing the selected hyper-rectangles to the rest of the slave labs. When the slave labs $W_i, i = 1, \dots, \rho - 1$ receive tasks from the master lab, each performs the **Sampling** and **Subdivision** steps sequentially. Then finds a local set of potential optimal hyper-rectangles and sends local data for the further global **Selection** step back to the master lab and becomes idle until further instructions are received. Suppose any of the termination conditions are satisfied. In that case, all slave labs receive the notification that the master lab has become inactive, and the slave labs will terminate themselves without further messaging. We refer to [81] for a more detailed description and analysis of parallel schemes.

3. DGO toolbox

The sequential and parallel implementation of DIRECT-type algorithms presented in the previous sections forms the basis for our DGO toolbox. The toolbox consists of two main parts:

- **DGO.mltbx** - MATLAB toolbox package containing implementations of DIRECT-type algorithms (from Table 1), including an extensive DIRECTlib library of the box and generally constrained test and practical engineering global optimization problems, often used for benchmarking DIRECT-type algorithms.
- **DGO.mlappinstall** - A single MATLAB app installer file containing everything necessary to install and run the DGO toolbox, including a graphical user interface (GUI), a convenient way to use it.

3.1. Graphical user interface

After installation (using **DGO.mlappinstall**), DGO can be launched from MATLAB **APPS**, located in the toolbar. The graphical interface of the main DGO toolbox window is shown in Fig. 4. Application is divided into three main parts: i) selection of the type of test problem from the

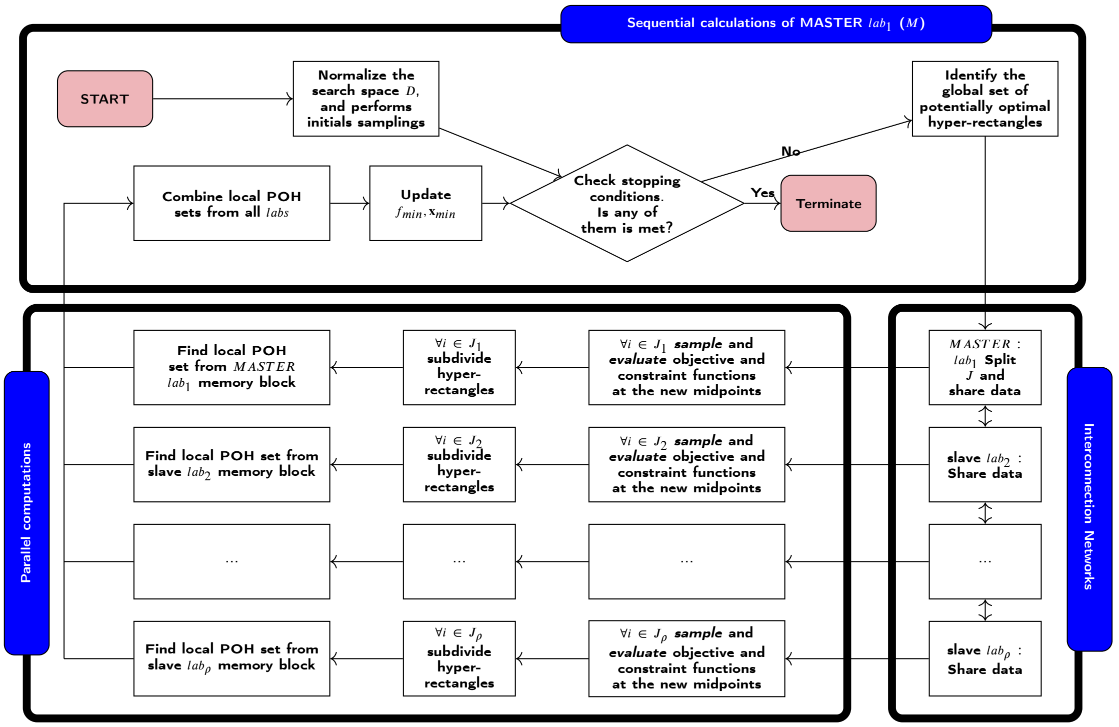

Sequential calculations of MASTER lab_1 (M)
START
Normalize the search space D, and performs initials samplings
Check stopping conditions. Is any of them is met?
No
Yes
Terminate
Identify the global set of potentially optimal hyper-rectangles
Combine local POH sets from all $labs$
Update f_{min}, x_{min}
Parallel computations
Find local POH set from $MASTER$ lab_1 memory block
$\forall i \in J_1$ subdivide hyper-rectangles
$\forall i \in J_1$ sample and evaluate objective and constraint functions at the new midpoints
Find local POH set from slave lab_2 memory block
$\forall i \in J_2$ subdivide hyper-rectangles
$\forall i \in J_2$ sample and evaluate objective and constraint functions at the new midpoints
Find local POH set from slave lab_p memory block
$\forall i \in J_p$ subdivide hyper-rectangles
$\forall i \in J_p$ sample and evaluate objective and constraint functions at the new midpoints
$MASTER$: lab_1 Split J and share data
slave lab_2 : Share data
slave lab_p : Share data
Interconnection Networks

Figure 3: Flowchart diagram for the parallel implementations of selected DIRECT-type algorithms.

DIRECTlib; ii) optimization problem setup and algorithmic options; iii) quote of the DIRECT-type algorithm and the type of implementation.

The first step is to specify the objective and constraint functions, loading them from the integrated DIRECTlib library or selecting them from the other problem. Examples of the structure needed are present. All test problems from the DIRECTlib library have up to three key features: known globally optimal solutions, a complete description of the problem including objective and constraint functions (if any), and problem visualization for two-dimensional problems. In total, DIRECTlib contains 119 test problems: 45 are bound-constrained, 35 linearly constrained, 39 with nonlinear constraints, and 11 are practical engineering problems. After selecting the optimization problem, the second step is to set up the bound constraints for each variable. At the first step, the user needs to specify the algorithm and the type of implementation. Two implementations are based on different data structures (static and dynamic), and the third is a parallel version of the algorithm. For simplicity, some toolbox options are set to default values and not displayed in the GUI but can be changed in the toolbox settings. After the termination (when any of the stopping conditions are met), the **Results** part displays the final solution and performance metrics. Additionally, the convergence process is shown in the **Convergence status** part.

3.2. MATLAB toolbox

After installation of the MATLAB toolbox (using **DGO.mltbx**), all implemented DIRECT-type algorithms and test problems can be freely accessed in the command window of MATLAB. Unlike using GUI, algorithms from the command line require more programming knowledge, and configurations must be done manually. All algorithms can be run using the same style and syntax:

```
[f_min, x_min, history] = algorithm(P, D, OPTS);
```

The left side of the equation specifies the output parameters. After the termination, the algorithm returns the best objective value (f_min), solution point (x_min), and history of the algorithmic performance during all iterations (history). The information presented here is: the iteration number, the total number of objective function evaluations, the current minimal value, and the total execution time.

On the right side, the algorithm name (algorithm) and at least two input parameters are needed to specify. The first one is the problem structure (P) consisting of an objective function:

```
P.f = 'objfun';
```

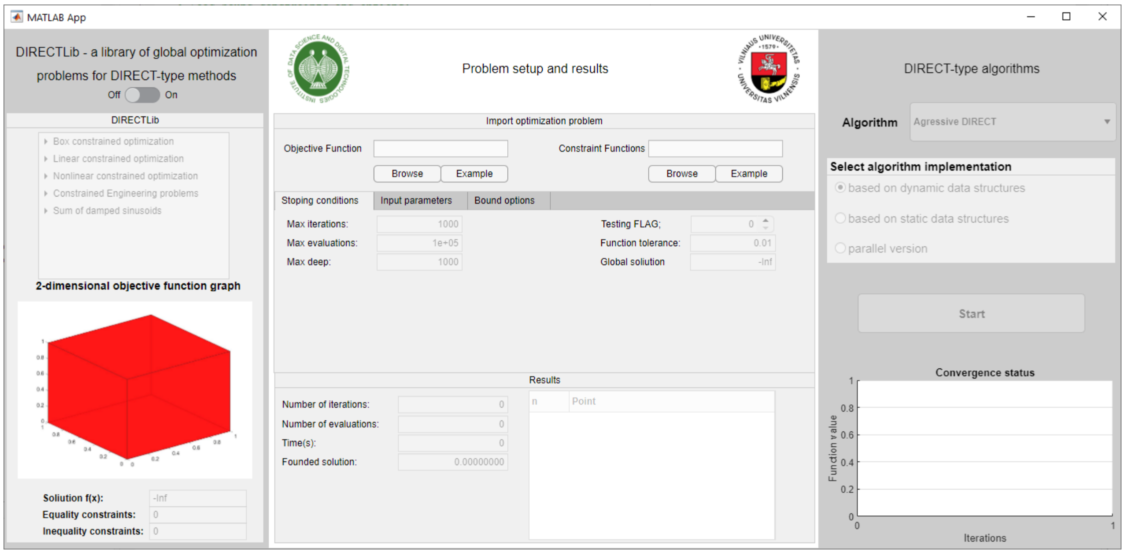
If the problem involves additional constraints, they also must be specified:

```
P.constraint = 'confun';
```

The second necessary parameter is the bound constraints for each dimension/variable:

```
D (i,1) ≤ x_i ≤ D (i,2), i = 1...n;
```

The last parameter is an optional variable (OPTS) to customize the default settings.


MATLAB App
DIRECTLib - a library of global optimization problems for DIRECT-type methods
Off On
DIRECTLib
Box constrained optimization
Linear constrained optimization
Nonlinear constrained optimization
Constrained Engineering problems
Sum of damped sinusoids
2-dimensional objective function graph
Problem setup and results
Import optimization problem
Objective Function
Constraint Functions
Browse
Example
Stoping conditions
Input parameters
Bound options
Results
n
Point
DIRECT-type algorithms
Select algorithm implementation
based on dynamic data structures
based on static data structures
parallel version
Start
Convergence status
Function value
Iterations

Soliution f(x): -Inf
Equality constraints: 0
Inequality constraints: 0

Max iterations: 1000
Max evaluations: 1e+05
Max deep: 1000
Testing FLAG; 0
Function tolerance: 0.01
Global soliution -Inf

Number of iterations: 0
Number of evaluations: 0
Time(s): 0
Founded solution: 0.00000000

Algorithm: Agressive DIRECT
Select algorithm implementation: [✓ based on dynamic data structures]

Figure 4: The graphical user interface (GUI) of DGO toolbox.

4. Experimental investigation of DIRECT-type algorithms in the DGO toolbox

This section presents the performance evaluation of DIRECT-type algorithms on test and engineering design problems from the DIRECTlib [80] library. DIRECTlib consists of the box and generally constrained test and practical engineering global optimization problems for various DIRECT-type algorithms benchmarking. We distinguish the following classes (types) of global optimization problems:

- **BC** – **B**ox-**C**onstrained problems;
- **LC** – **L**inearly-**C**onstrained problems;
- **GC** – **G**enerally-**C**onstrained problems.

A summary of all optimization problems in DIRECTlib and their properties is given in Appendix A, Table 12. We note that some test problems have several variants, e.g., *Bohachevsky*, *Shekel*, and some of them like, *Alpine*, *Csendes*, *Griewank*, can be used by changing the dimensionality of the problem. We used the following dimensions in our experimental setting: $n = 2, 5, 10, 15$.

All computations were carried out on a 6-core computer with 8th Generation Intel R Core^{TM} i7-8750H @ 2.20GHz Processor, 16 GB of RAM, and MATLAB R2020b. Performance analysis was carried out using physical cores only and disabled hyper-threading.

All global minima f^* are known for all test problems. Therefore, the investigated algorithms were stopped when it was generated such the point $\mathbf{x}$ with whom the percent error

$$pe = 100\% \times \begin{cases} \frac{f(\mathbf{x})-f^*}{|f^*|}, & f^* \neq 0, \\ f(\mathbf{x}), & f^* = 0, \end{cases} \tag{16}$$

is smaller than the tolerance value ε_{pe}, i.e., $pe \leq \varepsilon_{\text{pe}}$. Additionally, we stopped the tested algorithms when the number of function evaluations exceeded the prescribed maximal limit (equal to 2×10^6) or took more than twelve hours. In any of these situations, the final result is set to 2×10^6 to further process results. Two different values for ε_{pe} were considered: 10^{-2}, 10^{-8}. All algorithms were tested using the $\varepsilon_{\text{pe}} = 10^{-2}$ value. Moreover, algorithms containing additional schemes to speed up the solution's refinement have been tested using the $\varepsilon_{\text{pe}} = 10^{-8}$ value.

Additionally, we analyze and compare the algorithms' performance by applying the performance profiles [17] to the convergence test (16). Performance profiles are broadly used for benchmarking and evaluating different solvers' performance when running on a large set of problems. Benchmark results are generated by running a specific algorithm υ (from a group of algorithms $\mathcal{V}$ under consideration) for each problem p from a benchmark set $\mathcal{P}$ and recording the performance measure of interest. Such measures could be, e.g., the number of function evaluations, the computation time, the number of iterations, or the memory used. In our case, we used a number of function evaluations and the execution (computation) time criteria.

Performance profiles asses the overall performance of solvers using a performance ratio ($\lambda_{p,\upsilon}$)

$$\lambda_{p,\upsilon} = \frac{t_{\upsilon,p}}{\min\{t_{\upsilon,p} : p \in \mathcal{P}\}}, \tag{17}$$

where $t_{\upsilon,p} > 0$ is the number of functions evaluations (or the execution time) required to solve problem p by the algorithm υ and $\min\{t_{\upsilon,p} : \upsilon \in \mathcal{V}\}$ is the smallest number of function evaluations (or the lowest execution time) by an algorithm on this problem. Then, the performance profile ($\chi_\upsilon(\beta)$) of an algorithm $\upsilon \in \mathcal{V}$ is given by the cumulative distribution

Algorithm	Avg. # local searches	Failed	Average results			Average results ($n \leq 4$)			Average results ($n \geq 5$)			Median results		
			$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$
DIRECT	–	18/81	502, 101	263.29	1, 586	143, 290	215.98	2, 549	739, 073	299.67	989	13, 285	0.78	52
DIRECT-restart	–	23/81	606, 569	5, 518.48	157	259, 643	2, 989.24	83	838, 163	7, 231.22	203	12, 861	1.02	48
DIRECT-m	–	28/81	771, 018	334.42	2, 380	119, 777	371.78	2, 463	1, 198, 363	292.92	2, 328	32, 859	3.20	136
DIRECT-l	–	23/81	609, 628	4, 281.41	64, 431	130, 252	921.76	21, 834	924, 655	6, 494.21	90, 841	6, 903	15.44	454
DIRECT-rev	3	8/81	217, 082	472.69	3, 751	64, 954	423.54	3, 842	317, 752	513.42	3, 694	597	0.12	13
DIRECT-a	–	41/81	1, 020, 023	830.73	4, 648	263, 114	667.48	4, 264	1, 519, 178	910.06	4, 886	2, 000, 000	285.42	181
DIRMIN	745	6/81	179, 687	23.61	71	65, 011	22.03	89	255, 902	25.11	60	366	0.06	2
PLOR	–	29/81	726, 480	2, 458.05	54, 604	266, 980	2, 092.23	37, 747	991, 193	2, 628.55	65, 055	2, 981	0.71	359
glbSolve	–	27/81	699, 262	472.42	2, 866	130, 403	324.49	2, 540	1, 073, 122	550.10	3, 068	13, 951	0.94	52
glbSolve-sym	–	34/81	875, 499	3, 607.52	9, 420	452, 682	1, 246.74	10, 092	1, 158, 529	5, 057.67	9, 003	81, 777	400.90	251
glbSolve-sym2	–	38/81	943, 034	6, 388.08	8, 802	651, 078	1, 766.70	10, 591	1, 144, 618	9, 327.52	7, 693	199, 221	889.46	458
MrDIRECT	–	15/81	433, 889	111.58	1, 672	70, 799	8.76	362	677, 405	180.30	2, 500	8, 237	0.45	71
MrDIRECT$_{075}$	–	14/81	428, 846	206.14	3, 500	67, 508	18.10	504	666, 181	329.29	5, 357	6, 055	0.49	93
BIRECT	–	10/81	279, 471	1, 919.75	6, 830	65, 938	884.88	2, 643	420, 224	2, 309.96	9, 427	2, 370	1.28	72
GB-DISIMPL-C	–	46/81	1, 156, 420	3, 906.25	11, 888	234, 005	1, 887.82	15, 642	1, 763, 557	5, 103.07	9, 561	2, 000, 000	138.44	25
GB-DISIMPL-V	–	36/81	898, 337	19, 858.76	2, 110	74, 547	2, 111.40	1, 884	1, 437, 319	30, 614.41	2, 249	66, 257	4, 298.97	40
Gb-BIRECT	–	12/81	344, 351	2, 406.69	19, 318	68, 649	764.49	7, 685	525, 760	2, 810.85	26, 529	4, 994	1.56	139
BIRMIN	1	5/81	125, 583	2, 581.75	10, 659	64, 826	1, 387.25	6, 415	166, 578	3, 390.13	13, 290	338	0.14	22
Gb-glbSolve	–	26/81	688, 955	653.24	7, 889	132, 889	1, 024.89	15, 002	1, 054, 383	414.98	3, 479	22, 541	2.39	69
DISIMPL-C	–	46/81	1, 149, 469	4, 257.54	11, 953	219, 423	1, 917.73	13, 213	1, 720, 508	5, 720.91	11, 171	2, 000, 000	126.36	23
DISIMPL-V	–	34/81	844, 074	18, 265.14	710	67, 151	1, 487.40	422	1, 352, 377	28, 374.84	888	21, 828	667.34	26
ADC	–	32/81	857, 396	4, 630.86	20, 450	95, 110	1, 449.71	8, 710	1, 357, 152	6, 314.59	27, 728	53, 375	27.58	445
Aggressive DIRECT	–	14/81	475, 259	20.89	96	167, 352	11.06	87	678, 386	27.55	101	65, 253	2.14	45
DIRECT-G	–	10/81	310, 712	45.35	355	82, 249	15.68	301	461, 360	65.04	389	11, 103	0.45	52
DIRECT-L	–	16/81	430, 980	115.71	623	66, 776	38.69	352	670, 089	167.87	791	10, 209	0.52	51
DIRECT-GL	–	3/81	152, 508	11.73	100	9, 273	0.66	47	246, 098	18.96	133	7, 737	0.32	37

Table 3: The performance of algorithms from DGO using the number of function evaluations ($f_{eval.}$), the total execution time in seconds ($time$), and the total number of iterations ($iter.$) criteria on a set of box-constrained problems (from DIRECTlib) when $\varepsilon_{\text{pe}} = 10^{-2}$ value in stopping condition (16) was used.

function for the performance ratio

$$\chi_v(\beta) = \frac{1}{|\mathcal{P}|}\text{size}\{p \in \mathcal{P} : \lambda_{p,v} \leq \beta\}, \quad \beta \geq 1, \quad (18)$$

where $|\mathcal{P}|$ is the cardinality of $\mathcal{P}$. Thus, $\chi_v(\beta)$ is the probability for an algorithm $v \in \mathcal{V}$ that a performance ratio $\lambda_{p,v}$ for each $p \in \mathcal{P}$ is within a factor β of the best possible ratio.

The performance profiles seek to capture how well the certain algorithm v performs compared to other algorithms from $\mathcal{V}$ on the set of problems from $\mathcal{P}$. In particular, $\chi_v(1)$ gives the fraction of the problems in $\mathcal{P}$ for which algorithm v is the winner, i.e., the best according to the $\lambda_{p,v}$ criterion. In general, algorithms with high values for $\chi_v(\beta)$ are preferable [17, 79].

4.1. Comparison of DIRECT-type algorithms for box constrained optimization

In Table 3, we summarize experimental results when $\varepsilon_{\text{pe}} = 10^{-2}$ was used. The smallest number of unsolved problems is achieved using DIRECT-GL (3/81), while the second-best algorithm is BIRMIN (5/81), a hybrid version and enriched with the local search subroutine fmincon. In column '**Avg. # local searches**' we report the average number of local searches performed by each hybridized algorithm. Hybridization with fmincon allows solving more problems compared to, e.g., the globally biased version Gb-BIRECT (12/81). Among traditional DIRECT-type algorithms, the second-best position share BIRECT and DIRECT-G. Both algorithms failed to solve (10/81) problems.

Furthermore, hybridization allows significantly reducing the total number of function evaluations (see **Average results** and **Median results** columns). Among the traditional algorithms, the BIRECT and PLOR were the most effective algorithms based on the median number of function evaluations (see **Median results** column). However, for PLOR, such performance needs to be interpreted correctly. As PLOR restricts POH set to only two hyper-rectangles per iteration, a lower number of function evaluations are required to get closer to the solution for simpler (low-dimensional) problems. However, looking at the average number of function evaluations, even restricted to the simplest subset of problems (see **Average results** ($n \leq 4$)), PLOR performance is among the worst. PLOR has failed on a larger number of simpler test problems compared to other approaches. In contrast, DIRECT-GL is only in a ninth-place based on the median number of function evaluation criteria, but is the only algorithm that solves all simpler ($n \leq 4$) problems and is the best performing algorithm, including hybridized versions. Moreover, on average, DIRECT-GL required approximately

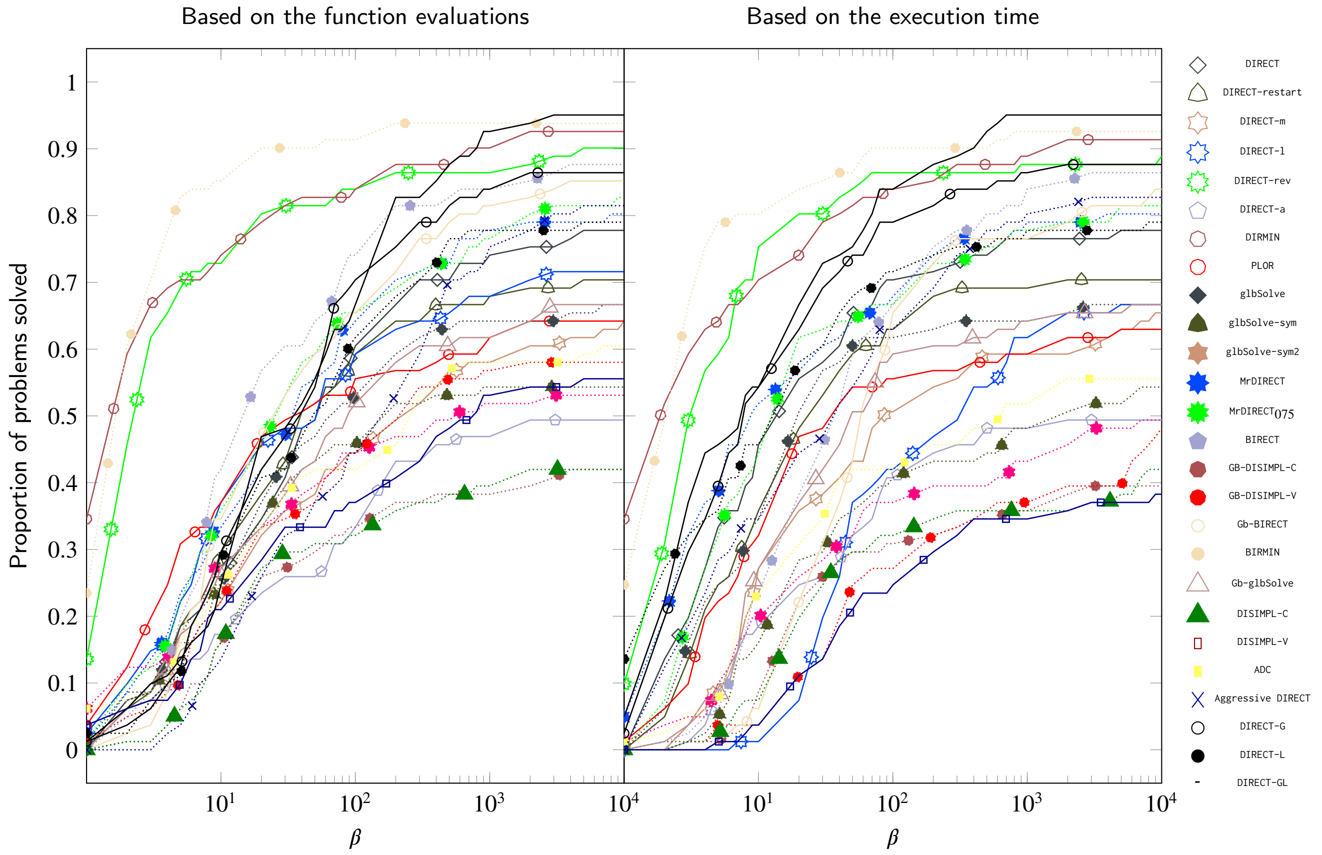


Figure 5: Performance profiles on $[1, 10000]$ of DIRECT-type algorithms from DGO toolbox on the whole set of box-constrained optimization problems from DIRECTlib when $\varepsilon_{\text{pe}} = 10^{-2}$ value in (16) was used.

84% percent fewer evaluations of the objective function than the second-best, BIRMIN algorithm, among all traditional DIRECT-type algorithms on a class of more challenging problems (see **Average results** ($n \geq 5$)). Not surprisingly, within this class, the hybridized BIRMIN algorithm delivers the best average performance. Compared with the second-best DIRECT-type algorithm, DIRECT-GL, BIRMIN requires approximately 32% percent fewer evaluations.

Looking at the execution time (*time*), DIRECT-GL and Aggressive DIRECT are the best among all traditional algorithms. Aggressive DIRECT does not have a traditional POH selection procedure. Instead, the algorithm selects at least one candidate from each group of different hyper-rectangles. Therefore, the number of selected hyper-rectangles per iteration is larger, especially for higher dimensionality test problems. Consequently, the number of iterations (*iter.*) using Aggressive DIRECT is among the smallest. In overall, DIRECT-GL showed the most promising among all tested traditional DIRECT-type algorithms, while simplicial partition-based algorithms (various DISIMPL-C variations) are not appealing for box-constrained problems.

The performance profiles on the interval $[1, 10000]$ of all algorithms are shown in Fig. 5. Again, the performance profiles confirm that the hybridized algorithms (enriched with fmincon procedure) are more efficient than traditional DIRECT-type approaches. The DIRMIN algorithm has the most wins and can solve about 35% of problems with the highest efficiency, i.e., using the minor function evaluation (see the left side of Fig. 5). Nevertheless, when the performance ratio increases, the DIRECT-GL algorithm starts outperforming all algorithms, including hybridized ones. Although DIRECT-GL delivers the best overall performance (based on function evaluations), performance profiles in Fig. 5 reveal that other DIRECT-type extensions (PLOR, DIRECT-l, BIRECT, Gb-BIRECT) have more wins than DIRECT-GL. Moreover, they can solve about 60% of problems (mainly lower conditionality) using a smaller number of function evaluations. The two-step-based selection strategy in DIRECT-GL selects a more extensive set of POH. While for more straightforward problems, this is detrimental, but it helps locate a global solution faster to find it with higher accuracy. In terms of execution time (see the right-hand side in Fig. 5), DIRECT-GL is the fastest among all the traditional DIRECT-type algorithms (excluding hybridized).

As some algorithms have integrated schemes to help speed up the refinement of solutions with higher accuracy, we tested them when the solution with much higher precision ($\varepsilon_{\text{pe}} = 10^{-8}$) is sought. In Table 4, we summarize our experimental findings. First, the number of failed problems is much higher when higher accuracy is needed (see the **Failed** column in Tables 3 and 4). The smallest

Algorithm	Avg. # local searches	Failed	Average results			Average results ($n \le 4$)			Average results ($n \ge 5$)			Median results		
			$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$	$f_{eval.}$	$time$	$iter.$
DIRECT	–	41/81	1,092,242	919.10	7,843	849,599	1,435.78	17,024	1,241,572	603.04	2,151	2,000,000	271.28	227
DIRECT-restart	–	29/81	778,771	7,840.37	200	333,394	4,377.27	114	1,035,618	10,191.29	252	53,629	6.00	91
DIRECT-rev	3	13/81	355,126	1,015.25	29,551	193,942	1,183.78	69,005	464,342	929.35	5,089	844	0.15	22
DIRMIN	897	12/81	330,944	40.89	86	130,958	32.40	104	464,196	47.09	74	501	0.09	3
glbSolve	–	58/81	1,476,272	1,156.43	9,973	859,763	1,695.75	18,393	1,855,620	813.28	4,753	2,000,000	369.45	1,201
MrDIRECT	–	36/81	965,988	1,069.29	10,761	683,962	1,677.14	20,980	1,167,840	698.98	4,296	500,849	49.23	397
MrDIRECT$_{075}$	–	35/81	1,025,586	420.20	6,365	201,290	99.95	2,459	1,527,193	619.43	8,786	945,417	138.55	749
BIRMIN	4	13/81	323,769	5,113.45	18,498	131,825	2,775.34	11,067	451,804	6,697.01	23,105	370	0.14	22
DIRECT-L	–	21/81	572,673	172.35	955	133,080	84.05	681	821,653	232.45	1,126	36,489	1.73	94
DIRECT-GL	–	6/81	277,081	29.10	242	82,357	26.45	245	401,651	31.66	239	28,211	1.09	61

Table 4: The performance of selected algorithms from DGO using the number of function evaluations ($f_{eval.}$), the total execution time in seconds ($time$), and the total number of iterations ($iter.$) criteria on a set of box-constrained problems (from DIRECTlib) when $\varepsilon_{\text{pe}} = 10^{-8}$ value in stopping condition (16) was used.

number of unsolved problems is achieved using DIRECT-GL (6/81), where all failed test problems belong to ($n \ge 5$) class. The DIRECT-GL algorithm turns out to be more efficient even than hybrid methods. Overall, DIRECT-GL required approximately 16% fewer function evaluations and took 29% less time than the second-best algorithm, BIRMIN (see **Average results** column in Table 4). However, BIRMIN algorithm has the best median value (see **Median results** column), therefore solving at least half of the problems with the best performance. Finally, let us stress the inefficiency of the original DIRECT algorithm (DIRECT and glbSolve implementations). As the median value is more than 2,000,000, this means that DIRECT failed more than half of the test problems to solve.

Finally, the performance profiles on the interval [1, 10000] of the selected algorithms are shown in Fig. 6. Once again, the performance profiles confirm that the hybridized algorithms are more efficient than traditional DIRECT-type approaches. However, when the performance ratio increases, DIRECT-GL outperforms all algorithms, including BIRMIN, in function evaluations and execution time.

4.2. Comparison of DIRECT-type algorithms for constrained global optimization

The comparison presented in this section was carried out using 80 global optimization test problems with various constraints. In DIRECTlib, 35 test problems contain linear constraints, 39 problems have nonlinear constraints where 5 of them include equality constraints. All necessary details about the test problems are given in Appendix A, Table 12. We used the same stopping condition in these experimental investigations as in the previous ones, and the value $\varepsilon_{\text{pe}} = 10^{-2}$.

Let us stress that 5 of the test problems contain equality constraints, which we transform into inequality constraints as follows:

$$\mathbf{h}(\mathbf{x}) = 0 \rightarrow |\mathbf{h}(\mathbf{x}) - \varepsilon_{\text{h}}| \le 0 \qquad (19)$$

where $\varepsilon_{\text{h}} > 0$ is a small tolerance for equality constraints. In our experiments, it was set to 10^{-8}.

4.2.1. *Test results on problems with hidden constraints*

In the first part, we compared DIRECT-type versions devoted to problems with hidden constraints. In this experimental part, we have used all constrained test problems. However, we assume that any information about the constraint functions is unavailable. In the experimental investigation, the hidden search area (D^{hidden}) is defined as

$$D^{\text{hidden}} = \{\mathbf{x} \in D : \mathbf{g}(\mathbf{x}) \le 0, \mathbf{h}(\mathbf{x}) = 0\} \qquad (20)$$

Still, this information is unavailable for the tested algorithms and used only to determine whether a certain point is feasible or not. Obtained experimental results are summarized in the upper part of Table 5 (see 'Performance of DIRECT-type algorithms for problems with hidden constraints'). First, let us note that 13 out of the 80 test problems contain complex constraints leading to a tiny feasible region. As the algorithms within this class do not use any information about constraint functions, none of the tested algorithms could find a single feasible point for any of these 13 test problems.

The best among all DIRECT-type algorithms for problems with hidden constraints is DIRECT-GLh (was unable to solve (18/80)), while the second-best is DIRECT-NAS (failed to solve (29/80)). The median number of function evaluations (see f_{eval} in **Median results**) is similar for both. Still, DIRECT-GLh is the best, primarily based on the number of iterations ($iter.$) and the execution $time$: DIRECT-GLh took around 4.5 times fewer iterations and approximately 33 times less execution time than the second best DIRECT-NAS, algorithm. The speed is the essential factor differentiating DIRECT-GLh from DIRECT-NAS.

For an extra subdividing step-based subDIRECT-Barrier, the user must define how often this step is activated. Unfortunately, the authors in [53] did not make any sensitivity analysis and guidance of this. In our experiments, we start the subdividing step at $sub^k, k = 1, 2, \dots$ iterations. We tested three different values for the variable sub, i.e., $sub = 2, 3$ and 5. Our

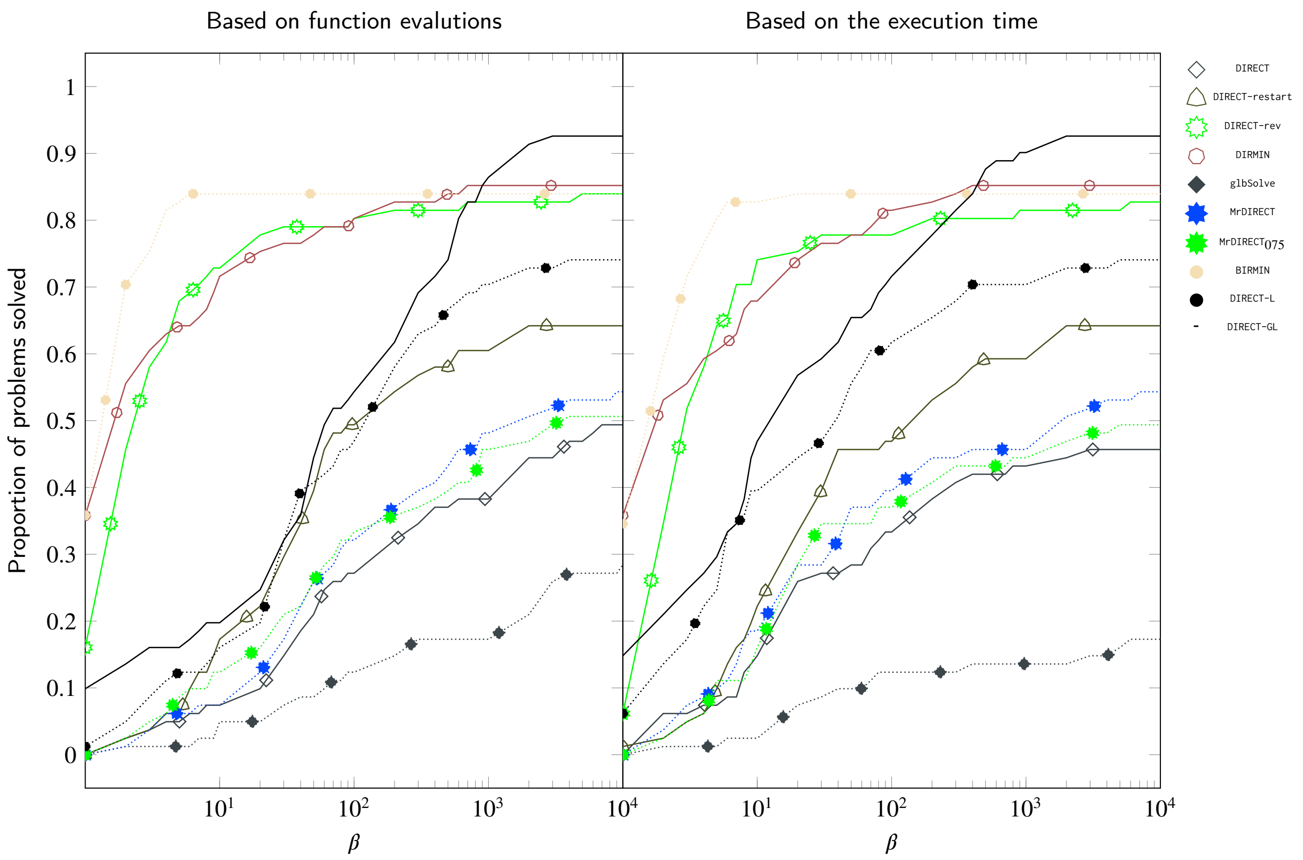


Figure 6: Performance profiles on $[1, 10000]$ of selected DIRECT-type algorithms from DGO toolbox on the whole set of box-constrained optimization problems from DIRECTlib when $\varepsilon_{\text{pe}} = 10^{-8}$ value in (16) was used.

Algorithm	**Parameter**	**Failed**	**Average results**			**Average results (Nonlin. constr.)**			**Average results (Lin. constr.)**			**Median results**		
			$f_{eval.}$	*time*	*iter.*	$f_{eval.}$	*time*	*iter.*	$f_{eval.}$	*time*	*iter.*	$f_{eval.}$	*time*	*iter.*
Performance of DIRECT-type algorithms for problems with hidden constraints														
DIRECT-NAS	–	29/80	711,720	15,372.02	20,049	940,617	20,404.43	30,224	445,705	9,523.55	8,223	9,124	18.68	144
DIRECT-Barrier	–	46/80	1,192,521	2,908.88	38,069	1,226,115	3,699.74	50,092	1,153,479	1,989.78	24,096	2,000,000	723.03	2596
subDIRECT-Barrier	$sub = 2$	53/80	1,364,353	860.50	6,896	1,330,214	923.82	6,319	1,404,028	786.92	7,565	2,000,000	295.60	65
subDIRECT-Barrier	$sub = 3$	48/80	1,240,771	773.74	3,623	1,274,469	799.58	875	1,201,609	743.71	6,816	2,000,000	286.48	251
subDIRECT-Barrier	$sub = 5$	47/80	1,229,037	1,359.66	11,199	1,337,592	1,514.72	10,276	1,102,878	1,179.45	12,272	2,000,000	559.34	1,294
DIRECT-GLh	–	18/80	470,807	312.40	104	631,983	473.22	96	283,495	125.51	114	7,068	0.57	32
Performance of DIRECT-type algorithms for generally constrained optimization problems														
DIRECT-GLc	–	11/80	330,659	67.46	352	492,568	110.70	554	142,494	17.21	117	3,759	0.34	37
DIRECT-GLce	–	7/80	258,462	40.02	270	388,986	64.12	373	106,772	12.01	149	9,768	0.86	75
DIRECT-GLce-min	–	1/80	37,495	6.18	35	64,003	10.91	52	6,689	0.68	14	123	0.06	5
DIRECT-L1	$\gamma = 10$	43/80	1,087,528	228.17	1,688	1,216,346	380.03	2,736	937,821	51.69	471	2,000,000	0.19	49
DIRECT-L1	$\gamma = 10^2$	41/80	1,051,478	870.98	3,369	1,189,695	1,455.73	3,567	890,848	191.41	3,139	2,000,000	2.49	64
DIRECT-L1	$\gamma = 10^3$	40/80	1,042,671	1,144.35	8,203	1,148,028	1,253.35	4,589	920,230	1,017.67	12,404	1,564,860	62.27	241
Performance of DIRECT-type algorithms devoted for problems with linear constraints only														
Lc-DISIMPL-C	–	5/35	N/A	N/A	N/A	N/A	N/A	N/A	290,402	6,424.68	387	443	0.12	27
Lc-DISIMPL-V	–	3/35	N/A	N/A	N/A	N/A	N/A	N/A	171,738	3,686.81	24	16	0.01	1

N/A – not available

Table 5: The performance of DIRECT-type algorithms for problems with hidden constrains based on the number of function evaluations ($f_{eval.}$), the total execution time in seconds (*time*), and the total number of iterations (*iter.*) criteria

experience showed that an extra subdividing step combined with a traditional barrier approach based on subDIRECT-Barrier did not significantly improve performance (based on the number of **Failed** problems and the **Average results**) over the original DIRECT-Barrier. The most obvious difference is that, on average, subDIRECT-Barrier subdivides much more POH per iteration because of an extra subdividing step, leading to a smaller number of iterations (*iter.*) and the execution *time*. subDIRECT-Barrier algorithm suffers solving larger dimension and problems where D^{hidden} contains nonlinear constraints (see **Average results (Nonlin. constr.)**

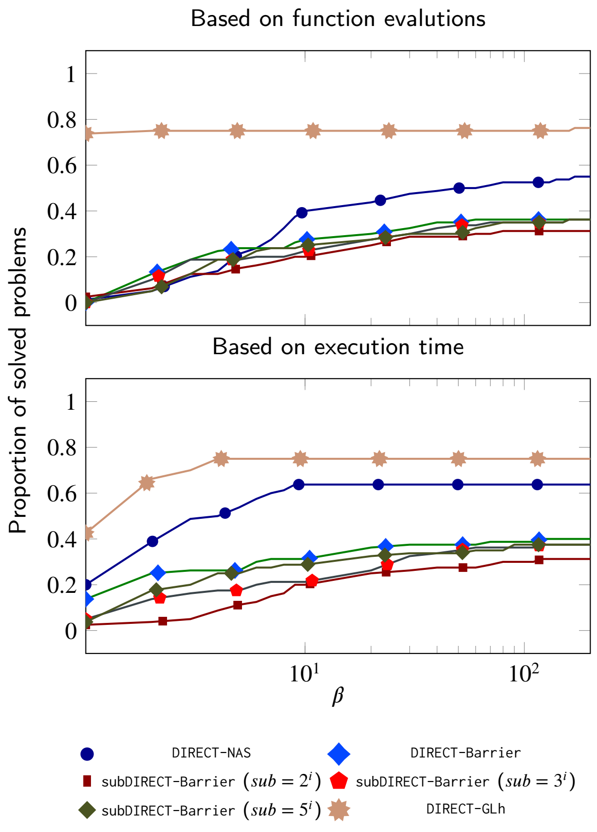


Figure 7: Performance profiles on $[1, 200]$ of DIRECT-type algorithms for problems with hidden constraints.

column). Therefore, this limits subDIRECT-Barrier applicability primarily to low-dimensional problems.

Finally, in Fig. 7, the comparative performance of algorithms using the performance profiles tool is demonstrated. They confirm that the DIRECT-GLh is the most effective optimizer in this class solves approximately 80% of the tested problems with the highest efficiency.

4.2.2. Test results on problems with general constraints

Better performance of DIRECT-type algorithms can be expected when the constraint function information is known. Let us note that all DIRECT-type algorithms considered in the previous section are included in this analysis. The new experimental results are added in the middle part of Table 5 (see ‘Performance of DIRECT-type algorithms for generally constrained optimization problems’) First, let us note that the recently proposed DIRECT-GLc and DIRECT-GLce algorithms of this class can much more successfully (compared to the algorithms for problems with hidden) solve problems contain complex and tiny feasible regions. The best average results among traditional DIRECT-type algorithms were obtained using the DIRECT-GLce (failed to solve 7/80). Interestingly, the performance based on the number of failed problems using DIRECT-GLc is worse, but based on the median results, it outperforms DIRECT-GLce quite clearly. This means that DIRECT-GLc is very effective while solving simpler problems, but the effectiveness drops when solving more complicated problems, e.g., higher dimensionality and nonlinear constraints.

While solving optimization problems with general constraints, the solution point is often located on the feasible region’s boundaries. The common problem of several DIRECT-type algorithms is that hyper-rectangles with infeasible midpoints situated closely to the edges of feasibility are penalized with large values, resulting in a low probability of being elected POH. In this way, these algorithms converge very slowly.

Among all approaches within this class, DIRECT-GLce-min is the only hybridized version. Again, incorporating the local minimization procedure into DIRECT-GLce improves the performance significantly, e.g., it reduces the overall number of function evaluations approximately 7 times. Moreover, DIRECT-GLce-min fails to solve only one test problem.

Finally, in Figs. 8 and 9, the comparative performance of DIRECT-type algorithms (except hybridized DIRECT-GLce-min) using the performance profiles tool is demonstrated. Performance profiles confirm the same trends, i.e., the overall advantage of methods incorporating constraint information versus designed explicitly for problems with hidden constraints.

4.2.3. Test results on problems with linear constraints

In the final part, we test the performance of DIRECT-type algorithms on problems with linear constraints. We consider all previously tested algorithms and two specifically designed simplicial partitions based Lc-DISIMPL-V and Lc-DISIMPL-C [61]. The main advantage of using simplices is that they can cover a feasible region defined by linear constraints. Thus any infeasible areas are not involved in the search. Moreover, for most problems from DIRECTlib, the solution is located at the intersection of linear constraints. Therefore Lc-DISIMPL-V finds it in the early or even in the first iteration. The performance profiles (see Fig. 10) reveal the overall effectiveness of the Lc-DISIMPL-V algorithm for such problems with linear constraints. Nevertheless, the efficiency of simplicial partition-based algorithms suffers from the problem dimension. Three greater dimensionality problems with linear constraints were unsolved by the Lc-DISIMPL-V algorithm (see the bottom part of Table 5, ‘Performance of DIRECT-type algorithms devoted for problems with linear constraints only’). Moreover, simplicial partition-based implementations are pretty slow. For example, on average, DIRECT-GLce is approximately 307 times faster than Lc-DISIMPL-V. Moreover, DIRECT-GLce solved all test problems with linear constraints.

5. DGO performance on engineering problems

In this section, the algorithms from DGO were tested on eleven engineering design problems: tension/compression spring, three-bar truss, NASA speed reducer, pressure vessel, welded beam, and six different versions of the

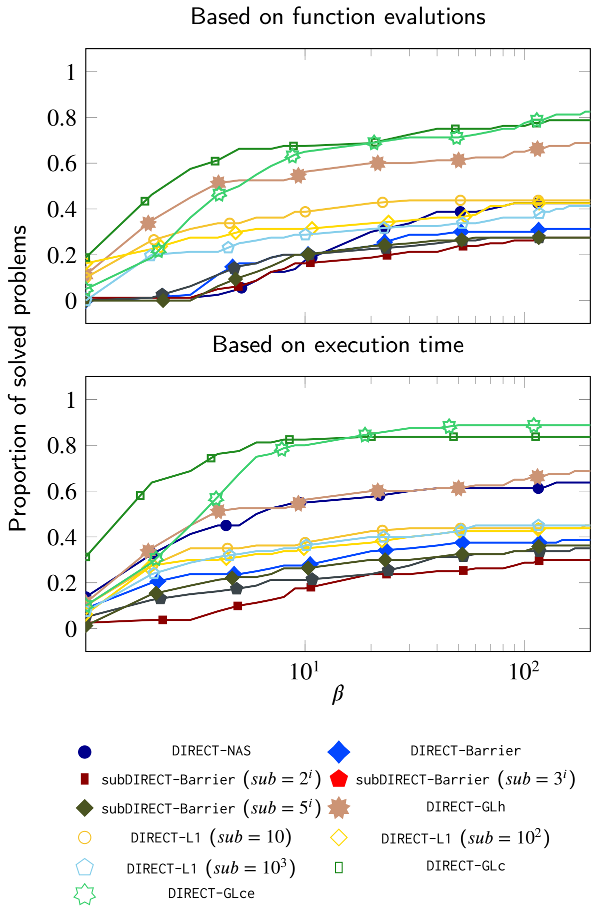


Figure 8: Performance profiles on $[1, 200]$ of DIRECT-type algorithms for problems with general and hidden constraints on the whole set of constrained optimization test problems.

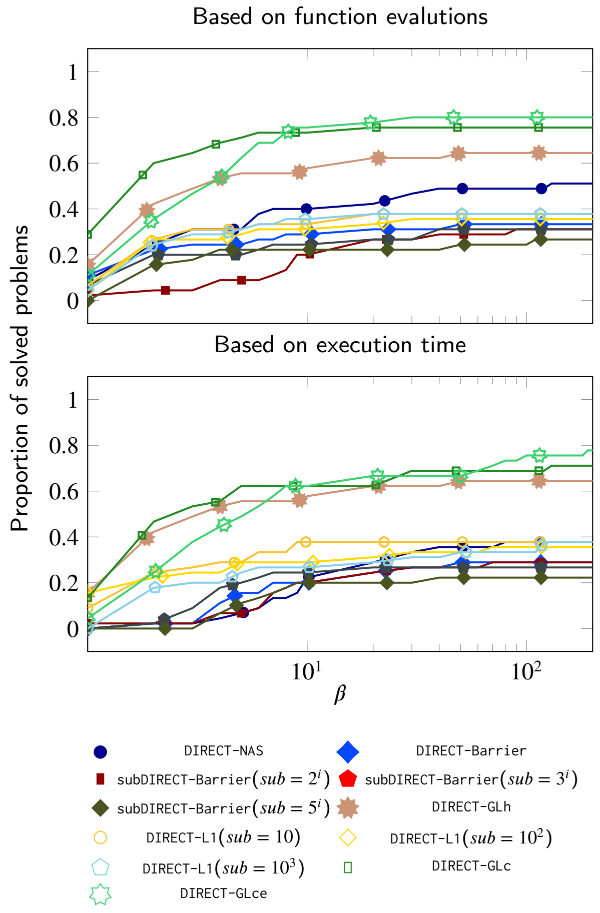


Figure 9: Performance profiles on $[1, 200]$ of DIRECT-type algorithms for problems with general and hidden constraints on the subset of constrained optimization test problems containing nonlinear constraints.

general nonlinear regression problem. The general nonlinear regression problem is box-constrained, while the others involve different types of constraints. Only the most promising algorithms (based on the results from Section 4) were considered. As the global minimums are known for all these engineering problems, we used the same stopping rule in the previous experiments. A detailed description of all engineering problems and mathematical formulations is given in Appendix B.

5.1. Tension/compression spring design problem

Here, we consider the tension-compression string design problem. This problem aims to minimize the string weight under the constraints on deflection, shear stress, surge frequency, and limits on the outside diameter. A detailed description of the practical problem can be found in [36], while in Appendix B.1 we give a short description and mathematical formulation.

A comparison of found solutions and performance metrics by the most promising algorithms from DGO is shown in Table 6. Four considered algorithms were able to solve this problem. Based on the number of function evaluation criteria, a slightly unexpectedly DIRECT-NAS was significantly better than other algorithms. The surprise comes from the fact that the algorithm does not use any information about the constraint functions. However, the DIRECT-NAS algorithm was approximate 4 times slower than the second-best method DIRECT-GLce.

Algorithm (input)	Iterations	f_{eval}	Time (s)	f_{min}
DIRECT-NAS	297	$17,659$	77.46	0.012680
DIRECT-Barrier	$40,478$	$2,000,000$	$2,465.81$	0.012867
subDIRECT-Barrier $(sub = 5)$	$15,626$	$2,000,000$	722.79	0.012867
DIRECT-GLh	700	$2,000,000$	214.81	0.012683
DIRECT-GLc	675	$423,209$	47.53	0.012680
DIRECT-GLce	624	$178,115$	20.45	0.012680
DIRECT-GLce-min	624	$178,115$	20.70	0.012680
DIRECT-L1 $(p = 10^3)$	$33,571$	$2,000,000$	$3,041.00$	0.012867

Table 6: Performance of the most promising DIRECT-type algorithms from DGO on a tension/compression design problem

5.2. Three-bar truss design problem

Here, we consider the three-bar truss design problem. The goal is to minimize the volume subject to stress

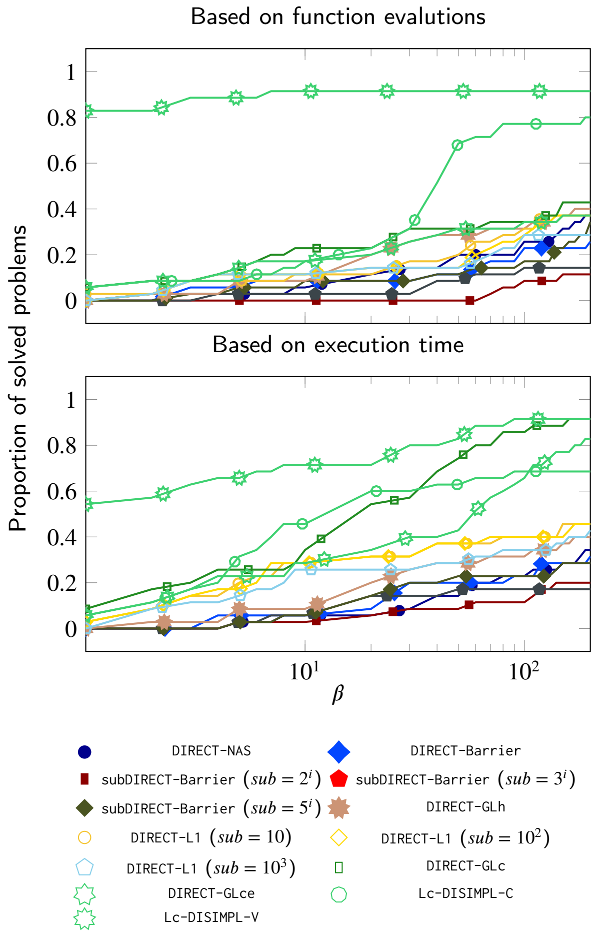


Figure 10: Performance profiles on $[1, 200]$ of DIRECT-type algorithms for problems with general and hidden constraints on the subset of constrained optimization test problems containing linear constraints.

constraints. The detailed description of the problem is given in [67], while in Appendix B.2, we provide a brief description and mathematical formulation.

A comparison of found solutions and performance metrics is shown in Table 7. Here, hybridized DIRECT-GLce-min was the most efficient optimizer. However, none of the algorithms had any difficulty solving this problem, and they found the solution in less than one-second time.

Algorithm (input)	Iterations	f_{eval}	Time (s)	f_{min}
DIRECT-NAS	29	339	0.05	263.915790
DIRECT-Barrier	13	125	0.02	263.915790
subDIRECT-Barrier ($sub = 5$)	14	161	0.02	263.915790
DIRECT-GLh	12	231	0.03	263.915790
DIRECT-GLc	17	727	0.08	263.911750
DIRECT-GLce	33	1,055	0.13	263.915790
DIRECT-GLce-min	6	93	0.03	263.895850
DIRECT-L1 ($p = 10^3$)	17	179	0.03	263.915790

Table 7: Performance of the most promising DIRECT-type algorithms from DGO on a three-bar truss design problem

5.3. NASA speed reducer design problem

Here we consider the NASA speed reducer design problem. The goal is to minimize the overall weight subject to constraints on the gear teeth' bending stress, surface stress, transverse deflection of the shaft, and stresses in the shafts. A detailed description of the problem can be found in [67], while in Appendix B.3, we provide a short description and mathematical formulation.

A comparison of the found solutions and performance metrics is shown in Table 8. Only two algorithms (DIRECT-GLce and DIRECT-GLce-min) were able to tackle this problem. Again, the hybridized DIRECT-GLce-min algorithm showed the best performance. Note that the found solution with DIRECT-L1 is better than the best know value f_{min}. However, the reported solution point is outside the feasible region and violates a few constraints.

Algorithm (input)	Iterations	f_{eval}	Time (s)	f_{min}
DIRECT-NAS	6,719	325,691	43,145.14	3006.874789
DIRECT-Barrier	32,031	2,000,000	2,625.88	3006.838136
subDIRECT-Barrier ($sub = 5$)	11,112	2,000,000	1,234.33	3006.838136
DIRECT-GLh	528	2,000,000	169.74	3003.135167
DIRECT-GLc	1623	2,000,000	373.40	3002.869474
DIRECT-GLce	254	123,175	9.70	2996.572800
DIRECT-GLce-min	55	10,229	0.93	2996.348212
DIRECT-L1 ($p = 10^3$)	661	26,667	3.01	2995.382568[a]

a – result is outside the feasible region

Table 8: Performance of the most promising DIRECT-type algorithms from DGO on a NASA speed reducer design problem

5.4. Pressure vessel design problem

In this subsection, we consider a pressure vessel design problem, and the goal is to minimize the total cost of the material, forming, and and to weld a cylindrical vessel. A detailed description of the problem can be found in [36], while in Appendix B.4, we provide a short description and mathematical formulation.

A comparison of the found solutions and performance metrics is shown in Table 9. Four algorithms solved this problem: DIRECT-NAS, DIRECT-GLh, DIRECT-GLce, and DIRECT-GLce-min, and hybridized DIRECT-GLce-min was the most efficient optimizer again. DIRECT-NAS is the best performing and outperformed the second-best by approximately 1.8 times, among traditional DIRECT-type algorithms. However, the DIRECT-NAS algorithm was about 26 times slower than the second-best method (DIRECT-GLh). As in the previous case, the DIRECT-L1 returned a better than the best know value f_{min}, but the solution point lies outside the feasible region again.

5.5. Welded beam design problem

The fifth engineering problem is the welded beam design. Here, the goal is to minimize a welded beam for a minimum cost, subject to seven constraints. The detailed description is presented in [50, 51], while in Appendix B.5, we provide a short description and mathematical formulation.

Algorithm (input)	Iterations	f_{eval}	Time (s)	f_{min}
DIRECT-NAS	273	31,081	126.64	7164.437307
DIRECT-Barrier	26,379	2,000,000	1,600.150	7234.041903
subDIRECT-Barrier ($sub = 5$)	15,626	2,000,000	1,328.98	7234.041903
DIRECT-GLh	252	55,837	4.80	7164.437300
DIRECT-GLc	2,358	2,000,000	433.87	7224.704257
DIRECT-GLce	322	88,585	8.52	7164.437301
DIRECT-GLce-min	5	91	0.05	7163.739611
DIRECT-L1 ($p = 10^3$)	87	2,295	0.23	7152.303079[a]

a – result is outside the feasible region

Table 9: Performance of the most promising DIRECT-type algorithms from DGO on a pressure vessel design problem

A comparison of the algorithms is shown in Table 10. In total, five algorithms were able to solve the problem, and once again, the DIRECT-GLce-min was the most efficient one. Again, the DIRECT-NAS algorithm showed the best performance (based on the total number of function evaluations) among traditional DIRECT-type algorithms, but significantly suffered based on the execution time.

Algorithm (input)	Iterations	f_{eval}	Time (s)	f_{min}
DIRECT-NAS	698	86,863	4692.00	1.724970
DIRECT-Barrier	22,934	2,000,000	1,363.24	1.728488
subDIRECT-Barrier ($sub = 5$)	10,954	2,000,000	860.06	1.728037
DIRECT-GLh	189	158,747	11.70	1.724970
DIRECT-GLc	211	108,683	9.19	1.724970
DIRECT-GLce	366	104,191	9.80	1.724970
DIRECT-GLce-min	3	163	0.06	1.724884
DIRECT-L1 ($p = 10^3$)	20,767	2,000,000	1,679.76	1.728488

Table 10: Performance of the most promising DIRECT-type algorithms from DGO on a welded beam design problem

5.6. General nonlinear regression problem

In the final part, a general nonlinear regression design problem is considered in the form of fitting a sum of damped sinusoids to a series of observations. The detailed description of the problem can be found in [24, 56, 73], while in Appendix B.6, we provide a short description and mathematical formulation. The problem is multimodal and is considered challenging, especially with the increase in the number of samples (T). The higher number of sinusoids (s) leads to a more accurate but at the same time and more challenging optimization problem. Our experiments have used three different values for $s = 1, 2,$ and 3 (correspond to $3, 6,$ and 9-dimensional problems) and two different values, $T = 10$ and $T = 100$ for each dimension n as was done in [56].

The obtained results are summarized in Table 11. Solving the lowest dimension ($n = 3$) cases (corresponding to $s = 1, T = 10$ and $s = 1, T = 100$) all algorithms located the solutions correctly. Excluding the hybridized BIRMIN, the PLOR algorithm was the most efficient one. However, in solving bigger conditionality problems ($n = 6$), half of the algorithms failed to find the correct solution. Finally, only two tested algorithms solved all nonlinear regression design problems: DIRECT-GL and BIRMIN.

6. Conclusion

This paper has introduced a new open-source DIRECT-type MATLAB toolbox (DGO) for derivative-free global optimization. The new toolbox combines various state-of-the-art DIRECT-type algorithms for the global solution of box-constrained, generally-constrained, and optimization problems with hidden constraints. All algorithms were implemented using two different data structures: static and dynamic. Additionally, several parallel schemes were adopted to promising algorithms. Furthermore, an online test library DIRECTlib, containing 119 global optimization test and engineering problems, has been presented.

The performance of various algorithms within DGO has been investigated via a detailed numerical study using the test problems from DIRECTlib. A further 11 examples of the use of the DGO for engineering design optimization have been investigated. The results demonstrate the promising performance of DGO in tackling these challenging problems. We also gave guidance on which algorithms to use for certain types of optimization problems.

Motivated by the promising performance, we plan to extend this work to facilitate the broader adoption of DGO. We plan to include newly appearing promising DIRECT-type algorithms within this toolbox continuously. Another direction is extending the developed algorithms using a hybrid CPU-GPU scheme. Finally, we will consider advanced data structures for better data organization and reduced communication overhead.

Source code statement

All implemented DIRECT-type algorithms (DGO toolbox) are available at the GitHub repository: https://github.com/blockchain-group/DGO, and can be used under GPL version 3.

Data statement

Data underlying this article (DIRECTlib) can be accessed on Zenodo at https://dx.doi.org/10.5281/zenodo.1218980, and used under the Creative Commons Attribution license.

References

[1] Baker, C.A., Watson, L.T., Grossman, B., Mason, W.H., Haftka, R.T., 2000. Parallel global aircraft configuration design space exploration, in: Tentner, A. (Ed.), High Performance Computing Symposium 2000, Soc. for Computer Simulation Internat. pp. 54–66.

[2] Bartholomew-Biggs, M.C., Parkhurst, S.C., Wilson, S.P., 2002. Using DIRECT to solve an aircraft routing problem. Computational Optimization and Applications 21, 311–323. doi:10.1023/A:1013729320435.

[3] Basudhar, A., Dribusch, C., Lacaze, S., Missoum, S., 2012. Constrained efficient global optimization with support vector machines. Structural and Multidisciplinary Optimization 46, 201–221. doi:10.1007/s00158-011-0745-5.

[4] Birgin, E.G., Floudas, C.A., Martínez, J.M., 2010. Global minimization using an augmented lagrangian method with variable lower-level constraints. Mathematical Programming 125, 139–162. doi:10.1007/s10107-009-0264-y.

	$s = 1, T = 10$				$s = 2, T = 10$				$s = 3, T = 10$			
Algorithm	Iteration	f_{eval}	Time(s)	f_{min}	Iteration	f_{eval}	Time(s)	f_{min}	Iteration	f_{eval}	Time(s)	f_{min}
Aggressive DIRECT	31	5,061	0.61	0.000087	311	2×10^6	227.87	0.005848	213	2×10^6	201.36	0.099088
DIRECT-GL	27	1,617	0.12	0.000021	184	62,965	2.80	0.000099	652	543,207	35.01	0.000086
DIRECT	32	501	0.24	0.000021	9,328	2×10^6	1,125.17	0.001228	3,186	2×10^6	553.10	0.007295
PLOR	35	305	0.04	0.000021	203,119	2×10^6	7,540.74	0.018320	93,457	2×10^6	2,590.26	0.009475
glbSolve	32	501	0.24	0.000021	2,805	1,978,935	762.10	0.000097	1,415	2×10^6	453.64	0.021190
BIRECT	58	616	0.43	0.000094	8,729	471,930	438.66	0.000096	36,981	2×10^6	7,780.35	0.004731
BIRMIN	5	101	0.03	1.04×10^{-13}	15	316	0.03	3.82×10^{-7}	85,458	722,243	5,915.73	8.40×10^{-9}
DISIMPL-V	151	3,024	3.19	0.000086	178,562	43,215.10	0.014843	0.014843	1	2×10^6	43,000.00	N/A
	$s = 1, T = 100$				$s = 2, T = 100$				$s = 3, T = 100$			
Aggressive DIRECT	36	6,921	1.54	0.000091	311	2×10^6	278.63	0.006979	213	2×10^6	280.41	0.117012
DIRECT-GL	29	1,829	0.19	0.000025	188	65,645	3.63	0.000097	251	158,989	9.60	0.000099
DIRECT	89	1,923	0.84	0.000025	9,191	2×10^6	1,178.59	0.001696	3,722	2×10^6	745.02	0.034532
PLOR	6,811	45,423	10.90	0.000025	203,159	2×10^6	8,823.94	0.020558	91,574	2×10^6	2,617.19	0.041690
glbSolve	89	1,923	0.82	0.000025	2,602	1,713,841	674.43	0.000096	1,273	2×10^6	504.28	0.033823
BIRECT	86	1,042	0.86	0.000096	13,028	675,332	792.08	0.000097	36,473	2×10^6	7,482.06	0.019857
BIRMIN	5	105	0.02	1.15×10^{-13}	15	292	0.06	2.23×10^{-7}	89,360	730,887	7,692.91	2.30×10^{-9}
DISIMPL-V	180	2,344	2.04	0.000086	6,840	185,135	43,000.00	0.018025	1	2×10^6	43,000.00	N/A

N/A – not available

Table 11: Performance of the most promising DIRECT-type algorithms from DGO on nonlinear regression design problem

[5] Björkman, M., Holmström, K., 1999. Global optimization using the DIRECT algorithm in Matlab. Advanced Modeling and Optimization 1, 17–37.

[6] Cagnina, L.C., Esquivel, S.C., Coello, C.A.C., 2008. Solving engineering optimization problems with the simple constrained particle swarm optimizer. Informatica (Ljubljana) 32, 319–326.

[7] Carter, R.G., Gablonsky, J.M., Patrick, A., Kelley, C.T., Eslinger, O.J., 2001. Algorithms for noisy problems in gas transmission pipeline optimization. Optimization and Engineering 2, 139–157. doi:10.1023/A:1013123110266.

[8] Characklis, G.W., Kirsch, B.R., Ramsey, J., Dillard, K.E., Kelley, C.T., 2006. Developing portfolios of water supply transfers. Water Resources Research doi:10.1029/2005WR004424.

[9] Chen, X., Kelley, C.T., 2016. Optimization with hidden constraints and embedded Monte Carlo computations. Optimization and Engineering 17, 157–175. doi:10.1007/s11081-015-9302-1.

[10] Choi, T.D., Eslinger, O.J., Kelley, C.T., David, J.W., Etheridge, M., 2000. Optimization of Automotive Valve Train Components with Implicit Filtering. Optimization and Engineering 1, 9–27. doi:10.1023/A:1010071821464.

[11] Clerc, M., 1999. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization, in: Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999, pp. 1951–1957. doi:10.1109/CEC.1999.785513.

[12] Costa, M.F.P., Rocha, A.M.A.C., Fernandes, E.M.G.P., 2018. Filter-based direct method for constrained global optimization. Journal of Global Optimization 71, 517–536. doi:10.1007/s10898-017-0596-8.

[13] Cox, S.E., Haftka, R.T., Baker, C.A., Grossman, B., Mason, W.H., Watson, L.T., 2001. A comparison of global optimization methods for the design of a high-speed civil transport. Journal of Global Optimization 21, 415–432. doi:10.1023/A:1012782825166.

[14] David, J.W., Kelley, C.T., Cheng, C., 1996. Use of an implicit filtering algorithm for mechanical system parameter identification. Technical Report. SAE Technical Paper.

[15] Di Serafino, D., Liuzzi, G., Piccialli, V., Riccio, F., Toraldo, G., 2011. A modified DIviding RECTangles algorithm for a problem in astrophysics. Journal of Optimization Theory and Applications 151, 175–190. doi:10.1007/s10957-011-9856-9.

[16] Dixon, L., Szegö, C., 1978. The global optimisation problem: An introduction, in: Dixon, L., Szegö, G. (Eds.), Towards Global Optimization. North-Holland Publishing Company. volume 2, pp. 1–15.

[17] Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Mathematical Programming 91, 201–213. doi:10.1007/s101070100263.

[18] Finkel, D.E., 2004. MATLAB source code for DIRECT. http://www4.ncsu.edu/~ctk/Finkel_Direct/. Online; accessed: 2017-03-22.

[19] Finkel, D.E., 2005. Global Optimization with the DIRECT Algorithm. Ph.D. thesis. North Carolina State University.

[20] Finkel, D.E., Kelley, C.T., 2004. An adaptive restart implementation of direct. Technical report CRSC-TR04-30, Center for Research in Scientific Computation, North Carolina State University, Raleigh .

[21] Finkel, D.E., Kelley, C.T., 2006. Additive scaling and the DIRECT algorithm. Journal of Global Optimization 36, 597–608. doi:10.1007/s10898-006-9029-9.

[22] Gablonsky, J.M., 2001. Modifications of the DIRECT Algorithm. Ph.D. thesis. North Carolina State University.

[23] Gablonsky, J.M., Kelley, C.T., 2001. A locally-biased form of the DIRECT algorithm. Journal of Global Optimization 21, 27–37. doi:10.1023/A:1017930332101.

[24] Gillard, J.W., Kvasov, D.E., 2017. Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Statistics and its Interface doi:10.4310/SII.2017.v10.n1.a6.

[25] Grbić, R., Nyarko, E.K., Scitovski, R., 2013. A modification of the DIRECT method for Lipschitz global optimizatio n for a symmetric function. Journal of Global Optimization 57, 1193–1212. doi:10.1007/s10898-012-0020-3.

[26] He, J., Verstak, A., Sosonkina, M., Watson, L.T., 2009a. Performance modeling and analysis of a massively parallel DIRECT–Part 2. The International Journal of High Performance Computing Applications 23, 29–41. doi:10.1177/1094342008098463.

[27] He, J., Verstak, A., Watson, L.T., Sosonkina, M., 2009b. Performance modeling and analysis of a massively parallel DIRECT–part 1. The International Journal of High Performance Computing Applications 23, 14–28. doi:10.1177/1094342008098462.

[28] He, J., Watson, L.T., Ramakrishnan, N., Shaffer, C.A., Verstak, A., Jiang, J., Bae, K., Tranter, W.H., 2002. Dynamic data structures for a DIRECT search algorithm. Computational Optimization and Applications 23, 5–25. doi:10.1023/A:1019992822938.

[29] He, J., Watson, L.T., Sosonkina, M., 2010. Algorithm 897: VTDIRECT95: Serial and Parallel Codes for the Global Optimization Algorithm DIRECT. ACM Transactions on Mathematical Software doi:10.1145/1527286.1527291.

[30] Hedar, A., 2005. Test functions for unconstrained global optimization. http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm. Online; accessed: 2017-03-22.

[31] Horst, R., Pardalos, P.M., Thoai, N.V., 1995. Introduction to Global Optimization. Nonconvex Optimization and Its Application, Kluwer Academic Publishers.

[32] John, H., 1975. Adaptation in natural and artificial systems. The

University of Michigan Press, Ann Arbor .

[33] Jones, D.R., 2001. The DIRECT global optimization algorithm, in: Floudas, C.A., Pardalos, P.M. (Eds.), The Encyclopedia of Optimization. Kluwer Academic Publishers, Dordrect, pp. 431–440.

[34] Jones, D.R., Martins, J.R.R.A., 2021. The DIRECT algorithm: 25 years later. Journal of Global Optimization 79, 521–566. doi:10.1007/s10898-020-00952-6.

[35] Jones, D.R., Perttunen, C.D., Stuckman, B.E., 1993. Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Application 79, 157–181. doi:10.1007/BF00941892.

[36] Kazemi, M., Wang, G.G., Rahnamayan, S., Gupta, K., 2011. Metamodel-based optimization for problems with expensive objective and constraint functions. Journal of Mechanical Design 133, 014505. doi:10.1115/1.4003035.

[37] Kennedy, J., Eberhart, R., 1995. Particle swarm optimization. In Proceedings of the IEEE international conference on neural networks IV , 1942–1948doi:10.1109/ICNN.1995.488968.

[38] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science doi:10.1126/science.220.4598.671.

[39] Kirsch, B.R., Characklis, G.W., Dillard, K.E., Kelley, C.T., 2009. More efficient optimization of long-term water supply portfolios. Water Resources Research doi:10.1029/2008WR007018.

[40] Liu, H., Xu, S., Chen, X., Wang, X., Ma, Q., 2017a. Constrained global optimization via a direct-type constraint-handling technique and an adaptive metamodeling strategy. Structural and Multidisciplinary Optimization 55, 155–177. doi:10.1007/s00158-016-1482-6.

[41] Liu, H., Xu, S., Wang, X., Wu, X., Song, Y., 2015a. A global optimization algorithm for simulation-based problems via the extended direct scheme. Engineering Optimization 47, 1441–1458. doi:10.1080/0305215X.2014.971777.

[42] Liu, Q., 2013. Linear scaling and the direct algorithm. Journal of Global Optimization 56, 1233–1245. doi:10.1007/s10898-012-9952-x.

[43] Liu, Q., Cheng, W., 2014. A modified DIRECT algorithm with bilevel partition. Journal of Global Optimization 60, 483–499. doi:10.1007/s10898-013-0119-1.

[44] Liu, Q., Yang, G., Zhang, Z., Zeng, J., 2017b. Improving the convergence rate of the DIRECT global optimization algorithm. Journal of Global Optimization 67, 851–872. doi:10.1007/s10898-016-0447-z.

[45] Liu, Q., Zeng, J., Yang, G., 2015b. MrDIRECT: a multilevel robust DIRECT algorithm for global optimization problems. Journal of Global Optimization 62, 205–227. doi:10.1007/s10898-014-0241-8.

[46] Liuzzi, G., Lucidi, S., Piccialli, V., 2010a. A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems. Computational Optimization and Applications 45, 353–375. doi:10.1007/s10589-008-9217-2.

[47] Liuzzi, G., Lucidi, S., Piccialli, V., 2010b. A partition-based global optimization algorithm. Journal of Global Optimization 48, 113–128. doi:10.1007/s10898-009-9515-y.

[48] Luszczek, P., 2009. Parallel programming in MATLAB. International Journal of High Performance Computing Applications 23, 277–283. doi:10.1177/1094342009106194.

[49] Matlab, 2020. Parallel Computing Toolbox ™ User ' s Guide. Book , 1–729.

[50] Mirjalili, S., Lewis, A., 2016. The Whale Optimization Algorithm. Advances in Engineering Software doi:10.1016/j.advengsoft.2016.01.008.

[51] Mirjalili, S., Mirjalili, S.M., Lewis, A., 2014. Grey Wolf Optimizer. Advances in Engineering Software doi:10.1016/j.advengsoft.2013.12.007.

[52] Mockus, J., Paulavičius, R., Rusakevičius, D., Šešok, D., Žilinskas, J., 2017. Application of Reduced-set Pareto-Lipschitzian Optimization to truss optimization. Journal of Global Optimization 67, 425–450. doi:10.1007/s10898-015-0364-6.

[53] Na, J., Lim, Y., Han, C., 2017. A modified DIRECT algorithm for hidden constraints in an LNG process optimization. Energy , 488–500doi:10.1016/j.energy.2017.03.047.

[54] Paulavičius, R., Chiter, L., Žilinskas, J., 2018. Global optimization based on bisection of rectangles, function values at diagonals, and a set of Lipschitz constants. Journal of Global Optimization 71, 5–20. doi:10.1007/s10898-016-0485-6.

[55] Paulavičius, R., Sergeyev, Y.D., Kvasov, D.E., Žilinskas, J., 2014. Globally-biased DISIMPL algorithm for expensive global optimization. Journal of Global Optimization 59, 545–567. doi:10.1007/s10898-014-0180-4.

[56] Paulavičius, R., Sergeyev, Y.D., Kvasov, D.E., Žilinskas, J., 2020. Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. Expert Systems with Applications 144, 11305. doi:10.1016/j.eswa.2019.113052.

[57] Paulavičius, R., Žilinskas, J., 2006. Analysis of different norms and corresponding Lipschitz constants for global optimization. Technological and Economic Development of Economy 36, 383–387. doi:10.1080/13928619.2006.9637758.

[58] Paulavičius, R., Žilinskas, J., 2007. Analysis of different norms and corresponding Lipschitz constants for global optimization in multidimensional case. Information Technology and Control 36, 383–387.

[59] Paulavičius, R., Žilinskas, J., 2013. Simplicial Lipschitz optimization without the Lipschitz constant. Journal of Global Optimization 59, 23–40. doi:10.1007/s10898-013-0089-3.

[60] Paulavičius, R., Žilinskas, J., 2014. Simplicial Global Optimization. SpringerBriefs in Optimization, Springer New York, New York, NY. doi:10.1007/978-1-4614-9093-7.

[61] Paulavičius, R., Žilinskas, J., 2016. Advantages of simplicial partitioning for Lipschitz optimization problems with linear constraints. Optimization Letters 10, 237–246. doi:10.1007/s11590-014-0772-4.

[62] Pillo, G.D., Liuzzi, G., Lucidi, S., Piccialli, V., Rinaldi, F., 2016. A DIRECT-type approach for derivative-free constrained global optimization. Computational Optimization and Applications 65, 361–397. doi:10.1007/s10589-016-9876-3.

[63] Pillo, G.D., Lucidi, S., Rinaldi, F., 2010. An approach to constrained global optimization based on exact penalty functions. Journal of Optimization Theory and Applications 54, 251–260. doi:10.1007/s10898-010-9582-0.

[64] Pintér, J.D., 1996. Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications. volume 6 of *Nonconvex Optimization and Its Applications*. Springer US. doi:10.1007/978-1-4757-2502-5.

[65] Piyavskii, S.A., 1967. An algorithm for finding the absolute minimum of a function. Theory of Optimal Solutions 2, 13–24. doi:10.1016/0041-5553(72)90115-2. in Russian.

[66] Preparata, F.P., Shamos, M., 1985. Computational Geometry. Monographs in Computer Science, Springer-Verlag New York. doi:10.1007/978-1-4612-1098-6.

[67] Ray, T., Liew, K.M., 2003. Society and civilization: An optimization algorithm based on the simulation of social behavior. IEEE Transactions on Evolutionary Computation 7, 386–396. doi:10.1109/TEVC.2003.814902.

[68] Rios, L.M., Sahinidis, N.V., 2007. Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization 56, 1247–1293. doi:10.1007/s10898-012-9951-y.

[69] Sergeyev, Y.D., Kvasov, D.E., 2006. Global search based on diagonal partitions and a set of Lipschitz constants. SIAM Journal on Optimization 16, 910–937. doi:10.1137/040621132.

[70] Sergeyev, Y.D., Kvasov, D.E., 2008. Diagonal Global Optimization Methods. FizMatLit, Moscow. In Russian.

[71] Sergeyev, Y.D., Kvasov, D.E., 2011. Lipschitz global optimization, in: Cochran, J.J., Cox, L.A., Keskinocak, P., Kharoufeh, J.P., Smith, J.C. (Eds.), Wiley Encyclopedia of Operations Research and Management Science (in 8 volumes). John Wiley & Sons, New York. volume 4, pp. 2812–2828.

[72] Sergeyev, Y.D., Kvasov, D.E., 2017. Deterministic Global Optimization: An Introduction to the Diagonal Approach.

SpringerBriefs in Optimization, Springer. doi:10.1007/978-1-4939-7199-2.

[73] Sergeyev, Y.D., Kvasov, D.E., Mukhametzhanov, M.S., 2016. On the least-squares fitting of data by sinusoids. Springer Optimization and Its Applications doi:10.1007/978-3-319-29975-4_11.

[74] Shubert, B.O., 1972. A sequential method seeking the global maximum of a function. SIAM Journal on Numerical Analysis 9, 379–388. doi:10.1137/0709036.

[75] Stoneking, D.E., Bilbro, G.L., Gilmore, P.A., Trew, R.J., Kelley, C.T., 1992. Yield optimization using a gaas process simulator coupled to a physical device model. IEEE Transactions on Microwave Theory and Techniques 40, 1353–1363. doi:10.1109/22.146318.

[76] Stripinis, L., Paulavičius, R., 2021. DGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization. https://github.com/blockchain-group/DGO.

[77] Stripinis, L., Paulavičius, R., 2021. A new DIRECT-GLh algorithm for global optimization with hidden constraints. Optimization Letters , 1–15doi:10.1007/s11590-021-01726-z.

[78] Stripinis, L., Paulavičius, R., Žilinskas, J., 2018. Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT. Optimization Letters 12, 1699–1712. doi:10.1007/s11590-017-1228-4.

[79] Stripinis, L., Paulavičius, R., Žilinskas, J., 2019. Penalty functions and two-step selection procedure based DIRECT-type algorithm for constrained global optimization. Structural and Multidisciplinary Optimization 59, 2155–2175. doi:10.1007/s00158-018-2181-2.

[80] Stripinis, L., Paulavičius, R., 2020. DIRECTLib – a library of global optimization problems for DIRECT-type methods, v1.2. URL: https://zenodo.org/record/3948890#.XyVOk5dR2Uk, doi:10.5281/zenodo.3948890.

[81] Stripinis, L., Žilinskas, J., Casado, L.G., Paulavičius, R., 2020. On MATLAB experience in accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization. Applied Mathematics and Computation , 1–25Revised.

[82] Strongin, R.G., Sergeyev, Y.D., 2000. Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms. Kluwer Academic Publishers, Dordrecht.

[83] Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Y.P.Chen, Auger, A., Tiwari, S., 2005. Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization. KanGAL , 251–256.

[84] Surjanovic, S., Bingham, D., 2013. Virtual library of simulation experiments: Test functions and datasets. http://www.sfu.ca/~ssurjano/index.html. Online; accessed: 2017-03-22.

[85] Vaz, A., Vicente, L., 2009. Pswarm: a hybrid solver for linearly constrained global derivative-free optimization. Optimization Methods and Software 24, 669–685. doi:10.1080/10556780902909948.

[86] Watson, L.T., Baker, C.A., 2001. A fully-distributed parallel global search algorithm. engineering computations. Engineering Computations 18, 155–169. doi:10.1108/02644400110365851.

A. DIRECTlib library

A summary of all optimization problems in DIRECTlib and their properties is given in Table 12. The first column denotes the problem type, and the second contains the problem name. The third column contains the source of the problem. The fourth through eighth columns (n, $\mathbf{g}$, $\mathbf{h}$, a, D) has the dimension (n), number of inequality ($\mathbf{g}$), and equality ($\mathbf{h}$) constraints, number of active constraints (a), and optimization domain (D), respectively. Finally, the last column contains the best known optimal solution (f^*).

Table 12: Key characteristics of the DIRECTlib [80] test problems for box-constrained global optimization

Problem		Source	Problem properties					f^*
Type	Name		n	**g**	**h**	a	D	
BC	*Ackley*	[30, 84]	n	0	0	0	$[-15, 30]_i, i = 1...n$	0.0000
	Alpine	[11]	n	0	0	0	$[0, 10]_i, i = 1...n$	-2.8081^n
	Beale	[30, 84]	2	0	0	0	$[-4.5, 4.5]_i, i = 1...n$	0.0000
	Bohachevsky 1	[30, 84]	2	0	0	0	$[-100, 110]_i, i = 1...n$	0.0000
	Bohachevsky 2	[30, 84]	2	0	0	0	$[-100, 110]_i, i = 1...n$	0.0000
	Bohachevsky 3	[30, 84]	2	0	0	0	$[-100, 110]_i, i = 1...n$	0.0000
	Booth	[30, 84]	2	0	0	0	$[-10, 10]_i, i = 1...n$	0.0000
	Branin	[30, 16, 84]	2	0	0	0	$[-5, 10]_1, [10, 15]_2$	0.3978
	Bukin	[84]	2	0	0	0	$[-15, 5]_1, [-3, 3]_2$	0.0000
	Csendes	[11]	n	0	0	0	$[-10, 20]_i, i = 1...n$	0.0000
	Colville	[30, 84]	4	0	0	0	$[-10, 10]_i, i = 1...n$	0.0000
	Cross-in-Tray	[84]	2	0	0	0	$[-10, 10]_i, i = 1...n$	−2.0626
	Dixon-Price	[30, 84]	n	0	0	0	$[-10, -10]_i, i = 1...n$	0.0000
	Drop-Wave	[84]	2	0	0	0	$[-5.12, -6.12]_i, i = 1...n$	−1.0000
	Easom	[30, 84]	2	0	0	0	$[-100, 100]_i, i = 1...n$	−1.0000
	Eggholder	[84]	2	0	0	0	$[-512, 512]_i, i = 1...n$	−959.6406
	Goldstein & Price	[30, 16, 84]	2	0	0	0	$[-2, 2]_i, i = 1...n$	3.0000
	Griewank	[30, 84]	n	0	0	0	$[-600, 700]_i, i = 1...n$	0.0000
	Hartman 3	[30, 84]	3	0	0	0	$[0, 1]_i, i = 1...n$	−3.8627
	Hartman 6	[30, 84]	6	0	0	0	$[0, 1]_i, i = 1...n$	−3.3223
	Holder Table	[84]	2	0	0	0	$[-10, 10]_i, i = 1...n$	−19.2085
	Hump	[30, 84]	2	0	0	0	$[-5, 5]_i, i = 1...n$	−1.0316
	Langermann	[84]	2	0	0	0	$[0, 10]_i, i = 1...n$	−4.1558
	Levy	[30, 84]	n	0	0	0	$[-5, 5]_i, i = 1...n$	0.0000
	Matyas	[30, 84]	2	0	0	0	$[-10, 10]_i, i = 1...n$	0.0000
	McCormick	[84]	2	0	0	0	$[-1.5, 4]_1, [-3, 4]_2$	−1.9132
	Michalewicz	[30, 84]	2	0	0	0	$[0, \pi]_i, i = 1...n$	−1.8013
	Michalewicz	[30, 84]	5	0	0	0	$[0, \pi]_i, i = 1...n$	−4.6876
	Michalewicz	[30, 84]	10	0	0	0	$[0, \pi]_i, i = 1...n$	−9.6601
	Perm($\beta = 0.5$)	[30, 84]	5	0	0	0	$[-n, n]_i, i = 1...n$	0.0000
	Perm($\beta = 10$)	[30, 84]	8	0	0	0	$[-n, n]_i, i = 1...n$	0.0000
	Powell 4	[30, 84]	4	0	0	0	$[-4, 5]_i, i = 1...n$	0.0000
	Power Sum	[30, 84]	4	0	0	0	$[0, 4]_i, i = 1...n$	0.0000
	Qing	[11]	n	0	0	0	$[-500, 500]_i, i = 1...n$	0.0000
	Rastrigin	[30, 84]	n	0	0	0	$[-6.12, 5.12]_i, i = 1...n$	0.0000
	Rosenbrock	[30, 16, 84]	n	0	0	0	$[-5, 10]_i, i = 1...n$	0.0000
	Rotated Ellipsoid	[84]	n	0	0	0	$[-65.536, 66.536]_i, i = 1...n$	0.0000
	Schwefel	[30, 84]	n	0	0	0	$[-500, 500]_i, i = 1...n$	0.0000
	Shekel($m = 5$)	[30, 84]	4	0	0	0	$[0, 10]_i, i = 1...n$	−10.1531
	Shekel($m = 7$)	[30, 84]	4	0	0	0	$[0, 10]_i, i = 1...n$	−10.4029
	Shekel($m = 10$)	[30, 84]	4	0	0	0	$[0, 10]_i, i = 1...n$	−10.5364
	Shubert	[30, 84]	2	0	0	0	$[-10, 10]_i, i = 1...n$	−186.7309
	Sphere	[30, 84]	n	0	0	0	$[-5.12, 6.12]_i, i = 1...n$	0.0000
	Styblinski-Tang	[11]	n	0	0	0	$[-5, 5]_i, i = 1...n$	$-39.1661n$
	Sum of Powers	[84]	n	0	0	0	$[-10, 15]_i, i = 1...n$	0.0000
	Trid	[30, 84]	6	0	0	0	$[-36, 36]_i, i = 1...n$	−50.0000
	Trid	[30, 84]	10	0	0	0	$[-100, 100]_i, i = 1...n$	−210.0000
	Zakharov	[30, 84]	n	0	0	0	$[-5, 11]_i, i = 1...n$	0.0000
LC	*Bunnag* 1	[85]	4	1	0	1	$[0, 3]_i, i = 1...n$	0.1117
	Bunnag 2	[85]	4	2	0	2	$[0, 4]_i, i = 1...n$	−6.4049
	Bunnag 3	[85]	5	3	0	1	$[0, 3]_1, [0, 2]_{2,5}, [0, 4]_{3,4}$	−16.3657
	Bunnag 4	[85]	6	2	0	1	$[0, 1]_{1,2,3,4,5}, [0, 20]_6$	−213.0470
	Bunnag 5	[85]	6	5	0	1	$[0, 2]_{1,3,6}, [0, 8]_2, [0, 1]_{4,5}$	−11.0000
	Bunnag 6	[85]	10	11	0	3	$[0, 1]_i, i = 1...n$	−268.0146
	Bunnag 7	[85]	10	5	0	0	$[0, 1]_i, i = 1...n$	−39.0000
	G01	[83]	13	9	0	6	$[0, 10]_i, [0, 100]_{10}, i = 1...9, 11$	−15.0000
	Genocop 9	[85]	3	5	0	2	$[0, 10]_i, i = 1...n$	−2.4714
	Genocop 10	[85]	4	5	0	0	$[0, 3]_1, [0, 10]_{2,3}, [0, 1]_4$	−4.5280
	Genocop 11	[85]	6	5	0	0	$[0, 5]_{1,3}, [0, 8]_2, [0, 1]_{4,5}, [0, 2]_6$	−11.0000
	Horst 1	[31]	2	3	0	1	$[0, 3]_1, [0, 2]_2$	−1.0625
	Horst 2	[31]	2	3	0	2	$[0, 2.5]_1, [0, 2]_2$	−6.8995
	Horst 3	[31]	2	3	0	0	$[0, 1]_1, [0, 1.5]_2$	−0.4444
	Horst 4	[31]	3	4	0	2	$[0.5, 2]_1, [0, 3]_2, [0, 2.8]_3$	−6.0858
	Horst 5	[31]	3	4	0	0	$[0, 1.2]_{1,2}, [0, 1.7]_3$	−3.7220
	Horst 6	[31]	3	7	0	2	$[0, 6]_1, [0, 5.0279]_2, [0, 2.6]_3$	−32.5784
	Horst 7	[31]	3	4	0	2	$[0, 6]_1, [0, 3]_{2,3}$	−52.8769
	hs021	[85]	2	1	0	1	$[2, 50]_1, [-50, 10]_2$	−99.9599
	hs021mod	[85]	7	3	0	1	$[2, 50]_1, [-50, 50]_2, [0, 50]_3, [2, 10]_4, [-10, 10]_5, [-10, 0]_6, [0, 10]_7$	4.0400
	hs024	[85]	2	3	0	2	$[0, 5]_i, i = 1...n$	−1.0000
	hs035	[85]	3	1	0	1	$[0, 3]_i, i = 1...n$	0.1111
	hs036	[85]	3	1	0	1	$[0, 20]_1, [0, 11]_2, [0, 15]_3$	−3300.0000
	hs037	[85]	3	2	0	1	$[0, 42]_i, i = 1...n$	−3456.0000
	hs038	[85]	4	2	0	0	$[-10, 10]_i, i = 1...n$	0.0000
	hs044	[85]	4	6	0	2	$[0, 5]_i, i = 1...n$	−15.0000
	hs076	[85]	4	3	0	1	$[0, 1]_{1,3,4}, [0, 3]_2$	−4.6818
	P14	[4]	4	4	0	2	$[10^{-5}, 3]_1, [10^{-5}, 4]_2, [0, 2]_3, [0, 1]_4$	−4.51420

Continued on next page

Table 12 Continued from previous page

Problem		Source	Problem properties					f^*
Type	Name		n	**g**	**h**	a	D	
	s224	[85]	2	4	0	1	$[0,6]_1, [0,11]_2$	−304.0000
	s231	[85]	2	2	0	0	$[-10,10]_i, i = 1...n$	0.0000
	s232	[85]	2	3	0	2	$[0,100]_i, i = 1...n$	−1.0000
	s250	[85]	3	2	0	1	$[0,20]_1, [0,11]_2, [0,42]_3$	−3300.0000
	s251	[85]	3	1	0	1	$[0,42]_i, i = 1...n$	−3456.0000
	zecevic2	[85]	3	2	0	1	$[0,10]_i, i = 1...n$	−4.1249
	zecevic3	[85]	2	2	0	1	$[0,10]_i, i = 1...n$	97.3094
	zecevic4	[85]	4	2	0	1	$[0,10]_i, i = 1...n$	7.5575
GC	*circle*	[85]	3	10	0	3	$[0,10]_i, i = 1...n$	4.5742
	G02	[83]	20	2	0	1	$[0,10]_i, i = 1...n$	−0.8036
	G03	[83]	10	0	1	1	$[0,10]_i, i = 1...n$	−1.0005
	G04	[83]	5	6	0	2	$[78,102]_1, [33,45]_2, [27,45]_{3,4,5}$	−30665.5386
	G05	[83]	4	2	3	3	$[10,1,200]_{1,2}, [-0.55,0.55]_{3,4}$	5126.4967
	G06	[83]	2	2	0	2	$[13,100]_1, [0,100]_2$	−6961.8138
	G07	[83]	10	8	0	6	$[-10,10]_i, i = 1...n$	24.3062
	G08	[83]	2	2	0	0	$[0,10]_i, i = 1...n$	−0.0958
	G09	[83]	7	4	0	2	$[-10,10]_i, i = 1...n$	680.6300
	G10	[83]	8	6	0	6	$[100,10,000]_1, [1,000,10,000]_{2,3}, [10,1,000]_i, i = 4...8$	7049.2480
	G11	[83]	2	0	1	1	$[-1,1]_i, i = 1...n$	0.7499
	G12	[83]	3	1	0	0	$[0.2,10]_i, i = 1...n$	−1.0000
	G13	[83]	5	0	3	3	$[-2.3,2.3]_{1,2}, [-3.2,3.2]_{3,4,5}$	0.0539
	G16	[83]	5	38	0	4	$[704.4148,906.3855]_1, [68.6,288.88]_2,$ $[0,134.75]_3, [193,287.0966]_4, [25,84.1988]_5$	−1.9051
	G18	[83]	9	13	0	6	$[0,10]_i, i = 1...n$	−0.8660
	G19	[83]	15	5	0	0	$[0,10]_i, i = 1...n$	32.6555
	G24	[83]	2	2	0	2	$[0,3]_1, [0,4]_2$	−5.5080
	*Goldstein & Price**	[53]	2	2	0	1	$[-2,2]_i, i = 1...n$	3.5389
	Gomez	[4]	2	1	0	1	$[-1,1]_i, i = 1...n$	−0.9711
	Himmelblau	[6]	5	5	0	2	$[78,102]_1, [33,45]_2, [27,45]_{3,4,5}$	−31025.5602
	P01	[4]	5	0	3	3	$[-5,5]_i, i = 1...n$	0.0293
	P02a	[4]	9	10	0	5	$[0,100]_1, [0,500]_i, 2 = 1...n$	−400.0000
	P02b	[4]	9	10	0	5	$[0,600]_1, [0,500]_i, 2 = 1...n$	−600.0000
	P02c	[4]	9	10	0	4	$[0,100]_1, [0,500]_i, 2 = 1...n$	−750.0000
	P02d	[4]	10	12	0	5	$[0,300]_{1,2,6}, [0,100]_{3,5,7}, [0,200]_{4,8}, [0,3]_9$	−600.0000
	P03a	[4]	6	1	4	5	$[0,1]_{1,2,3,4}, [10^{-5},16]_{5,6}$	0.3888
	P03b	[4]	2	1	0	1	$[10^{-5},16]_i, i = 1...n$	0.3888
	P04	[4]	2	1	0	1	$[0,6]_1, [0,4]_2$	−6.6666
	P05	[4]	3	2	0	2	$[0,9.422]_1, [0,5.903]_2, [0,267.42]_3$	201.1600
	P06	[4]	2	1	0	1	$[0,115.8]_1, [10^{-5},30]_2$	376.2900
	P07	[4]	2	4	0	1	$[-2,2]_i, i = 1...n$	−2.8284
	P08	[4]	2	2	0	1	$[-8,10]_1, [0,10]_2$	−118.7000
	P09	[4]	6	9	0	2	$[10^{-5},3]_1, [10^{-5},4]_{2,3}, [0,2]_{4,5}, [0,6]_6$	−13.4020
	P10	[4]	2	2	0	2	$[0,1]_i, i = 1...n$	0.7417
	P11	[4]	2	1	0	1	$[0,1]_i, i = 1...n$	−0.5000
	P12	[4]	2	2	0	0	$[0,2]_1, [0,3]_2$	−16.7390
	P13	[4]	3	0	2	2	$[10^{-5},34]_1, [10^{-5},17]_2, [100,300]_3$	189.3500
	P15	[4]	3	0	3	3	$[10^{-5},12.5]_1, [10^{-5},37.5]_2, [0,50]_3$	0.0000
	P16	[4]	5	6	0	0	$[0,1.5834]_1, [0,3.625]_2, [0,1]_3, [0,3]_4, [0,4]_5$	0.7049
	s365mod	[85]	7	9	0	5	$[0,19]_i, i = 1...n$	52.1399
	T1	[19]	n	1	0	1	$[-4,4]_i, i = 1...n$	$-n$
	zy2	[85]	2	3	0	1	$[0,10]_i, i = 1...n$	2.0000

Concluded

B. The mathematical formulation of engineering problems

B.1. Tension/compression spring design problem

The design variables of the tension/compression spring design problem [36] are the number of the wire diameter x_1, the winding diameter x_2, and active coils of the spring x_3. The objective function and the mechanical constraints are given by:

$$
\begin{aligned}
\min\ & f(\mathbf{x}) = x_1^2 x_2 (x_3 + 2) \\
\text{s.t. } & g_1(\mathbf{x}) = 1 - \frac{x_2^3 x_3}{71875 x_1^4} \leq \mathbf{0}, \\
& g_2(\mathbf{x}) = \frac{x_2(4x_2 - x_1)}{12566 x_1^3 (x_2 - x_1)} + \frac{2.46}{12566 x_1^2} - 1 \leq \mathbf{0}, \\
& g_3(\mathbf{x}) = 1 - \frac{140.54 x_1}{x_3 x_2^2} \leq \mathbf{0}, \\
& g_4(\mathbf{x}) = \frac{x_1 + x_2}{1.5} - 1 \leq \mathbf{0}
\end{aligned}
$$

where $0.05 \leq x_1 \leq 0.2,\ 0.25 \leq x_2 \leq 1.3,\ 2 \leq x_3 \leq 15$. The best known solution $\mathbf{x}^* = (0.05170517, 0.35710042, 11.28120672)$, where $f(\mathbf{x}^*) = 0.01267931$. Two of the constraint functions are active (g_1 and g_2).

B.2. Three-bar truss design problem

The three-bar truss design problem [67] has two design variables and three constraints. The optimization problem is formulated as follows:

$$
\begin{aligned}
\min\ & f(\mathbf{x}) = 100(2\sqrt{2}x_1 + x_2) \\
\text{s.t. } & g_1(\mathbf{x}) = \frac{\sqrt{2}x_1 + x_2}{\sqrt{2}x_1^2 + 2x_1 x_2} 2 - 2 \leq \mathbf{0}, \\
& g_2(\mathbf{x}) = \frac{x_2}{\sqrt{2}x_1^2 + 2x_1 x_2} 2 - 2 \leq \mathbf{0}, \\
& g_3(\mathbf{x}) = \frac{1}{x_1 + \sqrt{2}x_2} 2 - 2 \leq \mathbf{0}
\end{aligned}
$$

where $0 \leq x_1 \leq 1,\ 0 \leq x_2 \leq 1$. The best known solution $\mathbf{x}^* = (0.78867512, 0.40824832)$, where $f(\mathbf{x}^*) = 263.89584535$. One of the constraint functions is active (g_1).

B.3. NASA speed reducer design problem

The design variables of the NASA speed reducer design problem [67] are the face width x_1, the module of teeth x_2, the number of teeth on the pinion x_3, the length of the first shaft between the bearings x_4, the distance of the second shaft between the bearings x_5, the diameter of the first shaft x_6, and, finally, the width of the second shaft x_7. The optimization problem is formulated as follows:

$$
\begin{aligned}
\min\ f(\mathbf{x}) = {} & 0.7854 x_1 x_2^2 (3.3333 x_3^2 + 14.9334 x_3 - 43.0934) \\
& - 1.508 x_1 (x_6^2 + x_7^2) + 7.4777 (x_6^3 + x_7^3) \\
& + 0.7854 (x_4 x_6^2 + x_5 x_7^2) \\
\text{s.t. } & g_1(\mathbf{x}) = \frac{27}{x_1 x_2^2 x_3} - 1 \leq \mathbf{0},\ g_2(\mathbf{x}) = \frac{397.5}{x_1 x_2^2 x_3^2} - 1 \leq \mathbf{0}, \\
& g_3(\mathbf{x}) = \frac{1.93 x_4^3}{x_2 x_3 x_6^4} - 1 \leq \mathbf{0},\ g_4(\mathbf{x}) = \frac{1.93 x_5^3}{x_2 x_3 x_7^4} - 1 \leq \mathbf{0}, \\
& g_5(\mathbf{x}) = \frac{((\frac{745 x_4}{x_2 x_3})^2 + 16.9 \times 10^6)^{0.5}}{110 x_6^3} - 1 \leq \mathbf{0}, \\
& g_6(\mathbf{x}) = \frac{((\frac{745 x_5}{x_2 x_3})^2 + 157.5 \times 10^6)^{0.5}}{85 x_7^3} - 1 \leq \mathbf{0}, \\
& g_7(\mathbf{x}) = \frac{x_2 x_3}{40} - 1 \leq \mathbf{0},\ g_8(\mathbf{x}) = \frac{5 x_2}{x_1} - 1 \leq \mathbf{0}, \\
& g_9(\mathbf{x}) = \frac{x_1}{12 x_2} - 1 \leq \mathbf{0},\ g_{10}(\mathbf{x}) = \frac{1.5 x_6 + 1.9}{x_4} - 1 \leq \mathbf{0}, \\
& g_{11}(\mathbf{x}) = \frac{1.1 x_7 + 1.9}{x_5} - 1 \leq \mathbf{0}
\end{aligned}
$$

where
$2.6 \leq x_1 \leq 3.6,\ 0.7 \leq x_2 \leq 0.8,\ 17 \leq x_3 \leq 28,\ 7.3 \leq x_4 \leq 8.3,\ 7.8 \leq x_5 \leq 8.3,\ 2.9 \leq x_6 \leq 3.9,\ 5 \leq x_7 \leq 5.5$. The best known solution $\mathbf{x}^* = (3.5, 0.7, 17, 7.3, 7.8, 3.35021468, 5.28668323)$, where $f(\mathbf{x}^*) = 2996.34817613$. Three constraints are active (g_5, g_6 and g_8).

B.4. Pressure vessel design problem

There are four design variables in the pressure vessel design problem [36](in inches): the thickness of the pressure vessel x_1, the thickness of the head x_2, the inner radius of the vessel x_3, and the length of the cylindrical component x_4. The optimization problem is formulated as follows:

$$
\begin{aligned}
\min\ f(\mathbf{x}) = {} & 0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2 + 3.1661 x_1^2 x_4 \\
& + 19.84 x_1^2 x_3 \\
\text{s.t. } & g_1(\mathbf{x}) = -x_1 + 0.0193 x_3 \leq \mathbf{0}, \\
& g_2(\mathbf{x}) = -x_2 + 0.00954 x_3 \leq \mathbf{0}, \\
& g_3(\mathbf{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq \mathbf{0}, \\
& g_4(\mathbf{x}) = x_4 - 240 \leq \mathbf{0},\ g_5(\mathbf{x}) = 1.1 - x_1 \leq \mathbf{0}, \\
& g_6(\mathbf{x}) = 0.6 - x_2 \leq \mathbf{0}
\end{aligned}
$$

where $1 \leq x_1 \leq 1.375,\ 0.625 \leq x_2 \leq 1,\ 25 \leq x_3 \leq 150,\ 25 \leq x_4 \leq 240$. The best known solution $\mathbf{x}^* = (1.1, 0.625, 56.99481866, 51.00125165)$, where $f(\mathbf{x}^*) = 7163.73957163$. Three constraints are active (g_1, g_3 and g_5).

B.5. Welded beam design problem

The welded beam design problem [50, 51] is to design a welded beam at minimum cost, subject to some constraints

[50, 51]. The objective is to find a minimum fabrication cost. Considering the four design variables and constraints of shear stress τ, bending stress in the beam σ, buckling load on the bar P_c, and end deflection on the beam δ. The optimization model is summarized in the following equation:

$$\begin{aligned}
\min\ & f(\mathbf{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14 + x_2)\\
\text{s.t. } & g_1(\mathbf{x}) = \tau(\mathbf{x}) - 13600 \le \mathbf{0},\\
& g_2(\mathbf{x}) = \sigma(\mathbf{x}) - 3\times 10^4 \le \mathbf{0},\\
& g_3(\mathbf{x}) = x_1 - x_4 \le \mathbf{0},\\
& g_4(\mathbf{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14 + x_2) - 5 \le \mathbf{0},\\
& g_5(\mathbf{x}) = \delta(\mathbf{x}) - 0.25 \le \mathbf{0},\\
& g_6(\mathbf{x}) = P - P_c(\mathbf{x}) \le \mathbf{0},\\
& g_7(\mathbf{x}) = 0.125 - x_1 \le \mathbf{0},
\end{aligned}$$

with:

$$\begin{aligned}
&\tau(\mathbf{x}) = \sqrt{(\tau^1)^2 + (\tau^1)(\tau^2)x_2/R + (\tau^2)^2},\\
&\tau^1 = \frac{P}{\sqrt{2}x_1x_\varepsilon}, \tau^2 = \frac{MR}{J}, M = P(L + \frac{x_2}{2}),\\
&R = \sqrt{\frac{x_2^2)}{4} + (\frac{x_1 + x_3}{2})^2}, \sigma(\mathbf{x}) = \frac{6PL}{x_4x_3^2},\\
&J = 2(\sqrt{2}x_1x_2(\frac{x_2^2}{12} + \frac{1}{4}(x_1x_3)^2)), \delta(\mathbf{x}) = \frac{4PL^3}{Ex_4x_3^3},\\
&P_c = \frac{4.013E\sqrt{x_3^2x_4\varepsilon 6/36}}{L^2}(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}),\\
&P = 6000, L = 14, E = 3\times 10^7, G = 12\times 10^6,
\end{aligned}$$

where $0.1 \le x_1 \le 2,\ 0.1 \le x_2 \le 10,\ 0.1 \le x_3 \le 10,\ 0.1 \le x_3 \le 2$. The best known solution $\mathbf{x}^* = (0.20572551, 3.47062057, 9.03666456, 0.20573141)$, where $f(\mathbf{x}^*) = 1.72488430$. One of the constraint functions is active (g_3).

B.6. General nonlinear regression problem

Parameter estimation in the general nonlinear regression model [24, 56, 73] can be reduced to solving the minimization problem:

$$\min f(\mathbf{x}) = \sum_{t=1}^{T}(\kappa(t) - \phi(\mathbf{x}, t))^2$$

with:

$$\begin{aligned}
&\sum_{q=1}^{s}\kappa(t) = e^{td_q}\sin(2\pi t\omega_q + \theta_q),\\
&\sum_{q=1}^{s}\phi(\mathbf{x}, t) = e^{(x_{3(q-1)+1}t)}\sin(2\pi t x_{3(q-1)+2} + x_{3(q-1)+3})
\end{aligned}$$

where $-1 \le x_{3(q-1)+1} \le 0,\ 0 \le x_{3(q-1)+2,3(q-1)+2} \le 1, q = 1...s$. $\mathbf{d}$ is non-positive damping coefficients, $\boldsymbol{\omega}$ is frequencies, and $\boldsymbol{\theta}$ is phases of the sinusoids (hereafter, $d_q \in [-1, 0],\ \omega_q \in [0, 1],\ \theta_q \in [0, 1],\ q = 1...s$). For signal approximation, the parameter $\mathbf{x}$ of the problem is determined to fit best the real-valued signal values observed in the uniformly distributed time moments $t = 1, 2, ..., T$. The best known solution $f(\mathbf{x}^*) = 0$, where $\mathbf{x}^* = (-0.2, 0.4, 0.3)$, for $n = 3$, $\mathbf{x}^* = (-0.2, 0.4, 0.3, -0.3, 0.3, 0.1)$, for $n = 6$, and $\mathbf{x}^* = (-0.4, 0.6, 0.2, -0.3, 0.3, 0.1, -0.2, 0.4, 0.3)$, for $n = 9$.