DIRECTGO: A New DIRECT-Type MATLAB Toolbox for Derivative-Free Global Optimization

LINAS STRIPINIS and REMIGIJUS PAULAVIČIUS, Vilnius University Institute of Data Science and Digital Technologies

In this work, we introduce DIRECTGO, a new MATLAB toolbox for derivative-free global optimization. DIRECTGO 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 ways: using static and dynamic data structures for more efficient information storage and organization. Furthermore, parallel schemes are applied to some promising algorithms within DIRECTGO. The toolbox is equipped with a graphical user interface (GUI), ensuring the user-friendly use of all functionalities available in DIRECTGO. Available features are demonstrated in detailed computational studies using a comprehensive DIRECTGOlib v1.0 library of global optimization test problems. Additionally, 11 classical engineering design problems illustrate the potential of DIRECTGO to solve challenging real-world problems. Finally, the appendix gives examples of accompanying MATLAB programs and provides a synopsis of its use on the test problems with box and general constraints.

CCS Concepts: • Computing methodologies → Parallel algorithms; • Mathematics of computing → Solvers; • Software and its engineering → Operational analysis; • Theory of computation → Continuous optimization; Numeric approximation algorithms; • Applied computing → Operations research;

Additional Key Words and Phrases: Global optimization, derivative-free optimization, DIRECT-type algorithms, benchmarking, optimization software, MATLAB, TOMLAB

ACM Reference format:
Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGO: A New DIRECT-Type MATLAB Toolbox for Derivative-Free Global Optimization. ACM Trans. Math. Softw. 48, 4, Article 41 (December 2022), 46 pages. https://doi.org/10.1145/3559755

1 INTRODUCTION
The DIRECT (DIviding RECTangles) algorithm [39] is a well-known and widely used solution technique for derivative-free global optimization problems. The DIRECT algorithm extends classical Lipschitz optimization [62–64, 68, 71, 72, 78, 81], where the Lipschitz constant is not assumed to be known. This property makes DIRECT-type methods especially attractive for solving various real-world optimization problems (see, e.g., [2, 3, 9, 14, 16, 25, 50, 61, 66, 89] and

Linas Stripinis and Remigijus Paulavičius contributed equally to this research.
Authors’ address: L. Stripinis and R. Paulavičius, Vilnius University Institute of Data Science and Digital Technologies, Akademijos 4, Vilnius LT-08663, Lithuania; emails: [linas.stripinis, remigijus.paulavicius]@mif.vu.lt.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
0098-3500/2022/12-ART41 $15.00
https://doi.org/10.1145/3559755

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
Moreover, a recent review and comparison in [75] 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 [34], Simulated annealing [42], and Particle swarm optimization [41].

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., [19, 22, 25, 37, 39, 48–51, 66, 76] and the references given therein);
- Linearly constrained/symmetric (see, e.g., [27, 60, 65–67] and the references given therein);
- Generally constrained (see, e.g., [13, 21, 37, 44, 89] and the references given therein);
- Containing hidden constraints (see, e.g., [24, 58, 84] and the references given therein).

MATLAB [53] is one of the most broadly used mathematical computing environments in scientific and technical computing (see, e.g., [7, 33, 96]). Many widely used implementations of the original DIRECT algorithm (see, e.g., [6, 20, 24]) as well as various later introduced DIRECT-type extensions (see, e.g., [44, 47, 48, 66]), were developed using MATLAB. Motivated by this, we developed a DIRECT-type global optimization toolbox (DIRECTGO) within the MATLAB environment. This article presents the first version of the toolbox DIRECTGO v1.0.0 [85]. However, the toolbox will be continuously updated with new DIRECT-type algorithms, resulting in new releases.

The DIRECTGO toolbox is equipped with a graphical user interface (GUI), which links to a DIRECTGOLib v1.0 [86, 92] library and ensures the user-friendly use of all functionalities available in DIRECTGO. The DIRECTGOLib v1.0 library is a continuation of our previous DIRECTLib [90], which was widely used in our different previous studies (see, e.g., [88, 89, 93]). However, DIRECTLib was designed as a static library and did not offer the global optimization community comfortable opportunities to contribute to it. Therefore, a new DIRECTGOLib v1.0 is designed as an open-source GitHub repository to which other researchers can easily contribute.

The first publicly available DIRECT implementations and many others introduced later typically use static data structures for storage and organization. Our recent work [93] showed that the MATLAB implementation of the same DIRECT-type algorithm based on dynamic data structure often has a significant advantage over implementation based on the static data structure.

Therefore, each algorithm in DIRECTGO is implemented using both static and dynamic data structures. As various applications can benefit from parallel computing, the Single Program Multiple Data (SPMD) parallel scheme (see [93] for more information) is used to implement some approaches.

1.1 Contributions and Structure

We summarize our main contributions below:

- We develop a new MATLAB toolbox (DIRECTGO) 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 [30, 93].
- We adapt the SPMD parallel scheme [93] for selected DIRECT-type algorithms.
- We create a new library (DIRECTGOLib v1.0 [92]) of test and engineering global optimization problems for usage with DIRECTGO and convenient contribution to it through GitHub [86].
- We perform a comprehensive experimental study on the effectiveness of various DIRECT-type approaches.

MATLAB [53] is one of the most broadly used mathematical computing environments in scientific and technical computing (see, e.g., [7, 33, 96]). Many widely used implementations of the original DIRECT algorithm (see, e.g., [6, 20, 24]) as well as various later introduced DIRECT-type extensions (see, e.g., [44, 47, 48, 66]), were developed using MATLAB. Motivated by this, we developed a DIRECT-type global optimization toolbox (DIRECTGO) within the MATLAB environment. This article presents the first version of the toolbox DIRECTGO v1.0.0 [85]. However, the toolbox will be continuously updated with new DIRECT-type algorithms, resulting in new releases.

The DIRECTGO toolbox is equipped with a graphical user interface (GUI), which links to a DIRECTGOLib v1.0 [86, 92] library and ensures the user-friendly use of all functionalities available in DIRECTGO. The DIRECTGOLib v1.0 library is a continuation of our previous DIRECTLib [90], which was widely used in our different previous studies (see, e.g., [88, 89, 93]). However, DIRECTLib was designed as a static library and did not offer the global optimization community comfortable opportunities to contribute to it. Therefore, a new DIRECTGOLib v1.0 is designed as an open-source GitHub repository to which other researchers can easily contribute.

The first publicly available DIRECT implementations and many others introduced later typically use static data structures for storage and organization. Our recent work [93] showed that the MATLAB implementation of the same DIRECT-type algorithm based on dynamic data structure often has a significant advantage over implementation based on the static data structure.

Therefore, each algorithm in DIRECTGO is implemented using both static and dynamic data structures. As various applications can benefit from parallel computing, the Single Program Multiple Data (SPMD) parallel scheme (see [93] for more information) is used to implement some approaches.

1.1 Contributions and Structure

We summarize our main contributions below:

- We develop a new MATLAB toolbox (DIRECTGO) 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 [30, 93].
- We adapt the SPMD parallel scheme [93] for selected DIRECT-type algorithms.
- We create a new library (DIRECTGOLib v1.0 [92]) of test and engineering global optimization problems for usage with DIRECTGO and convenient contribution to it through GitHub [86].
- We perform a comprehensive experimental study on the effectiveness of various DIRECT-type approaches.

MATLAB [53] is one of the most broadly used mathematical computing environments in scientific and technical computing (see, e.g., [7, 33, 96]). Many widely used implementations of the original DIRECT algorithm (see, e.g., [6, 20, 24]) as well as various later introduced DIRECT-type extensions (see, e.g., [44, 47, 48, 66]), were developed using MATLAB. Motivated by this, we developed a DIRECT-type global optimization toolbox (DIRECTGO) within the MATLAB environment. This article presents the first version of the toolbox DIRECTGO v1.0.0 [85]. However, the toolbox will be continuously updated with new DIRECT-type algorithms, resulting in new releases.

The DIRECTGO toolbox is equipped with a graphical user interface (GUI), which links to a DIRECTGOLib v1.0 [86, 92] library and ensures the user-friendly use of all functionalities available in DIRECTGO. The DIRECTGOLib v1.0 library is a continuation of our previous DIRECTLib [90], which was widely used in our different previous studies (see, e.g., [88, 89, 93]). However, DIRECTLib was designed as a static library and did not offer the global optimization community comfortable opportunities to contribute to it. Therefore, a new DIRECTGOLib v1.0 is designed as an open-source GitHub repository to which other researchers can easily contribute.

The first publicly available DIRECT implementations and many others introduced later typically use static data structures for storage and organization. Our recent work [93] showed that the MATLAB implementation of the same DIRECT-type algorithm based on dynamic data structure often has a significant advantage over implementation based on the static data structure.

Therefore, each algorithm in DIRECTGO is implemented using both static and dynamic data structures. As various applications can benefit from parallel computing, the Single Program Multiple Data (SPMD) parallel scheme (see [93] for more information) is used to implement some approaches.
Table 1. Classification of DIRECT-Type Implementations (Within the DIRECTGO Toolbox) Based on the Type of Constraints

| Problem type | Algorithm name | Implementation | Description and References |
|--------------|----------------|----------------|-----------------------------|
| Box constrained | glibSolve | ++ + | Finkel’s implementation [28] of the original DIRECT [39] algorithm |
| Linearly constrained | Lc-DISIMPL-C, Lc-DISIMPL-V | ++ | Our implementation of algorithms from [46, 67] (based on Björkman’s glibSolve [6] implementation) |

- We design a user-friendly application with a GUI.
- We make DIRECTGO open-source, i.e., freely available to anyone [85].

The rest of this article is organized as follows. Section 2 provides the classification of existing DIRECT-type algorithms and describes the algorithms implemented within our toolbox in more detail. The parallel scheme used in implementing some algorithms is also discussed here. DIRECTGO toolbox is introduced and described in Section 3. The detailed computational study of the DIRECTGO toolbox using classical global optimization test and engineering design problems DIRECTGOLib v1.0 [92] 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 the algorithms implemented within our DIRECTGO toolbox in more detail. For a thorough review, we refer to a recent survey [38].

The derivative-free DIRECT algorithm [39] is an efficient deterministic technique to solve global optimization [36, 79, 94] problems subject to simple box constraints

$$\min_{x \in D} f(x),$$

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

The DIRECT algorithm includes three main steps: selection, sampling, and partitioning (subdivision). At the initial iteration, the DIRECT algorithm normalizes the feasible region $D$ to be the unit hyper-cube $D$ and refers to the 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 \( c_1 \in \bar{D} \) (see the left panel of Figure 1). Then, DIRECT selects \( \bar{D} \) and samples at \( c_1 \pm \delta e_j, j = 1, \ldots, n \), where \( e_j \) is the \( j \)th unit vector and \( \delta \) is equal to one-third of the maximum side length of \( \bar{D} \). The subdivision procedure in DIRECT is based on \( n \)-dimensional trisection along all longest dimensions (sides). When several dimensions have the maximum side length, DIRECT starts trisection from the dimension with the lowest \( w_j \) and continues to the highest \([38, 39]\).

Here, \( w_j \) is defined as the best function values sampled along dimension \( j \)

\[
w_j = \min\{f(c_i + \delta_i e^j), f(c_i - \delta_i e^j)\},
\]

where \( c_i \) is the center of the hyper-rectangle \( \bar{D}_i \), and \( j \in M - \text{set of dimensions with the maximum side length} \).

Figure 1 illustrates the DIRECT algorithm’s selection, sampling, and subdivision (trisection) for a two-dimensional Rosenbrock test function.

The selection procedure at the initial step is trivial as we have only one candidate \( \bar{D} \). To formalize the selection of potentially optimal hyper-rectangles (POHs) in the future iterations, we define the current partition at the iteration \( k \)

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

where \( \bar{D}^k_i = [a^i_j, b^i_j] = \{x \in \mathbb{R}^n : 0 \leq a^i_j \leq x^j \leq b^i_j \leq 1, j = 1, \ldots, n, \forall i \in \mathbb{I}^k\} \) 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 POHs from the current partition \( \mathcal{P}^k \). DIRECT assesses the potentiality based on the lower bound estimates for the objective function \( f \) over each hyper-rectangle \( \bar{D}^k_i \) as stated in Definition 2.1.

**Definition 2.1 (Potentially Optimal Hyper-rectangle).** Let \( c^k_i \) denote the center sampling point and \( \delta^k_i \) be a measure of the hyper-rectangle \( \bar{D}^k_i \). Let \( \epsilon > 0 \) be a positive constant and \( f_{\min} \) be the best currently found value of the objective function. A hyper-rectangle \( \bar{D}^k_j, j \in \mathbb{I}^k \) is said to be potentially optimal if there exists some rate-of-change (Lipschitz) constant \( \tilde{L} > 0 \) such that

\[
\begin{align*}
f(c^k_j) - \tilde{L}\delta^k_j &\leq f(c^k_i) - \tilde{L}\delta^k_i, \quad \forall i \in \mathbb{I}^k, \\
f(c^k_j) - \tilde{L}\delta^k_j &\leq f_{\min} - \epsilon|f_{\min}|,
\end{align*}
\]

![Fig. 1. Illustration of selection, sampling, and subdivision (trisection) used in the original DIRECT algorithm [39] on a two-dimensional Rosenbrock test function in the first two iterations.](image-url)
where the measure of the hyper-rectangle $D_i^k$ is

$$\delta^k_i = \frac{1}{2} \|b_i^k - a_i^k\|_2.$$  \hspace{1cm} (5)

The hyper-rectangle $D_i^k$ is potentially optimal if the lower Lipschitz bound for the objective function computed by the left-hand side of (3) is the smallest one with some positive constant $\tilde{L}$ among the hyper-rectangles of the current partition $P_k$. In (4), the parameter $\epsilon$ is used to protect from an excessive refinement of the local minima $[39, 60]$. Authors obtained good results for $\epsilon$ values ranging from $10^{-3}$ to $10^{-7}$ in [39].

A geometric interpretation of the selection procedure is given on the right panel of Figure 2. Here, each hyper-rectangle is represented as a point. The $x$-axis shows the measure ($\delta^k_i$) while the $y$-axis—the objective function value attained at the midpoint ($c_i^k$) of a certain hyper-rectangle. The hyper-rectangles meeting conditions (3) and (4) are points on the lower-right convex hull (highlighted in blue color). Condition (4) prevents wasting function evaluations on tiny hyper-rectangles where only a negligible improvement can be expected.

Then at 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, summarized in Algorithm 1.

### 2.1 DIRECT-Type Algorithms for Box-Constrained Global Optimization

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 DIRECTGO 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 DIRECTGO toolbox.
Table 2. Summary of Main Characteristics of DIRECT-Type Algorithms for Box-Constrained Global Optimization

| Alg. | Step | Partitioning scheme | Selection of POH & comments on additional steps, if any | Input param. |
|------|------|---------------------|--------------------------------------------------------|--------------|
| Aggressive DIRECT | | | Relaxed criteria of POH selection. At each iteration, the algorithm selects and divides the best hyper-rectangles of each size $(\delta_k^b)$ | No input parameters |
| DIRECT-G | | | Uses enhanced global selection (Definition 2 in [88]). | |
| DIRECT-L | | | Uses enhanced local selection (Definition 3 in [88]). | |
| DIRECT-GL | | | Uses the unique union of two sets obtained using enhanced global and local selection (Definitions 2 and 3 [88]). | |
| DIRECT | | | | |
| DIRECT-restart | | | Uses two different $\varepsilon$ values during the selection: 0 if there is an improvement in the solution, and 0.01 otherwise. | |
| DIRECT-m | | | Uses the median value $f_{\text{median}}$ in (6). | |
| DIRECT-1 | | | Uses the infinity norm in (5) and selects at most one POH from each group having the same measure $(\delta_k^b)$. Additional minimization procedure $f_{\text{mincon}}$ is employed. | |
| DIRECT-a | | | Uses the average value $f_{\text{average}}$ in Equation (7). | |
| DIRMN | | | $f_{\text{mincon}}$ is performed from each selected POH. | |
| PLOR | | | The set of POH is reduced to just two: with the maximal $(\delta_{k_{\text{max}}})$ and the minimal $(\delta_{k_{\text{min}}})$ measures. | |
| glbSolve | | | Discards unnecessary hyper-rectangles for symmetric functions. | |
| glbSolve-sym2 | | | Performs the selection of POH on three different sets (‘levels’). MrDIRECTt uses different $\varepsilon$ values at each level. | |
| MrDIRECT | | | Uses an adaptive scheme for balancing the local and global search. | |
| MrDIRECT075 | | | Local minimization procedure $f_{\text{mincon}}$ is embedded into the BIRMIN algorithm. | |
| Gb-glSolve | | | Uses an adaptive scheme for balancing the local and global search. | |
| Gb-BIRECT | | | Uses the function evaluate at the two points on the convex hull and at a point on the extended version of the hypersimplices [76] algorithm. | |
| BIRMIN | | | Uses an adaptive scheme for balancing the local and global search. | |
| BIRECT | | | Uses an adaptive scheme for balancing the local and global search. | |
| ADC | | | | |
| DISIMPL-C | | | Uses the function evaluate at the two points of the main diagonal, as shown in Figure 3. Notice that up to $2^n$ hyper-rectangles can share the same vertex, leading (in a long sequence) to a smaller number of sampled points than the total number of hyper-rectangles. | |
| Gb-DISIMPL-C | | | | |
| Gb-DISIMPL-V | | | | |
| DISIMPL-V | | | | |
**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 \( c_1 \). Set \( f_{\text{min}} = f(c_1) \), \( c_{\text{min}} = c_1 \). Initialize algorithmic performance measures, and stopping criteria.

2. while stopping criteria are not satisfied do
   3. **Selection.** Identify the sets \( S \) of POHs (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 \( f_{\text{min}}, c_{\text{min}}, \) and algorithmic performance measures.
   5. **Subdivision.** Each POH \( \bar{D}_j \in S \) subdivide (trisect) and update the partition \( \mathcal{P} \).

6. end

7. **Return** \( f_{\text{min}}, c_{\text{min}}, \) and performance measures.

in the current partition. Furthermore, as in the revised version of DIRECT [37], ADC trisects each selected POH along just one of the longest dimensions. Such a diagonal scheme potentially obtains more comprehensive information about the objective function than center-based sampling, which sometimes may take many iterations to find the solution. For example, a hyper-rectangle containing the optimum with a bad function value at the midpoint makes him undesirable for further selection. The ADC algorithm intuitively reduces this chance for both sampling points in the hyper-rectangle containing the optimum solution.

**BIIRECT (BIsecting RECTangles) [59]** is motivated by the diagonal partitioning strategy [76, 77, 79]. As the name suggests, the bisection is used instead of a trisection typical for DIRECT-type algorithms. In BIIRECT, the objective function is evaluated at two points on the diagonal equidistant between themselves and a diagonal’s vertices, as shown in Figure 4. Such a sampling strategy enables the reuse of sampling points in descendant hyper-rectangles. Moreover, as in the ADC case, using two-points-based diagonal sampling, potentially more comprehensive information about the objective function is considered than in the center-based sampling.

In DISIMPL [65], simplicial partitions are considered instead of hyper-rectangles. The hyper-cube \( \bar{D} \) is partitioned into \( n! \) simplices by the standard face-to-face simplicial division based on the combinatorial vertex triangulation at the first iteration. After this, all simplices share the diagonal of the feasible region and have equal hyper-volume. In [65], we proposed two different sampling strategies. Both are included in the DIRECTGO 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 total number of initial simplices grows speedily with the dimension increase. 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 \[65, 66\] and problems with linear constraints \[67\].

In the DIRECT-restart algorithm \[19\], the authors introduced an adaptive scheme for the $\epsilon$ parameter. Condition (4) 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 $\epsilon = 0$, and the same value for $\epsilon$ 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 $\epsilon = 0.01$ value to prevent an excessive local search. If the algorithm finds an improvement or fails to see the progress within 50 iterations in this phase, DIRECT-restart switches to $\epsilon = 0$. Then, if another 50 iterations pass without improvement, this may indicate that the global minimum has been found, and one should work on refining it to higher accuracy.

The authors of MrDIRECT \[49\] and MrDIRECT\_075 \[45\] algorithms introduced three different levels to perform the selection procedure:

- At level 2, DIRECT is run as usual, with $\epsilon = 10^{-5}$.
- At level 1, the selection is limited to only 90% of $\hat{D}_{k}^i \in P^k$; 10% of the largest hyper-rectangles are ignored. Here, $\epsilon = 10^{-7}$ is used.
- At level 0, the selection is limited to 10% of the largest hyper-rectangles (ignored at level 1) using $\epsilon = 0$.

Both algorithms cycle through these levels using “W-cycle”: 21011012. The main difference between the proposed algorithms is that MrDIRECT uses fixed $\epsilon = 10^{-4}$ value at all levels, while MrDIRECT\_075 follows above-mentioned rules.

In \[2\], the authors relaxed the selection criteria 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 $(\delta_i^k)$ having the lowest function value. Therefore, using Aggressive DIRECT in the situation presented in Figure 2, a hyper-rectangle with the slightest measure $\delta_i^k$ would also be selected and divided. The aggressive version performs
more function evaluations per iteration than 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 [28, 29, 31, 98]. Note that the authors did not specify which hyper-rectangle should be selected from the same group ($\delta_k^i$) if more than one with identical objective function values exist. Thus, the hyper-rectangle with the larger index value was selected in our implementations.

In [25], the algorithm named DIRECT-1 was proposed. In most DIRECT-type algorithms, the measure of the hyper-rectangle is calculated by a half-length of a diagonal (see (5)). In DIRECT-1, this measure is evaluated by the length of its longest side. This measure corresponds to the infinity norm and allows the DIRECT-1 algorithm to group more hyper-rectangles with the same measure. Thus, there are fewer different measures, so fewer POHs are selected. Moreover, with DIRECT-1, at most one hyper-rectangle is selected from each group, even if there is more than one POH in the same group. Such a strategy allows a reduction in the number of divisions within a group. Once again, the same rule is adapted [88] to determine which hyper-rectangle to select from several possible ones.

In [22], 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 objective function values are large enough. The authors proposed a scaling of function values by subtracting the median ($f_{\text{median}}$) of the collected function values to overcome this. DIRECT-m replaces the Equation (4) in Definition 2.1 to

$$f(c_j) - \tilde{L}\delta_j \leq f_{\text{min}} - \epsilon |f_{\text{min}} - f_{\text{median}}|. \quad (6)$$

Similarly, in [46], the authors extended the same idea in DIRECT-a to reduce the objective function’s additive scaling. Instead of the median value, the authors proposed to use the average value ($f_{\text{average}}$) at each iteration

$$f(c_j) - \tilde{L}\delta_j \leq f_{\text{min}} - \epsilon |f_{\text{min}} - f_{\text{average}}|. \quad (7)$$

Another extension of the DIRECT algorithm was proposed in [27]. 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. The algorithm determines which hyper-rectangles can be safely discarded, considering the problem’s symmetrical nature, and avoids exploration over equivalent subregions.

In the PLOR algorithm [56], the set of POHs is reduced to just two, corresponding to the first and last point on the Pareto front (see the right panel in Figure 2). Therefore, only hyper-rectangles with the lowest function value and the most extensive measure, breaking ties in favor of a better center-point function value, are selected.

Our recent extension, DIRECT-GL [88], introduced a new approach to identifying the extended set of POHs. 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 [39]). 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 POHs in each iteration twice [88], and the algorithm separately handles the found independent sets $G$ (using Definition 2 from [88]—DIRECT-G) and $L$ (using Definition 3 from [88]—DIRECT-L). Following the same trend from [93], the version used in this article
slightly differs compared to [88]. 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 to reduce the data communication between the computational units in the parallel algorithm version [83]. At the same time, we found that this way modified DIRECT-GL was, on average, more effective than the original one.

Several globally biased (Gb-) versions of DIRECT-type algorithms were introduced and investigated [60, 61]. Proposed approaches are primarily oriented for solving extremely difficult global 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. The proposed globally biased strategy is combined with glbSolve, BIRECT, DISIMPL-C, and DISIMPL-V algorithmic frameworks within our DIRECTGO toolbox.

Finally, three different hybridized DIRECT-type algorithms are proposed (DIRECT-rev [37], DIRMIN [50], BIRMIN [61]). In our implementation, all algorithms are combined with the same local search routine—fmincon. The DIRMIN algorithm suggests running a local search starting from the midpoint of every POH. However, such an approach likely generates more local searches than necessary, as many start 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 [37] 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 POH is selected if several equally good exist (the same measure and objective values) in Definition 2.1. We have applied the same rule from [88] to determine which hyper-rectangle to select from several ones when needed.

2.2 DIRECT-Type Algorithms for Generally Constrained Global Optimization

The original DIRECT algorithm [39] only solves optimization problems with the variables’ bounds. In this subsection, we consider a generally constrained global optimization problem of the form:

\[
\min_{x \in D} f(x) \\
\text{s.t. } g(x) \leq 0, \\
h(x) = 0,
\]

(8)

where \( f : \mathbb{R}^n \rightarrow \mathbb{R}, g : \mathbb{R}^n \rightarrow \mathbb{R}^m, h : \mathbb{R}^n \rightarrow \mathbb{R}^r \) are (possibly non-linear) continuous functions. The feasible region is a non-empty set, consisting of points that satisfy all constraints, i.e., \( D_{\text{feas}} = D \cap \Omega \neq \emptyset \), where \( \Omega = \{ x \in \mathbb{R}^n : g(x) \leq 0, h(x) = 0 \} \). 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 non-linear, non-differentiable, non-convex, and multi-modal.

The first DIRECT-type algorithm for problems with general constraints was introduced in [37]. Finkel in [21] 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., [4, 13, 44, 69, 70, 89] and the references given therein). Below we briefly review approaches implemented in the current release of the DIRECTGO toolbox (see Table 1).
An exact L1 penalty approach DIRECT-L1 [20] is transforming the original constrained problem (8) in the form:
\[
\min_{x \in D} f(x) + \sum_{i=1}^{m} \max\{\gamma_i g_i(x), 0\} + \sum_{i=1}^{r} \gamma_{i+m} |h_i(x)|,
\]
where \(\gamma_i\) are penalty parameters. Experiments in [21] 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 algorithm’s performance [21, 44, 66, 67, 89].

In [89], we have introduced a new DIRECT-type extension based on the DIRECT-GL [88] algorithm. The new DIRECT-GLce algorithm uses an auxiliary function approach that combines objective and constraint functions and does not require 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.:
\[
\min_{x \in D} \phi(x),
\]
where
\[
\phi(x) = \sum_{i=1}^{m} \max\{g_i(x), 0\} + \sum_{i=1}^{r} |h_i(x)|.
\]
The algorithm works in this phase until at least one feasible point \((x \in D_{\varepsilon_{\phi}}^{\text{feas}}}) is found, where
\[
D_{\varepsilon_{\phi}}^{\text{feas}} = \{x : 0 \leq \phi(x) \leq \varepsilon_{\phi}, x \in D\}.
\]
The \(\varepsilon_{\phi}\) is a small user-specified tolerance for the sum of constraint functions Equation (11). When feasible points are located, the effort is switched to improve the feasible solutions. In the second phase, DIRECT-GLce uses the transformed problem (8):
\[
\min_{x \in D} f(x) + \tilde{\xi}(x, f_{\text{feas}}^{\text{min}}),
\]
\[
\tilde{\xi}(x, f_{\text{feats}}^{\text{min}}) = \begin{cases} 
0, & x \in D_{\varepsilon_{\phi}}^{\text{feas}} \\
0, & x \in D_{\varepsilon_{\text{cons}}^{\text{inf}}}^{\text{inf}} \\
\phi(x) + \Delta, & \text{otherwise},
\end{cases}
\]
where
\[
D_{\varepsilon_{\text{cons}}^{\text{inf}}} = \{x : f(x) \leq f_{\text{feas}}^{\text{min}}, \varepsilon_{\phi} < \phi(x) \leq \varepsilon_{\text{cons}}, x \in D\},
\]
and \(\varepsilon_{\text{cons}}\) is a small tolerance for constraint function sum, which automatically varies during the optimization process. An auxiliary function \(\xi(x, f_{\text{feas}}^{\text{min}})\) depends on the sum of the constraint functions and the parameter \(\Delta = |f(x) - f_{\text{feas}}^{\text{min}}|\), equal to the absolute difference between the best feasible function value found so far \(f_{\text{feas}}^{\text{min}}\) and the objective value at an infeasible center point. The purpose of the parameter \(\Delta\) is to forbid the convergence to infeasible regions by penalizing the objective value at infeasible points. In such a way, the formulation (13) does not require any penalty parameters and determines the convergence of the algorithm to a feasible solution. The value of \(\xi(x, f_{\text{feas}}^{\text{min}})\) is updated when a smaller value of \(f_{\text{feas}}^{\text{min}}\) 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 (13) minimizes the transformed problem:

$$\min_{x \in D} f(x) + \xi(x, f_{\text{feas}}),$$

$$\xi(x, f_{\text{feas}}) = \begin{cases} 
0, & x \in D_{\text{feas}} \\
\phi(x) + \Delta, & \text{otherwise.}
\end{cases}$$

Experimental investigation in [89] showed that the algorithm has the most wins in this comparison and can solve about 50% of the problems with the highest efficiency. Unfortunately, the DIRECT-GLc algorithm’s efficiency decreases, solving more challenging problems (with non-linear constraints and \(n \geq 4\)), where DIRECT-GLce is significantly better. Therefore, 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 [89], we observed that 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. In this section, we consider optimization problems with only linear constraints.

In [67], we have extended the original simplicial partitioning-based DISIMPL algorithm [65, 66] 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 very subtly. In such a way, the new algorithms (Lc-DISIMPL-C and Lc-DISIMPL-V) [67] perform the search only in the feasible region, in contrast to other DIRECT-type approaches. Nevertheless, the authors in [67] 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

Optimization problems with hidden constraints often occur when the objective function is not defined everywhere [10]. Typical examples of such situations are the simulation crashes [17] and failure of computations within the objective function [9, 11, 15, 82]. As in [10, 17], we call these internal to \(f\) constraints “hidden constraints” and assume that \(f\) fails to return a value when evaluated at \(x \notin D_{\text{feas}}\). Some authors alternatively may use other terms like “crash,” “unknown,” “unspecified,” and “forgotten” constraints [1, 17].

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

$$\min_{x \in D_{\text{feas}}} f(x),$$

where \(f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}\) denotes an extended real-valued, most likely “black-box” objective function. A priori an unknown feasible region \(D_{\text{feas}}\) is defined as a non-empty set

$$D_{\text{feas}} = D \setminus D_{\text{hidden}} \neq \emptyset,$$

and \(D_{\text{hidden}}\) are not given by explicit formulae hidden constraints. Such a problem formulation leads to a complex and analytically undefined feasible region. Hidden constraints are typically handled by returning NaN or \(\infty\) evaluating the objective function at \(x \notin D_{\text{feas}}\). Therefore, using
NaN hyper-rectangles with an infeasible center point would not be selected as potential optimal at all [84]. In the case of $\infty$ (or any other high value), they would be left unexplored as long as there are the same size hyper-rectangles with feasible centers [21, 84]. Unfortunately, most DIRECT-type algorithms cannot be directly (without any modifications) applied for the problem’s (16) solution.

In the current version of the DIRECTGO toolbox, four such algorithms are available (see Table 1). One of the first proposed modifications for such problems was the barrier method (DIRECT-Barrier) [24]. The DIRECT-Barrier 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 [21, 66, 89]. 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 [21] 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 (16).

The second DIRECT-type approach for hidden constraints is based on Neighbourhood Assignment Strategy (NAS) [24]. DIRECT-NAS’s main idea is to assign the value at infeasible point $\mathbf{x}^{inf} \notin D^{feas}$ relative to the objective values attained in the feasible points from the neighborhood of $\mathbf{x}^{inf}$. 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 more than one feasible center point inside the enlarged region, DIRECT-NAS assigns the smallest function value to the infeasible midpoint plus a small epsilon $f(\mathbf{x}^{feas}) + \epsilon f(\mathbf{x}^{feas})$, where $\epsilon = 10^{-6}$ was proposed to use. If inside the enlarged region has no feasible points, DIRECT-NAS assigns the largest objective function value found so far $f_{max} + \lambda$, where $\lambda = 1$ was proposed to use. This strategy does not allow the DIRECT-NAS algorithm to move beyond the feasible region by penalizing infeasible midpoints with large values. However, the algorithm’s principal concern is the slow convergence 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 [58] incorporates the previously mentioned barrier approach techniques. 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 depends on when (how often) the subdividing step is performed. Therefore, new subdivisions can grow drastically, especially for higher dimensionality problems.

The most recent version for hidden constraints DIRECT-GLh [84] is based on our previous DIRECT-GL [88] 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 [88]. In such a way, DIRECT-GLh does not penalize infeasible hyper-rectangles with large values (as was suggested by previous proposals), which are close to the $\mathbf{x}_{min}$ and assure a faster and more comprehensive examination of hidden regions. Moreover, this approach
employs additional procedures to efficiently handle infeasible initial points (see [84] for experimental justification).

2.4 Implementation of DIRECT-Type Algorithms Within DIRECTGO

2.4.1 Sequential Implementation of the Algorithms. The performance of DIRECT-type algorithms highly depends on computer implementation. Most publicly available DIRECT implementations (see, e.g., DIRECT v4.0 [20] and glbSolve [6]) use static data memory management [6, 20, 24]. In the “Implementation” column of Table 1, we provide information on which data structures were used in our implementations and whether a particular algorithm was implemented in parallel. Below we look at the main advantages and disadvantages of each of them.

With static data management, all information received after the domain partitioning is stored in the contiguous memory blocks. This includes objective and constraint function values, index numbers, center point coordinates, side lengths of hyper-rectangles, and so on. Such implementation can quickly access the elements for further selection, sampling, and subdivision steps. An apparent drawback of the static data structure is unpredictable memory demand due to different characteristics of the optimization problems. Thus many DIRECT-type algorithmic implementations use large static arrays to store the current state of the space partitioning. If any array is insufficient to store the required information, this can lead to code failure.

Another disadvantage of static data structures used in DIRECT implementations is that they require unnecessary recalculations in each iteration. One of the essential tasks in the DIRECT-type algorithms is the selection step. This step requires sorting all existing hyper-rectangles by the same size of diameter. Such sorting becomes especially inefficient when the optimization process is longer and the amount of data gets large, e.g., for higher dimensionality problems or when a solution with high accuracy is required.

In [30], the authors proposed using dynamic data structures. Information received after space partitioning is sorted by hyper-rectangle diameters and stored in columns. All rectangles of the same diameter are stored in the column in any order. In [30], the authors mentioned the idea of sorting columns by function values in descending order or inserting all new data in sorted sequences separately. However, any of these ideas have not been investigated further. With dynamic data structures, the selection step is much more efficient. It can be performed only in the set consisting of the best function values from each column. Such implementation saves lots of time compared with the static data structure-based implementation.

In [93], we have compared two different implementations (static and dynamic) of the same DIRECT-GLce algorithm. The dynamic implementation of the code required, on average, 62% less total execution time than static-based. The difference was even more significant when the number of function evaluations was high.

One of the apparent drawbacks of the dynamic data structure is unpredictable columns size. Fully processed POHs must be removed from the previous columns and added to a new/existing column. During the algorithm’s execution, there can be many hyper-rectangle diameters. Depending on the dimension of the problem, usually, the initial array is allocated of reasonably large size. If the array provides insufficient size, new blocks of columns will be reallocated as needed. In practice, only a few of these columns need reallocation at any given time.

2.4.2 Parallel Implementations of the Algorithms. We use the MathWorks official extension to the MATLAB language—the Parallel Computing Toolbox [53] for parallel implementations. The Parallel Computing Toolbox provides several parallel programming paradigms [52], like threads, parallel for-loops, and SPMD. In [93], we concluded that the SPMD-based parallel implementation
Fig. 5. Flowchart diagram for the parallel implementations of selected DIRECT-type algorithms.

of the DIRECT-GLce is the most efficient and significantly outperforms the other two based on parfor-loops.

Therefore, in the DIRECTGO toolbox, parallel implementations are based on the SPMD functionality within the Parallel Computing Toolbox, used to allocate the work across multiple labs in the MATLAB software environment. Each lab stores information on its main memory block, and data are exchanged through the message passing over the interconnection network [53]. The master-slave paradigm is used to implement dynamic load balancing. The flowchart of the parallel algorithmic framework is illustrated in Figure 5. One lab is the master, denoted by lab_1, and the other labs are slaves lab_i, i = 2, . . . , p. The master also acts as a slave. Each iteration must be done in a sequence to preserve the determinism.

The master performs the following tasks:

- **The initialization step**: normalizes the domain (D) and evaluates the objective and constraint functions at the center point. Here, only the optimization problem and the information about the domain D are shared with slaves lab_i, i = 2, . . . , p.
- **At each iteration**:
  - checks the stopping conditions and informs the slaves if any of them have been met;
  - finds the full set of POHs by performing a selection step considering the combined set of local POHs;
  - splits the full set of POHs among all slaves and itself equally;
  - gives instructions to the slaves having an excess of POHs (in their local memory) to share them with those who have a deficit, including itself;
  - sends or receives POHs according to its instructions;
  - performs the sampling, subdivision, and local selection steps as the slave;
  - receives from slaves the information about their local POHs sets.
The slaves perform the following tasks:

- **At each iteration:**
  - send or receive POHs, according to the master’s instructions;
  - perform the sampling and subdivision steps sequentially;
  - perform the selection using the information in their local memory and send local POHs to the master;
  - terminate when such an instruction from the master is received.

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 \( (lab_i, i = 1, \ldots, \rho) \) receive tasks from the master lab, each sequentially performs the sampling and subdivision steps. Then finds a local set of POHs and sends local data back to the master lab for the further global selection step. After this, each slave 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 [93] for a more detailed description and analysis of parallel schemes.

We should note that not all DIRECT-type implementations can use the latter scheme of parallelism. For example, implementations using the `conhull` function, which returns all the points on a convex hull, cannot. To preserve the determinism, only the master should select POHs and have all data stored in its memory. The framework shown in Figure 5 is inappropriate, and a new one should be developed. The DIRECT-NAS algorithm has an additional expensive constraint handling step not addressed in the proposed parallel scheme. As summarized in Table 1, currently, 17 DIRECT-type algorithms are implemented in parallel within the DIRECTGO toolbox.

### 3 DIRECTGO TOOLBOX

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

- **DIRECTGO.mltbx** - MATLAB toolbox package containing implementations of DIRECT-type algorithms (from Table 1), including an extensive `DIRECTGOLib v1.0` library of the box and generally constrained test and practical engineering global optimization problems, often used for benchmarking DIRECT-type algorithms.

- **DIRECTGO.mlappinstall** - A single MATLAB app installer containing everything necessary to install and run the DIRECTGO toolbox, including a GUI.

#### 3.1 Graphical User Interface

After installation (using `DIRECTGO.mlappinstall`), DIRECTGO can be launched from MATLAB APPS, located in the toolbar. The graphical interface of the main DIRECTGO toolbox window is shown in Figure 6. Application is divided into three main parts: (i) selection of the problem’s type from `DIRECTGOLib v1.0`; (ii) setting up an optimization problem and algorithmic options; (iii) selection of DIRECT-type algorithm, his implementation, and convergence plot of obtained results.

The first step is to specify the objective and constraint functions, loading them from the integrated `DIRECTGOLib v1.0` library or selecting them from other sources. Examples of the structure needed are present. All test problems from the `DIRECTGOLib v1.0` library have up to
three key features: (i) known globally optimal solutions, (ii) a complete description of the problem, including objective and constraint functions (if any), and (iii) problem visualization (only for two-dimensional problems).

After selecting the optimization problem, the second step is to set up the bound constraints for each variable. First, 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, 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 **DIRECTGO.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:

1. \( f_{\text{min}} = \text{algorithm}(P); \)
2. \( f_{\text{min}} = \text{algorithm}(P, \text{OPTS}); \)
3. \( f_{\text{min}} = \text{algorithm}(P, \text{OPTS}, D); \)
4. \( [f_{\text{min}}, x_{\text{min}}] = \text{algorithm}(P, \text{OPTS}, D); \)
5. \( [f_{\text{min}}, x_{\text{min}}, \text{history}] = \text{algorithm}(P, \text{OPTS}, D); \)

The left side of the equations specifies the output parameters. After the termination, the algorithm returns the best objective value (\( f_{\text{min}} \)), solution point (\( x_{\text{min}} \)), 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 minimum value, and execution time.
The algorithm name (algorithm) and at least one input parameter are needed to specify on the right side. 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 parameter (OPTS) customizes the default algorithmic settings. The third parameter (D) is used to specify the bound constraints for each variable (see Equation (1)).

4 EXPERIMENTAL INVESTIGATION OF DIRECT-TYPE ALGORITHMS IN THE DIRECTGO TOOLBOX

This section presents the performance evaluation of DIRECT-type algorithms on test and engineering design problems from the DIRECTGOLib v1.0 [86, 92]. The DIRECTGOLib v1.0 library consists of the box and generally constrained test and practical engineering global optimization problems for various DIRECT-type algorithms benchmarking. Experimental results presented in this section are also available in digital form in the Results/TOMS directory of the Github repository (https://github.com/blockchain-group/DIRECTGO [85]). The most recent version of DIRECTGOLib v1.2 [87, 91] has additional test functions introduced in other parallel studies but has not been considered in this article. Despite this, our vision is to develop DIRECTGOLib further so that all algorithms can use the latest version of DIRECTGOLib without any additional preparations.

We distinguish the following classes (types) of global optimization problems:

- **BC** – Box-Constrained problems;
- **LC** – Linearly Constrained problems;
- **GC** – Generally Constrained problems.

A summary of all optimization problems in DIRECTGOLib v1.0 and their properties is given in Appendix A, Table 13. We note that some test problems have several variants, e.g., Bohachevsky, Shekel, and some of them, like Alpine, Csendes, and Griewank, can be used by changing the problem’s dimensionality. We used the following dimensions in our experimental setting: \( n = 2, 5, 10, 15 \). For some test problems, the second dimension \( (n = 2) \) was skipped because the problem was then too easy to solve, and sometimes we skipped \( n = 15 \) because the resulting problem was too hard that none of the algorithms were able to solve it. The fourth column in Table 13 indicates the exact dimensions used for all test problems.

All computations were carried out on a 6-core computer with 8th Generation Intel R Core™ 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 \( x \) with whom the percent error

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

is smaller than the tolerance value \( \varepsilon_{pe} \), i.e., \( pe \leq \varepsilon_{pe} \). Additionally, we stopped the tested algorithms when the number of function evaluations exceeded the prescribed maximal limit (equal to \( 2 \times 10^6 \)) or took more than 43,000.00 seconds. In any of these situations, the final result is set to \( 2 \times 10^6 \) to further process results. Two different values for \( \varepsilon_{pe} \) were considered: \( 10^{-2}, 10^{-8} \). By default,
algorithms were tested using the $\epsilon_{pe} = 10^{-2}$ value. Algorithms incorporating additional schemes to speed up the solution’s refinement have been tested using the $\epsilon_{pe} = 10^{-8}$ value.

Additionally, we analyze and compare the algorithms’ performance by applying the data profiles [57] to the convergence test (17). The data profile is a popular and widely used tool for benchmarking and evaluating the performance of several algorithms (solvers) when run on a large problem set. Benchmark results are generated by running a certain algorithm $v$ (from a set of algorithms $V$ under consideration) for each problem $u$ from a benchmark set $\mathcal{U}$ and recording the performance measure of interest. The performance measure could be, for example, the number of function evaluations, the computation time, the number of iterations, or the memory used. We used a number of function evaluations and the execution (computation) time criteria.

The data profiles provide the percentage of problems that can be solved with a given budget of the desired performance measure. The data profile is defined

$$\lambda_v(\alpha) = \frac{1}{\text{card}(\mathcal{U})} \text{size } \{ u \in \mathcal{U} : t_{u,v} \leq \alpha \},$$

where $t_{u,v} > 0$ is the number of performance measure required to solve problem $u$ by the algorithm $v$, and $\text{card}(\mathcal{U})$ is the cardinality of $\mathcal{U}$. In our case, the $\lambda_v(\alpha)$ shows the percentage of problems that can be solved within $\alpha$ function evaluations, or seconds.

In the experimental studies, the developed DIRECT-type algorithms (within the DIRECTGO toolbox) were compared among themselves and with three TOMLAB DIRECT-type solvers:

- TOMLAB/glbSolve [35] – implementation of the DIRECT algorithm [39];
- TOMLAB/glccSolve [35] – implementing an extended DIRECT version [37, 39] capable of handling linear and non-linear constrained problems;
- TOMLAB/glccCluster [35] – implementation of the DIRECT algorithm [39], hybridized with local search subroutine and clustering techniques.

We note that the TOMLAB/glcCluster algorithm has a large number and different input parameters that may significantly impact the algorithm’s performance. Even a parameter such as the maximum allowed number of function evaluations can significantly impact the algorithm’s performance. Our aim was not to find the optimal parameters values, as this is a complex process, but to investigate how these algorithms compare using the default values (provided by TOMLAB software developers). When the algorithm reaches the default limit for the maximum number of function evaluations ($M_{\text{max}}$) set in the default parameters, we restarted the algorithm using the final status from the previous run (“warm start” [35]) with doubled $2M_{\text{max}}$. In such a way, sometimes a default $M_{\text{max}}$ value was doubled up to our maximal limit of evaluations $2 \times 10^6$ was reached.

The Scripts/TOMS directory of the Github repository (https://github.com/blockchain-group/DIRECTGO) provides four different MATLAB scripts for cycling through all different classes of test problems used in this article. The constructed scripts can be handy for reproducing the results presented here, as well as for comparison and evaluation of newly developed algorithms.

### 4.1 Comparison of DIRECT-Type Algorithms for Box Constrained Optimization

Table 3 summarizes experimental results using $\epsilon_{pe} = 10^{-2}$. The smallest number of unsolved problems is achieved using DIRECT–GL (3/81). At the same time, the second, third, and fourth best algorithms are TOMLAB/glccCluster (4/81), BIRMIN (5/81), and DIRMIN (5/81), hybrid versions enriched with the local search subroutines. In column “Avg. # local searches”, we report the average number of local searches performed by each hybridized algorithm. Hybridization of the BIRMIN algorithm allows solving more problems compared to, e.g., the
| Algorithm | Avg. # local searches | Failed | Average results | Average results (n ≤ 4) | Average results (n ≥ 5) | Median results |
|-----------|----------------------|--------|----------------|-------------------------|------------------------|---------------|
| DIRECT    | 16/81                | 54.60  | 98.99          | 98.89                   | 98.91                  | 99.01         |
| DIRECT-restart | 23/81            | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| DIRECT-b  | 27/81                | 84.85  | 98.99          | 98.99                   | 98.91                  | 99.01         |
| DIRECT-I  | 28/81                | 84.85  | 98.99          | 98.99                   | 98.91                  | 99.01         |
| DIRECT-rev | 3/81                 | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| DIRECT-α  | 41/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| HOMIN     | 7/81                 | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| PLO        | 31/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GbBISOLVE | 23/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GbBISOLVE+ | 36/81               | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GbBISOLVE+α | 35/81          | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GB-DSIMPL-C | 4/81             | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GB-DSIMPL-V | 36/81             | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| DIRECT-I  | 13/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| BIRMIN    | 5/81                 | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| Gb-gBISOLVE | 25/81         | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GB-FSPC 6C | 4/81             | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| MB-DSIMPL-V | 36/81        | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| MB-DSIMPL-V | 36/81         | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| AO        | 30/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| AGressive DIRECT | 14/81      | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| DIRECT-L  | 16/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| DIRECT-GL  | 3/81                 | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GbBISOLVE+ | 24/81                | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| GbBISOLVE+α | 21/81            | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |
| TOMLAB/glcCluster | 21/81    | 56.85  | 97.99          | 97.99                   | 97.92                  | 97.97         |

* – a hybrid version of the algorithm, enriched with the local search subroutine.

globally biased version Gb-BIRECT (12/81). Among traditional DIRECT-type algorithms, the second and third best algorithms are BIRECT and DIRECT-G. Both methods failed to solve (9/81) and (10/81) test problems accordingly.

Furthermore, hybridization significantly reduces the total number of function evaluations (see Average results and Median results columns). The BIRECT and PLO were the most effective algorithms among the traditional algorithms based on the median number of function evaluations (see Median results column). However, for PLO, such performance needs to be interpreted correctly. As PLO 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 ≤ 4)), PLO performance is among the worst. PLO has failed on a larger number of simpler test problems than other approaches. In contrast, DIRECT-GL is only in eight place based on the median number of function evaluation criteria but is the only algorithm that solves all simpler (n ≤ 4) problems and is the best performing algorithm, including hybridized versions. Moreover, on average, DIRECT-GL required approximately 35 percent fewer evaluations of the objective function than the second-best, BIRECT algorithm, among all traditional DIRECT-type algorithms on a class of more challenging problems (see Average results (n ≥ 5)). Not surprisingly, the hybridized TOMLAB/glcCluster and BIRMIN algorithms within this class deliver the best average performance. Compared with the best traditional DIRECT-type algorithm, DIRECT-GL, TOMLAB/glcCluster, and BIRMIN require approximately 37% and 32% fewer evaluations.

Based on 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
measures. 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. Overall, DIRECT-GL showed the most promising performance among all tested traditional DIRECT-type algorithms.

The data profiles of all algorithms are shown in Figure 7. Again, the data profiles confirm that the hybridized algorithms (enriched with fmincon procedure) are more efficient than traditional DIRECT-type approaches. All three versions (DIRECT-rev, DIRMIN, and BIRMIN) can solve about 60% of problems in less than 1,000 function evaluations (see the left panel of Figure 7). However, by increasing the maximal budget of function evaluations, which is needed for more challenging problems, the TOMLAB/glClust, and traditional DIRECT-GL algorithm outperformed all others, including hybridized ones. While DIRECT-GL delivers the best overall performance (based on function evaluations), data profiles in Figure 7 reveal that other DIRECT-type extensions (PLOR, DIRECT-l, BIRECT, Gb-BIRECT) perform better when the maximal budget of function evaluations is ≤ 10^5. Moreover, they can solve about 70% of problems (mainly lower dimensionality) quicker. The two-step-based selection strategy in DIRECT-GL selects a more extensive set of POH. While for more straightforward problems, this is detrimental, it often helps to locate a global solution with higher accuracy faster. In terms of execution time (see the right panel in Figure 7), DIRECT-GL is the fastest among the traditional DIRECT-type algorithms (excluding hybridized).

Some algorithms have integrated schemes helping speed up the refinement of solutions. Therefore, we tested them with a much higher precision solution (εpe = 10^{-8}). In Table 4, we summarize our experimental findings.
Table 4. The Performance of Selected DIRECT-Type Algorithms from DIRECTG0 and TOMLAB Based on 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 DIRECTGOLib v1.0) Using $\epsilon_{pe} = 10^{-8}$ in (17)

| Algorithm | Avg. # local searches | Failed | Average results $|\epsilon| \leq 10^{-8}$ | Average results $|\epsilon| \leq 10^{-5}$ | Median results $|\epsilon| \leq 10^{-8}$ | Median results $|\epsilon| \leq 10^{-5}$ |
|-----------|------------------------|--------|------------------------------------------|------------------------------------------|------------------------------------------|------------------------------------------|
| DIRECT    | 40.91                  | 378.16 | 1.065.71 / 915.13 / 7.832                | 1.483.34 / 17.456                      | 1.426.590 / 855.54 / 1.154              | 1.297.032 / 268.16 / 0.224               |
| DIRECT+restart | 28.91                  | 76.78  | 1.484.21 / 3.403 / 2.10               | 2.331.23 / 4.372.27 / 5.14           | 1.035.618 / 10.191.29 / 8.262          | 53.629 / 4.00 / 17.01                  |
| DIRECT-rev | 15.91                  | 45.97  | 396.246 / 1.955.13 / 19.366            | 200.401 / 1.445.29 / 9.595          | 511.449 / 1.065.63 / 25.104           | 544 / 0.16 / 21                      |
| BIRMIN    | 1.01                   | 12.38  | 450.873 / 26.52 / 124                 | 113.327 / 35.94 / 706              | 109.201 / 39.81 / 1.162              | 351 / 0.02 / 2                       |
| glbSolve  | 35.91                  | 35.84  | 1.424.737 / 20.42 / 9.669              | 252.086 / 1.208.33 / 19.964         | 1.795.885 / 449.53 / 4.906           | 2.000.000 / 340.86 / 1.201            |
| DIRECT    | 38.91                  | 686.56 | 1.066.521 / 6.028.44 / 11.737         | 206.737 / 1.733.35 / 21.678        | 1.106.796 / 2.350.90 / 9.017         | 142.125 / 60.28 / 406                |
| DIRECT-rev | 39.91                  | 1,396.766 | 476.46 / 6.438  | 207.144 / 1.004.82 / 1.939  | 0.124.299 / 609.68 / 8.732  | 112.661 / 141.57 / 706                |
| BIRMIN    | 4                     | 12.91  | 289.793 / 4.821.15 / 18.742           | 116.309 / 2.865.45 / 11.448         | 114.609 / 5.979.93 / 23.033          | 37.59 / 0.06 / 21                     |
| DIRECT-L  | 21.91                  | 572.082 | 15.41 / 954  | 136.742 / 86.78 / 695  | 2.420.100 / 189.42 / 1.104  | 21.968 / 14.45 / 87                  |
| DIRECT-GL | 28.91                  | 669.81 | 277.242 / 29.26 / 241                | 85.123 / 26.84 / 252             | 290.294 / 31.52 / 235             | 28.231 / 1.17 / 60                    |
| TOMLAB/glccCluster | 46          | 753.378 | 16.907.54 | 315.937 / 6.959.64 / 3 | 200.121 / 22.553.22 / 3 | 66.396 / 54.86 / 3                |

*- a hybrid version of the algorithm, enriched with the local search subroutine.

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 number of unsolved problems is achieved using DIRECT-GL (6/81), where all failed test problems belong to the ($n \geq 5$) class. The DIRECT-GL algorithm turns out to be more efficient even than hybrid methods. Overall, DIRECT-GL required approximately 7% fewer function evaluations and took 32% less time than the second and third best algorithms, BIRMIN and DIRMIN, accordingly (see Average results column in Table 4).

However, the BIRMIN algorithm has the best median value (see Median results column), 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, glbSolve failed more than half of the test problems to solve. The performance of the TOMLAB/glccCluster algorithm, which showed the best average results in the previous study, has also decreased significantly (see the Average results column in Tables 3 and 4). It turns out that the use of fmincon with default parameters in hybridized methods (DIRECT-rev, DIRMIN, and BIRMIN) is much more effective in finding a solution with higher accuracy than the TOMLAB/glccCluster. Furthermore, TOMLAB/glccCluster turned out to be the slowest algorithm among all algorithms, including hybridized ones.

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 DIRECTGOLib v1.0, 35 test problems contain linear constraints, 39 problems have non-linear constraints where 5 include equality constraints. All necessary details about the test problems are given in Appendix A, Table 13.

We used the same stopping condition in these experimental investigations as in the previous ones, and the value $\epsilon_{pe} = 10^{-2}$.

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

$$h(x) = 0 \rightarrow |h(x)| \leq \epsilon_h,$$

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

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
Fig. 8. Data profiles of selected DIRECT-type algorithms from DIRECTG0 and TOMLAB on the whole set of box-constrained optimization problems from DIRECTG0.1 using $\epsilon_{pe} = 10^{-8}$ in (17).

Table 5. The Performance of DIRECT-Type Algorithms from DIRECTG0 and TOMLAB Based on 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 Constrained (Hidden, General, and Linear) Optimization Problems

| Algorithm       | Parameter | Failed | Average results | Average results (Non-lim. constr.) | Average results (Lin. constr.) | Median results |
|-----------------|-----------|--------|-----------------|-----------------------------------|-------------------------------|----------------|
|                 |           |        | $f_{eval}$ | $time$ | $iter$ | $f_{eval}$ | $time$ | $iter$ | $f_{eval}$ | $time$ | $iter$ | $f_{eval}$ | $time$ | $iter$ | $f_{eval}$ | $time$ | $iter$ |
| DIRECT-NMS      |           | N/A    | 30/90         | 784,197 | 16,871.99 | 20,694 | 984,146 | 20,454.37 | 51,130 | 558,915 | 11,123.22 | 7,278 | 853 | 18.68 | 144 |
| DIRECT-Barrier  |           | N/A    | 49/9 | 1,266,749 | 2,977.93 | 39,949 | 1,365,138 | 3,821.83 | 51,630 | 1,237,931 | 1,801.48 | 22,865 | 2,000,000 | 723.94 | 3,802 |
| subDIRECT-Barrier | $ub = 2$ | N/A    | 56/9 | 1,491,636 | 363.18 | 266 | 1,192,816 | 859.06 | 146 | 1,444,119 | 826.76 | 420 | 2,000,000 | 330.29 | 64 |
| subDIRECT-Barrier | $ub = 3$ | N/A    | 50/8 | 1,284,972 | 775.91 | 3,634 | 1,315,122 | 773.81 | 902 | 1,246,207 | 778.62 | 1,747 | 2,000,000 | 286.48 | 258 |
| subDIRECT-Barrier | $ub = 5$ | N/A    | 49/9 | 1,262,177 | 1,364.94 | 11,424 | 1,466,887 | 1,526.85 | 10,624 | 1,121,836 | 1,165.92 | 12,406 | 2,000,000 | 594.87 | 2,132 |
| DIRECT-Glibc    |           | N/A    | 19/38        | 496,619 | 354.06 | 11 | 933,165 | 491.81 | 96 | 297,918 | 85.53 | 135 | 3,924 | 68.68 | 31 |

* = a hybrid version of the algorithm, enriched with the local search subroutine.
N/A – Not available.

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. We have used all constrained test problems but assumed that any information about the constraints is unavailable. In the experimental investigation, the hidden search area ($D_{\text{hidden}}$) was defined as

$$D_{\text{hidden}} = \{x \in D : g(x) \leq 0, h(x) = 0\}. \quad (20)$$

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 these 13 test problems.
The best among all DIRECT-type algorithms for problems with hidden constraints is DIRECT-GLh (failed to solve (19/80)), while the second-best is DIRECT-NAS (failed to solve (30/80)). The median number of function evaluations (see \( f_{\text{eval}} \) in Median results) is slightly better for the DIRECT-GLh. Furthermore, DIRECT-GLh is the best, primarily based on the number of iterations (\( \text{iter.} \)) and the execution time: DIRECT-GLh took around 4.6 times fewer iterations and approximately 27.5 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 [58] did not make any sensitivity analysis and guidance. In our experiments, we start the subdividing step at \( \text{sub}^k, k = 1, 2, \ldots \) iterations. We tested three different values for the variable \( \text{sub} \), i.e., \( \text{sub} = 2, 3, \) and 5. Our 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 (\( \text{iter.} \)) and the execution time. subDIRECT-Barrier algorithm suffers solving larger dimensionality and problems where \( D_{\text{hidden}} \) contains non-linear constraints (see Average results (Non-lin. constr.) column). Therefore, this limits subDIRECT-Barrier applicability primarily to low-dimensional problems.

Finally, in Figure 9, the comparative performance of algorithms using the data profiles is demonstrated. They confirm that the DIRECT-GLh is the most effective optimizer in this class and has the highest efficiency based on the function evaluations and execution time.

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 can be much more successful (compared to the algorithms for problems with hidden) in solving problems containing complex and tiny feasible regions. The best average results among traditional DIRECT-type algorithms for constrained optimization were obtained using the DIRECT-GLce (failed to solve 8/80). 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. It looks that DIRECT-GLc is effective on simpler problems, but the effectiveness drops in solving more complicated problems, e.g., higher dimensionality with non-linear constraints. We also note that the solution point is often located on the feasible region’s boundaries for optimization problems with general constraints. The common problem of some DIRECT-type algorithms in this class (DIRECT-Barrier, subDIRECT-Barrier) 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 as POH. In such situations, these algorithms converge very slowly.

The best traditional DIRECT-type algorithm based on the median value was the TOMLAB/glCsolve method (see Median results column in Table 5). However, this is the only category where this algorithm showed the best results. Overall, the TOMLAB/glCsolve algorithm required 59% times more objective function evaluations than DIRECT-GLce. Furthermore, the TOMLAB/glCsolve appears to be the slowest algorithm in this class.
Fig. 9. Data profiles of DIRECT-type algorithms for problems with hidden constraints on the whole set of constrained optimization test problems. Explicit information about the constraints for all these algorithms was unknown.

Fig. 10. Data profiles of DIRECT-type algorithms from DIRECTGO and TOMLAB on the whole set of constrained optimization test problems.

Hybridized DIRECT–GLce–min and TOMLAB/glCCluster are the only two candidates among all approaches within this class. Again, incorporating the local minimization procedure into DIRECT–GLce improves the performance significantly, e.g., it reduces the overall number of function evaluations approximately four times. Moreover, DIRECT–GLce–min fails to solve only two test problems. TOMLAB/glCCluster fails to solve eight test instances and, on average, is around 318 times slower than the DIRECT–GLce–min method.

Finally, in Figure 10, the comparative performance using the data profiles tool is demonstrated. Data profiles confirm the same trends, i.e., the hybridized versions are the best performing, and the overall advantage of methods that incorporate constrained information versus designed explicitly for problems with hidden constraints (see also Figure 9).

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 [67]. The main advantage of 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 DIRECTGOLib v1.0, 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 data profiles (see Figure 11) reveal the overall effectiveness of the Lc-DISIMPL-V algorithm for such problems with linear constraints. The latter algorithm even outperformed the hybridized DIRECT-GLce-min method, which is surprisingly enough. Nevertheless, the efficiency of simplicial partition-based algorithms suffers from the problem dimension. Three higher dimensionality linearly constrained problems were unsolved by the Lc-DISIMPL-V algorithm (see the bottom part of Table 5, "Performance of DIRECT-type algorithms devoted to problems with linear constraints only"). Moreover, simplicial partition-based implementations are pretty slow. For example, on average, DIRECT-GLce is approximately 248 times faster than Lc-DISIMPL-V. Moreover, DIRECT-GLce-min and TOMLAB/glcCluster solved all test problems with linear constraints.

5 DIRECTGOL PERFORMANCE ON ENGINEERING PROBLEMS

In this section, the algorithms from DIRECTGOL and TOMLAB were tested on 11 engineering design problems: tension/compression spring, three-bar truss, NASA speed reducer, pressure vessel, welded beam, and six different versions of the general non-linear regression problem. The general non-linear regression problem is box-constrained, while the others involve different constraints. Only the most promising algorithms (based on Section 4) were considered. We used the same stopping rule as the global minimums are known for all these engineering problems. 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 [40], while in Appendix B.1, we give a short description and mathematical formulation.

A comparison of found solutions and performance metrics by the algorithms from DIRECTGOL and TOMLAB 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
Table 6. Performance of the DIRECT-Type Algorithms from DIRECTGO and TOMLAB on a Tension/Compression Design Problem

| Algorithm (input) | Parameter | Iterations | \(f_{eval}\) | Time (s) | \(f_{min}\) |
|-------------------|-----------|------------|--------------|-----------|-----------|
| DIRECT-NAS        | –         | 297        | 17,659       | 77.46     | 0.012680  |
| DIRECT-Barrier    | –         | 40,478     | \(>2 \times 10^6\) | 2.465.81  | 0.012867  |
| subDIRECT-Barrier | \(sub = 2\) | 256        | \(>2 \times 10^6\) | 207.89    | 0.012708  |
| subDIRECT-Barrier | \(sub = 3\) | 2,187      | \(>2 \times 10^6\) | 198.94    | 0.012741  |
| subDIRECT-Barrier | \(sub = 5\) | 15,626     | \(>2 \times 10^6\) | 722.79    | 0.012867  |
| DIRECT-GLh        | –         | 700        | \(>2 \times 10^6\) | 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         | \(y = 10^1\) | 34,395     | \(>2 \times 10^6\) | 3,265.38  | 0.012755  |
| DIRECT-L1         | \(y = 10^2\) | 33,897     | \(>2 \times 10^6\) | 3,147.87  | 0.012867  |
| DIRECT-L1         | \(y = 10^3\) | 33,571     | \(>2 \times 10^6\) | 3,041.00  | 0.012867  |
| TOMLAB/glcsolve   | –         | 64         | \(>2 \times 10^6\) | 560.36    | 0.014669  |
| TOMLAB/glcCluster*| –         | 12         | 1,528,205    | 43,000.00 | 0.014669  |

\* - a hybrid version of the algorithm, enriched with the local search subroutine.

significantly better than other algorithms. The surprise is that the algorithm does not use any information about the constraint functions. However, the DIRECT-NAS algorithm was approximately four times slower than the second-best method DIRECT-GLce. Another surprise was that none of the hybridized algorithms performed well on this practical problem. Moreover, the TOMLAB/glccCluster algorithm failed to find a solution within the given time limit.

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 constraints. A detailed description of the problem is given in [74], 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 (with the proper input parameters) had any difficulty solving this problem, and they found the solution in less than one-second time.

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 [74], 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 three algorithms (DIRECT-GLce, DIRECT-GLce-min, and TOMLAB/glccCluster) were able to tackle this problem. Again, the hybridized DIRECT-GLce-min algorithm showed the best performance. Note that the found solutions with DIRECT-L1 are better than the best-known value \(f_{min}\). However, the reported solution point is outside the feasible region and violates some constraints. The TOMLAB/glcsolve was very close to a solution, but could not find it with a required accuracy within the maximum time limit.
In this subsection, we consider a pressure vessel design problem, and the goal is to minimize the total cost of the material, form, and weld a cylindrical vessel. A detailed description of the problem can be found in [40], 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. Five algorithms solved this problem: DIRECT-NAS, DIRECT-GLh, DIRECT-GLce, TOMLAB/glCCluster, and DIRECT-GLce-min was the most efficient optimizer again. DIRECT-NAS is the best

| Algorithm (input) | Parameter | 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 = 2   | 512        | $>2 \times 10^6$ | 141.63   | 283.223781 |
| subDIRECT-Barrier| sub = 3   | 17         | 333        | 0.02     | 263.915800 |
| 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        | $\gamma = 10^1$ | 1          | 1          | 0.01     | 199.705600* |
| DIRECT-L1        | $\gamma = 10^2$ | 2          | 11         | 0.01     | 262.344700* |
| DIRECT-L1        | $\gamma = 10^3$ | 17         | 179        | 0.03     | 263.915790 |
| TOMLAB/glC Solve | –         | 25         | 647        | 0.43     | 263.910482 |
| TOMLAB/glC Cluster* | –       | 1          | 992        | 0.55     | 263.910482 |

* – a hybrid version of the algorithm, enriched with the local search subroutine.

a – result is outside the feasible region.

### 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, form, and weld a cylindrical vessel. A detailed description of the problem can be found in [40], 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. Five algorithms solved this problem: DIRECT-NAS, DIRECT-GLh, DIRECT-GLce, TOMLAB/glCCluster, and 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-known value $f_{\text{min}}$, but the solution points lay outside the feasible region.

### 5.5 Welded Beam Design Problem

The fifth engineering problem is the welded beam design. The goal is to minimize a welded beam for a minimum cost, subject to seven constraints. The detailed description is presented in [54, 55], while in Appendix B.5, we provide a short description and mathematical formulation.

A comparison of the algorithms is shown in Table 10. In total, six algorithms were able to solve the problem, and once again, the DIRECT-GLce-min was the most efficient one. Again, the

#### Table 9. Performance of the DIRECT-Type Algorithms from DIRECTGO and TOMLAB on a Pressure Vessel Design Problem

| Algorithm (input) | Parameter | Iterations | $f_{\text{eval}}$ | Time (s) | $f_{\text{min}}$ |
|-------------------|-----------|------------|-------------------|----------|-----------------|
| DIRECT-NAS        | –         | 273        | 31,081            | 126.64   | 7,164.437307    |
| DIRECT-Barrier    | –         | 26,379     | $>2 \times 10^8$  | 1,600.15 | 7,234.041903    |
| subDIRECT-Barrier | $sub = 2$ | 128        | $>2 \times 10^6$  | 190.11   | 7,234.402264    |
| subDIRECT-Barrier | $sub = 3$ | 729        | $>2 \times 10^6$  | 99.05    | 7,234.222516    |
| subDIRECT-Barrier | $sub = 5$ | 15,626     | $>2 \times 10^6$  | 1,328.98 | 7,234.041903    |
| DIRECT-GLh        | –         | 252        | 55,837            | 4.80     | 7,164.347300    |
| DIRECT-GLc        | –         | 2,358      | $>2 \times 10^6$  | 433.87   | 7,224.704257    |
| DIRECT-GLce       | –         | 322        | 88,585            | 8.52     | 7,164.347301    |
| DIRECT-GLce-min*  | –         | 1          | 134               | 0.24     | 7,163.739570    |
| DIRECT-L1         | $\gamma = 10^1$ | 86 | 2,117 | 0.34 | 7,025.940549\(^a\) |
| DIRECT-L1         | $\gamma = 10^2$ | 86 | 2,099 | 0.33 | 7,037.428049\(^a\) |
| DIRECT-L1         | $\gamma = 10^3$ | 87 | 2,295 | 0.23 | 7,152.303079\(^a\) |
| TOMLAB/glcSolve   | –         | 123        | $>2 \times 10^6$  | 529.69   | 8,260.982616    |
| TOMLAB/glcCluster*| –         | 1          | 10,026            | 5.74     | 7163.739569     |

\(^a\) - a hybrid version of the algorithm, enriched with the local search subroutine.

#### Table 10. Performance of the DIRECT-Type Algorithms from DIRECTGO and TOMLAB on a Welded Beam Design Problem

| Algorithm (input) | Parameter | Iterations | $f_{\text{eval}}$ | Time (s) | $f_{\text{min}}$ |
|-------------------|-----------|------------|-------------------|----------|-----------------|
| DIRECT-NAS        | –         | 698        | 86,863            | 4,692.00 | 1.724970        |
| DIRECT-Barrier    | –         | 22,934     | $>2 \times 10^6$  | 1,363.24 | 1.728488        |
| subDIRECT-Barrier | $sub = 2$ | 128        | $>2 \times 10^6$  | 152.58   | 1.728060        |
| subDIRECT-Barrier | $sub = 3$ | 729        | $>2 \times 10^6$  | 128.37   | 1.728043        |
| subDIRECT-Barrier | $sub = 5$ | 10,954     | $>2 \times 10^6$  | 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         | $\gamma = 10^1$ | 21,143 | $>2 \times 10^6$  | 1,995.01 | 1.728491        |
| DIRECT-L1         | $\gamma = 10^2$ | 20,879 | $>2 \times 10^6$  | 1,814.33 | 1.728491        |
| DIRECT-L1         | $\gamma = 10^3$ | 20,767 | $>2 \times 10^6$  | 1,679.76 | 1.728488        |
| TOMLAB/glcSolve   | –         | 86         | $>2 \times 10^6$  | 526.06   | 2.473711        |
| TOMLAB/glcCluster*| –         | 1          | 9,884             | 5.85     | 1.724852        |

\(^a\) - a hybrid version of the algorithm, enriched with the local search subroutine.
5.6 General Non-Linear Regression Problem

In the final part, a general non-linear 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 [26, 61, 80], while in Appendix B.6, we provide a short description and mathematical formulation. The problem is multi-modal and is considered challenging, especially with the increase in the number of samples \(T\). The higher number of sinusoids \(\zeta\) leads to a more accurate but, at the same time, more challenging optimization problem.

Our experiments have used three different values for \(\zeta = 1, 2,\) 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 [61].

The obtained results are summarized in Tables 11 and 12. Solving the lowest dimension \(n = 3\) cases (corresponding to \(\zeta = 1, T = 10\) and \(\zeta = 1, T = 100\)) all algorithms located solution correctly (the success rate is 100%). Among the hybridized methods, the BIRMIN and DIRECT-rev proved the most effective. Among the traditional algorithms, the PLOR algorithm is the most efficient in solving the first problem (with \(\zeta = 1, T = 10\) parameters). However, by increasing the number of samples \(T\), PLOR performed worst among all DIRECT-type algorithms. The most efficient algorithm for the case with \(\zeta = 1\) and \(T = 100\) was glbSolve.

For the higher dimensionality case \(n = 6\), more than half of the algorithms failed to find the correct solution. The success rates for these two cases are 42.86% and 39.28%, respectively.
The DIRECT−GL algorithm has shown a significant advantage among the traditional DIRECT-type algorithms. The best two performing hybridized methods were the BIRMIN and DIRECT−rev.

Finally, most DIRECT-type methods have faced significant challenges in solving two variants of the highest dimensional (n = 9) case. The success rates on these two cases are only 17.86% and 17.86%, respectively. While hybridized methods had no significant difficulties, among the traditional, only the DIRECT−GL algorithm solved both variants.

6 CONCLUSION

This article has introduced a new open-source DIRECT-type MATLAB toolbox (DIRECTGO) 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 DIRECTGOLib v1.0, containing 199 global optimization test and engineering problems, has been presented.

The performance of various algorithms within DIRECTGO has been investigated via a detailed numerical study using the test problems from DIRECTGOLib v1.0. A further 11 examples of using the DIRECTGO for engineering design optimization have been investigated. The results demonstrate the promising performance of DIRECTGO in tackling these challenging problems. We also gave guidance on which algorithms to use for specific optimization problems.

Motivated by the promising performance, we plan to extend this work to facilitate the broader adoption of DIRECTGO. 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 organization and reduced communication overhead.

APPENDICES

A DIRECTGOLIB V1.0 LIBRARY

A summary of all optimization problems in DIRECTGOLib v1.0 [86, 92] and their properties are given in Table 13. 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 column specifies the dimension (n) of the used problems. When problems are of various dimensionality, all considered dimensions are listed. The fifth through eighth columns (g, h, a, D) specify the number of inequality (g) and equality (h) constraints, the number of active constraints (a), and two versions of optimization domains, default (D) and perturbed (D̃), respectively. The default domains are taken from the literature. However, whenever the global minimum point lay for at least one algorithm from DIRECTGO at the initial sampling point, the default feasible region (D) was perturbed and used in the experimental analysis. Here, the sign “-” means that D̃ is the same as D. Finally, the last column contains the best-known optimal solution value (f∗).
| Problem          | Source   | n | g | h | e | D  | Problem properties | \( f^* \) |
|------------------|----------|---|---|---|---|-----|-------------------|----------|
| **AC**           | [92, 95] | 2 | 5,10 | 0 | 0 | 0 | \(-15, 30\)^1 | \(-15, 35\)^* | 0.0000 |
| **Beale**        | [12]     | 5 | 10,15 | 0 | 0 | 0 | \([0, 10]^m\)    | -2.8081*  | 0.0000 |
| **Behavevsky1**  | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-4, 5, 4.5]^n\) | 0.0000   |       |
| **Behavevsky2**  | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-100, 100]^n\) | 0.0000   |       |
| **Behavevsky3**  | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-100, 100]^n\) | 0.0000   |       |
| **Booth**        | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | 0.0000   |       |
| **Braitin**      | [18, 32, 95] | 2 | 0 | 0 | 0 | 0 | \([-5, 10] \times [0, 15]\) | 0.3978   |       |
| **Bukin6**       | [95]     | 2 | 0 | 0 | 0 | 0 | \([-15, 5] \times [-3, 3]\) | 0.0000   |       |
| **Coulve**       | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | 0.0000   |       |
| **Cross_in_Tray**| [95]     | 2 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | -2.0626  |       |
| **Csendes**      | [12]     | 5 | 10,15 | 0 | 0 | 0 | \([-10, 10]^n\)   | \([-10, 21]^*\) | 0.0000 |
| **Dixon_and_Price** | [32, 95] | 2 | 5,10 | 0 | 0 | 0 | \([-10, 10]^n\)   | 0.0000   |       |
| **Drop_wave**    | [95]     | 2 | 0 | 0 | 0 | 0 | \([-5.12, -5.12]^n\) | \([-5.12, -6.12]^*\) | -1.0000 |
| **Easom**        | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-100, 100]^n\) | -1.0000  |       |
| **Eggholder**    | [95]     | 2 | 0 | 0 | 0 | 0 | \([-512, 512]^n\) | -959.6406 |       |
| **Goldstein_and_Price** | [18, 32, 95] | 2 | 0 | 0 | 0 | 0 | \([-2, 2]^n\) | 3.0000 |       |
| **Griewank**     | [32, 95] | 5 | 10,15 | 0 | 0 | 0 | \([-600, 600]^n\) | 0.0000   |       |
| **Hartman3**     | [32, 95] | 3 | 0 | 0 | 0 | 0 | \([3]^n\)         | -3.8627  |       |
| **Hartman6**     | [32, 95] | 6 | 0 | 0 | 0 | 0 | \([0, 1]^n\)      | -3.3223  |       |
| **Holder_Table** | [95]     | 2 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | -19.2085 |       |
| **Hump**         | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-5.5]^n\)      | -1.9316  |       |
| **Langermann**   | [95]     | 2 | 0 | 0 | 0 | 0 | \([0.10]^n\)      | -4.1558  |       |
| **Levy**         | [32, 95] | 5 | 10,15 | 0 | 0 | 0 | \([-5.5]^n\)      | 0.0000   |       |
| **Matyas**       | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | \([-10, 15]^*\) | 0.0000 |
| **McCormick**    | [95]     | 2 | 0 | 0 | 0 | 0 | \([-1.5, 4] \times [-3, 4]\) | -1.9132 |       |
| **Michalewicz**  | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([0, 20]^n\)     | -1.8013  |       |
| **Michalewicz**  | [32, 95] | 5 | 0 | 0 | 0 | 0 | \([0, 7]^n\)      | -4.6876  |       |
| **Michalewicz**  | [32, 95] | 10 | 0 | 0 | 0 | 0 | \([0, 4]^n\)      | -9.6601  |       |
| **Parr**         | [32, 95] | 5 | 0 | 0 | 0 | 0 | \([-4, 4]^n\)     | 0.0000   |       |
| **Powell**       | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([-4, 4]^n\)     | \([-4, 5]^*\) | 0.0000 |
| **Power_Sum**    | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([0, 4]^n\)      | 0.0000   |       |
| **Qing**         | [12]     | 5 | 10,15 | 0 | 0 | 0 | \([-500, 500]^n\) | 0.0000   |       |
| **Rastrigin**    | [32, 95] | 2 | 5,10 | 0 | 0 | 0 | \([-5.12, 5.12]^n\) | \([-6.12, 5.12]^*\) | 0.0000 |
| **Rosenbrock**   | [18, 32, 95] | 5 | 10,15 | 0 | 0 | 0 | \([-5.10]^n\) | 0.0000 |       |
| **Rosenbrock**   | [95]     | 5 | 10,15 | 0 | 0 | 0 | \([-65.566, 65.566]^n\) | \([-65.566, 65.566]^*\) | 0.0000 |
| **Schwefel**     | [32, 95] | 2 | 5,10 | 0 | 0 | 0 | \([-500, 500]^n\) | 0.0000   |       |
| **Shekel2**      | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([0,10]^n\)      | -10.1531 |       |
| **Shekel3**      | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([0,10]^n\)      | -10.4029 |       |
| **Shekel10**     | [32, 95] | 4 | 0 | 0 | 0 | 0 | \([0,10]^n\)      | -10.5364 |       |
| **Shaffer**      | [32, 95] | 2 | 0 | 0 | 0 | 0 | \([-10, 10]^n\)   | -186.7309 |       |
| **Sphere**       | [32, 95] | 5 | 10,15 | 0 | 0 | 0 | \([-5,5]^n\)      | \([-5.12, 6,12]^*\) | 0.0000 |
| **Styblinski_Tang** | [12]    | 5 | 10,15 | 0 | 0 | 0 | \([-5.5]^n\)      | -39.1661 |       |
| **Sum_of_Powers**| [95]     | 5 | 10,15 | 0 | 0 | 0 | \([-1.1]^n\)      | \([-1.2, 5]^*\) | 0.0000 |
| **Sum_Square**   | [95]     | 5 | 10,15 | 0 | 0 | 0 | \([-10, 10]^n\)   | \([-10, 15]^*\) | 0.0000 |
| **Tribe**        | [32, 95] | 6 | 0 | 0 | 0 | 0 | \([-36, 36]^n\)   | -50.0000 |       |
| **TriD0**        | [32, 95] | 10 | 0 | 0 | 0 | 0 | \([-100, 100]^n\) | -210.0000 |       |
| **Zakharov**     | [32, 95] | 2 | 5,10 | 0 | 0 | 0 | \([-5.10]^n\) | \([-5.11]^*\) | 0.0000 |

(Continued)
| Type     | Name            | Source | g | h | a | D         | $f^*$       |
|----------|-----------------|--------|---|---|---|------------|-------------|
| Goldstein_and_PriceC | Gomez | [5]    | 2 | 1 | 0 | [-2, 1]   | -0.9711     |
| Himmeleblau | P1  | [23]   | 5 | 0 | 3 | [-5.5]    | 0.0293      |
|           | P2a | [23]   | 5 | 0 | 6 | [0, 500]  | -400.0000   |
|           | P2b | [23]   | 5 | 0 | 5 | [0, 500]  | -600.0000   |
|           | P2c | [23]   | 5 | 0 | 4 | [0, 500]  | -750.0000   |
|           | P2d | [23]   | 5 | 0 | 5 | [0, 100]  | -400.0000   |
|           | P3a | [23]   | 6 | 1 | 4 | 0.1       | 0.3888      |
|           | P3b | [23]   | 2 | 1 | 0 | 10^{-5}, 16] | 0.3888     |
|           | P4  | [23]   | 2 | 1 | 0 | [0, 6]   | -6.6666     |
|           | P5  | [23]   | 2 | 2 | 2 | 0.9, 4.22 | 201.1600    |
|           | P6  | [23]   | 2 | 1 | 0 | [0, 115.8] | 376.2900    |
|           | P7  | [23]   | 2 | 0 | 1 | [-2, 1]  | -2.8284     |
|           | P8  | [23]   | 2 | 0 | 1 | [-8, 10] | -118.7000   |
|           | P10 | [23]   | 2 | 0 | 2 | [0, 1]   | 0.7417      |
|           | P11 | [23]   | 2 | 1 | 0 | [0, 2]   | -0.5000     |
|           | P12 | [23]   | 1 | 2 | 0 | [0, 2]   | -16.7390    |
|           | P13 | [23]   | 3 | 0 | 2 | 2 10^{-5}, 34] | 0.3888     |
|           | P15 | [23]   | 3 | 0 | 3 | 10^{-3}, 12.5] | 0.7417     |
|           | P16 | [23]   | 2 | 0 | 2 | [0, 1]   | 0.7417      |
| s365mod  | Tproblem | [21]   | 2 | 3 | 4, 5, 6, 7, 8 | 0.7417     |
|           | zy2 | [97]   | 3 | 3 | 0 | 1 0, 10] | -n          |
|           | zeevei3 | [97]  | 2 | 0 | 1 | [0, 10] | 97.3094     |
|           | zeeveic4 | [97] | 2 | 0 | 1 | [0, 10] | 7.5575      |
B \hspace{1em} THE MATHEMATICAL FORMULATION OF ENGINEERING PROBLEMS

B.1 Tension/Compression Spring Design Problem

The design variables of the tension/compression spring design problem [40] 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

$$
\min f(x) = x_1^2 x_2 (x_3 + 2)
$$

s.t. $g_1(x) = 1 - \frac{x_3^2 x_2}{71875 x_1^4} \leq 0$, $g_2(x) = \frac{x_2 (4 x_2 - x_1)}{12566 x_1^2 (x_2 - x_1)} + \frac{2.46}{12566 x_1^2} - 1 \leq 0$, $g_3(x) = 1 - \frac{140.54 x_1}{x_3 x_2^2} \leq 0$, $g_4(x) = \frac{x_1 + x_2}{1.5} - 1 \leq 0,$

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 $x^* = (0.05169591, 0.35688327, 11.29337893)$, where $f(x^*) = 0.01267867$. 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 [74] has two design variables and three constraints. The optimization problem is formulated as follows:

$$
\min f(x) = 100 (2 \sqrt{2} x_1 + x_2)
$$

s.t. $g_1(x) = \frac{\sqrt{2} x_1 + x_2}{\sqrt{2} x_1^2 + 2 x_1 x_2} - 2 - 2 \leq 0$, $g_2(x) = \frac{x_2}{\sqrt{2} x_1^2 + 2 x_1 x_2} - 2 \leq 0$, $g_3(x) = \frac{1}{x_1 + \sqrt{2} x_2} - 2 \leq 0,$

where $0 \leq x_1 \leq 1, \ 0 \leq x_2 \leq 1.$ The best known solution $x^* = (0.78867531, 0.40824778)$, where $f(x^*) = 263.89584337.$ 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 [74] 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:

$$
\min f(x) = 0.7854 x_1 x_3^2 (3.3333 x_2^2 + 14.9334 x_3 - 43.0934) - 1.508 x_1 (x_5^2 + x_7^2) + 7.4777 (x_6^2 + x_7^2)
\quad + 0.7854 (x_4 x_5^2 + x_3 x_7^2)
$$

s.t. $g_1(x) = \frac{27}{x_1 x_2^2 x_3} - 1 \leq 0$, $g_2(x) = \frac{397.5}{x_1 x_2^2 x_3^2} - 1 \leq 0$, $g_3(x) = \frac{1.93 x_4^3}{x_2 x_3 x_6^4} - 1 \leq 0$, $g_4(x) = \frac{1.93 x_3^3}{x_2 x_3 x_7^4} - 1 \leq 0$, $g_5(x) = \frac{(\frac{745 x_4}{x_2 x_3})^2 + 16.9 \times 10^6}{110 x_6^3} - 0.5 \leq 1,$

where

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
\[
g_6(x) = \frac{\left(\frac{745x_5}{x_2x_3} + 157.5 \times 10^6\right)^{0.5}}{85x_7^3} - 1 \leq 0,
\]
\[
g_7(x) = \frac{x_2x_3}{40} - 1 \leq 0, \quad g_8(x) = \frac{5x_2}{x_1} - 1 \leq 0, \quad g_9(x) = \frac{x_1}{12x_2} - 1 \leq 0,
\]
\[
g_{10}(x) = \frac{1.5x_6 + 1.9}{x_4} - 1 \leq 0, \quad g_{11}(x) = \frac{1.1x_7 + 1.9}{x_5} - 1 \leq 0,
\]
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 \( x^* = (3.5, 0.7, 17, 7.3, 7.8, 3.35021467, 5.28668323) \), where \( f(x^*) = 2996.34816924 \). Three constraints are active \( (g_5, g_6, \text{and } g_8) \).

### B.4 Pressure Vessel Design Problem

There are four design variables in the pressure vessel design problem [40] (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:

\[
\begin{align*}
\text{min } & f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\
\text{s.t. } & g_1(x) = -x_1 + 0.0193x_3 \leq 0, \quad g_2(x) = -x_2 + 0.00954x_3 \leq 0, \\
& g_3(x) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0, \\
& g_4(x) = x_4 - 240 \leq 0, \quad g_5(x) = 1.1 - x_1 \leq 0, \quad g_6(x) = 0.6 - x_2 \leq 0,
\end{align*}
\]

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 \( x^* = (1.1, 0.625, 56.99481865, 51.00125173) \), where \( f(x^*) = 7163.73956887 \). Three constraints are active \( (g_1, g_3, \text{and } g_5) \).

### B.5 Welded Beam Design Problem

The welded beam design problem [54, 55] is to design a welded beam at minimum cost, subject to some constraints [54, 55]. The objective is to find a minimum fabrication cost. Considering the four design variables and constraints of shear stress \( \tau \), bending stress in the beam \( \sigma \), buckling load on the bar \( P_c \), and end deflection on the beam \( \delta \). The optimization model is summarized in the following equation:

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

with

\[
\tau(x) = \sqrt{(\tau^1)^2 + (\tau^1)(\tau^2)x_2/R + (\tau^2)^2}, \quad P_c = \frac{4.013E_4}{L^2} \frac{x_3^2x_4^6/36}{(1 - \frac{x_3}{2L} \sqrt{\frac{E}{4G}})},
\]

\[
R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2},
\]

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
\[ r^1 = \frac{P}{\sqrt{2}x_1x^*}, \quad r^2 = \frac{MR}{J}, \quad \sigma(x) = \frac{6PL}{x_4x^*_3}, \quad J = 2 \left( \sqrt{2}x_1x_2 \left( \frac{x^2}{12} + \frac{1}{4}(x_1x_3)^2 \right) \right), \]

\[ \delta(x) = \frac{4PL^2}{Ex_4x^*_3}, \quad M = P \left( L + \frac{x_2}{2} \right), \]

where \(0.1 \leq x_1 \leq 2, 0.1 \leq x_2 \leq 10, 0.1 \leq x_3 \leq 10, 0.1 \leq x_3 \leq 2.\) The best known solution \(x^* = (0.20572963, 3.47048893, 9.03662399, 0.20572964),\) where \(f(x^*) = 1.72485237.\) One of the constraint functions is active \((g_3).\)

### B.6 General Non-Linear Regression Problem

Parameter estimation in the general non-linear regression model \([26, 61, 80]\) can be reduced to solving the minimization problem:

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

with

\[ \sum_{q=1}^{\varsigma} \kappa(t) = e^{\ell_d q} \sin(2\pi t \omega q + \theta q), \quad \sum_{q=1}^{\varsigma} \phi(x, t) = e^{(x_{3(q-1)+1} t)} \sin(2\pi t x_{3(q-1)+2} + x_{3(q-1)+3}), \]

where \(-1 \leq x_{3(q-1)+1} \leq 0, 0 \leq x_{3(q-1)+2,3(q-1)+2} \leq 1, q = 1 \ldots \varsigma. d is non-positive damping coefficients, \(\omega is frequencies, and \(\theta is phases of the sinusoids (\varsigma) (hereafter, \(d_q \in [-1, 0], \omega_q \in [0, 1], \theta_q \in [0, 1], q = 1 \ldots \varsigma). For signal approximation, the parameter \(x of the problem is determined to fit best the real-valued signal values observed in the uniformly distributed time moments \(t = 1, 2, \ldots, T.\) The general non-linear regression problem is multi-modal, especially with the increase in the number of samples \(T.\) The increase of the sinusoid number \(\varsigma leads to a more accurate but at the same time more challenging optimization problem. In our experimental study, six versions of the problem were considered. The sinusoid number was fixed to \(\varsigma = 1, 2, and 3 (corresponding to 3, 6, and 9-dimensional cases, while the value of \(T to 10 and 100.\) The best known solutions: (i) \(f(x^*) = 0 and x^* = (-0.2, 0.4, 0.3) for n = 3; (ii) f(x^*) = 0 and x^* = (-0.3, 0.3, 0.1, -0.2, 0.4, 0.3) for n = 6; (iii) f(x^*) = 0 and x^* = (-0.4, 0.6, 0.2, -0.3, 0.3, 0.1, -0.2, 0.4, 0.3) for n = 9.

### C QUICK DIRECTGO USER GUIDE

This section provides a brief user guide on how to use DIRECTGO software. The following subsections provide examples of using algorithms (their implementations) to solve box constrained and problems with various constraints, including the parallel usage of the algorithms.

#### C.1 Example of Box Constrained Global Optimization Algorithm Usage

Any DIRECT-type algorithmic implementation from the DIRECTGO MATLAB toolbox for box constrained global optimization can be called using the same style and syntax introduced in Section 3.2. In this example, we use the PLOR algorithm and solve Bukin\textsuperscript{o} test problem (see Table 13).
The *Bukin6* test problem is defined in MATLAB in the following way:

```matlab
function y = Bukin6(x)
    if nargin == 0
        y.nx = 2; % Dimension of the problem
        xl = [-15; -3];
        y.xl = @(i) xl(i);
        xu = [5; 3];
        y.xu = @(i) xu(i);
        y.fmin = @(nx) get_fmin(nx); % Known solution value
        y.xmin = @(nx) get_xmin(nx); % Known solution point
        return
    end
    term1 = 100*sqrt(abs(x(2) - 0.01*x(1)^2));
    term2 = 0.01*abs(x(1) + 10);
    y = term1 + term2; % Return function value at x
end
```

Each test problem in the DIRECTGOLib v1.0 stores the information about the problem structure together with the objective function. In this case, in the *Bukin6.m* file, the following information is stored: (i) the dimensionality of the problem; (ii) the lower and upper bounds for each variable; (iii) the objective function value of the known solution; (iv) the solution point. For some problems, the optimum might depend on the number of variables, therefore the solution values and points are returned as a functions for all test problems in DIRECTGOLib v1.0.

The optimization problem is passed to the algorithm as part of a `P` structure. For a box-constrained problem, only one field of the `P` structure is needed:

```matlab
>> P.f = 'Bukin6';
```

When a user wants to change the default algorithmic settings, the `OPTS` structure should be used:

```matlab
>> opts.maxevals = 50; % Maximal number of function evaluations
>> opts.maxits = 100; % Maximal number of iterations
>> opts.testflag = 1; % 1 if global minima is known, 0 otherwise
>> opts.tol = 0.01; % Tolerance for termination if testflag = 1
```

Now we are ready to call the dynamic data structure based PLOR implementation (`dPlor.m`) to solve this problem:

```matlab
>> [f_min, x_min, history] = dPlor(P, OPTS);
```
The iterative output stored in the `history` parameter contains the following information:

```matlab
>> history

history =

    1.0000    5.0000   16.7833    0.0023
    2.0000    7.0000   16.7833    0.0030
    3.0000   13.0000    5.6500    0.0039
    4.0000   19.0000    5.6500    0.0046
    5.0000   27.0000    1.9537    0.0053
    6.0000   33.0000    1.9537    0.0060
    7.0000   41.0000    0.7167    0.0070
    8.0000   47.0000    0.7167    0.0077
    9.0000   55.0000    0.3060    0.0086
```

Here, the first column shows the iteration number, while the second is the total number of function evaluations. The third column shows how the best objective function value improves at each iteration, while the last column shows the execution time in seconds. In this example, the PLOR algorithm was terminated when the maximum number of function evaluations (`opts.maxevals = 50`) exceeded.

The convergence plot is shown on the right panel of Figure 12, while the left panel illustrates the `Bukino` test function over its domain.

### C.2 Example of Constrained Global Optimization Algorithm Usage

Any DIRECT-type algorithmic implementation for constrained global optimization problems can be used the same way as box-constrained problems. However, for constrained problems, implemented algorithms extract additional information from the function’s definition, such as the number of
inequality constraints, the number of equality constraints, and the constraint functions. Let us take the \textit{G06} problem (see Table 13) as an example, defined in the following way:

```matlab
function y = G06(x)
    if nargin == 0
        % Extract info from the function
        y.nx = 2;
        % Dimension of the problem
        y.ng = 2;
        % Number of g(x) constraints
        y.nh = 0;
        % Number of h(x) constraints
        xl = [13, 0];
        % Lower bounds for each variable
        xu = @(i) 100;
        % Known solution value
        y.fmin = @(nx) get_fmin(nx);
        y.xmin = @(nx) get_xmin(nx);
        y.confun = @(i) G06c(i);
        return
    end
    y = (x(1) - 10)^3 + (x(2) - 20)^3; % Return function value at x
end
```

```matlab
function [c, ceq] = G06c(x)
    c(1) = -(x(1) - 5)^2 - (x(2) - 5)^2 + 100;
    c(2) = (x(1) - 6)^2 + (x(2) - 5)^2 - 82.81;
    ceq = [];
end
```

```matlab
function fmin = get_fmin(~)
    fmin = -6961.813875127380248;
end
```

```matlab
function xmin = get_xmin(~)
    xmin = [14.09500000002011322; 0.8429607896175201];
end
```

Same as in Appendix C.1, the constrained optimization problem is passed to the algorithm as part of a \textit{P} structure:

```matlab
>> P.f = 'G06';
```

Next, assume that a user wants to stop the search as soon as the known solution is within a 0.01\% error. The \textit{OPTS} structure should be specified as follows:

```matlab
>> opts.maxevals = 10000; % Maximal number of function evaluations
>> opts.maxits = 1000; % Maximal number of iterations
>> opts.testflag = 1; % 1 if global minima is known, 0 otherwise
>> opts.tol = 0.01; % Tolerance for termination if testflag = 1
>> opts.showits = 1; % Print iteration status
```

The desired solver (in this case implementation of the DIRECT-GLc algorithm) is run using:
\[ [f_{\text{min}}, x_{\text{min}}, \text{history}] = \text{dDirect\_GLc}(P, \text{OPTS}); \]

Since opts.showits = 1, the optimization result after each iteration is printed in the MATLAB command window:

Phase II - searching feasible point:
- con viol: 2404.4400000000 fn evals: 5
- con viol: 515.5511111111 fn evals: 7
...
- con viol: 0.1374240038 fn evals: 123
- con viol: 0.0000000000 fn evals: 159 f_min: -5612.1483164940

Phase I - Improve feasible solution:
- Iter: 1 f_min: -5886.5625227848 time(s): 0.05935 fn evals: 197
- Iter: 2 f_min: -5931.8554991123 time(s): 0.06473 fn evals: 241
...
- Iter: 13 f_min: -6873.0583159376 time(s): 0.13197 fn evals: 947
- Iter: 14 f_min: -6901.509981387 time(s): 0.13869 fn evals: 1027

Minima was found with Tolerance: 1

We see that the solution \( f_{\text{min}} = -6901.509981387 \) (within a 0.01% error) was found after 14 iterations.

C.3 Parallel Algorithm Usage

This section briefly explains how to use parallel versions of the algorithms. We can see which algorithms are implemented in parallel in Table 1. Assume a user wishes to use parallel code for the PLOR algorithm. First, a parallel implementation of the PLOR algorithm (parallel_dPlor.m) should be chosen. Next, a user should specify the number of workers (computational threads). For parallel PLOR, it is reasonable to select 2, as only two POH are selected per iteration. In this case, MATLAB parallel pool size should be specified using the parpool command, after which the parallel algorithm should be executed:

\[ \text{parpool}(2); \]
\[ [f_{\text{min}}, x_{\text{min}}, \text{history}] = \text{parallel\_dPlor}(P, \text{OPTS}); \]

By default, the parpool command starts the MATLAB pool on the local machine with one worker per physical CPU core. Using parpool(2), we limit the number of workers to 2. After this, the parallel code is executed using both workers. However, it should be taken into account that creating parallel parpool takes some time. Therefore, using the parallel PLOR algorithm is inefficient in solving simple problems. The use of parallel codes should address higher-dimensionality, more expensive optimization problems [93]. When all necessary calculations in parallel mode are finished, the following command:

\[ \text{delete}(\text{gcp}); \]

shuts down the parallel pool.
SOURCE CODE STATEMENT

All implemented DIRECT-type algorithms (DIRECTGO toolbox) are available at the GitHub repository: https://github.com/blockchain-group/DIRECTGO and can be used under the MIT license. We welcome contributions and corrections to it.

DATA STATEMENT

DIRECTGOLib - DIRECT Global Optimization test problems Library is designed as a continuously growing open-source GitHub repository (https://github.com/blockchain-group/DIRECTGOLib) to which anyone can easily contribute. Therefore, the most recent version is slightly different from the one used in these studies. The exact data underlying this article (DIRECTGOLib v1.0) can be accessed either on GitHub or at Zenodo (connected with GitHub):

- at GitHub: https://github.com/blockchain-group/DIRECTGOLib/tree/v1.0,
- at Zenodo: https://doi.org/10.5281/zenodo.6491863,

and used under the MIT license.

REFERENCES

[1] François Bachoc, Céline Helbert, and Victor Picheny. 2020. Gaussian process optimization with failures: Classification and convergence proof. *Journal of Global Optimization* 78, 3 (2020), 483–506. DOI: https://doi.org/10.1007/s10898-020-00920-0
[2] Chuck A. Baker, Layne T. Watson, Bernard Grossman, William H. Mason, and Raphael T. Haftka. 2001. *Parallel Global Aircraft Configuration Design Space Exploration*. Nova Science Publishers, Inc., 79–96.
[3] M. C. Bartholomew-Biggs, S. C. Parkhurst, and S. P. Wilson. 2002. Using DIRECT to solve an aircraft routing problem. *Computational Optimization and Applications* 21, 3 (2002), 311–323. DOI: https://doi.org/10.1023/A:1013729320435
[4] A. Basudhar, C. Dribusch, S. Lacaze, and S. Missoum. 2012. Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization* 46, 2 (2012), 201–221. DOI: https://doi.org/10.1007/s00158-011-0745-5
[5] E. G. Birgin, C. A. Floudas, and J. M. Martinez. 2010. Global minimization using an augmented Lagrangian method with variable lower-level constraints. *Mathematical Programming* 125, 1 (2010), 139–162. DOI: https://doi.org/10.1007/s10107-009-0264-y
[6] Mattias Björkman and Kenneth Holmström. 1999. Global optimization using the DIRECT algorithm in Matlab. *Advanced Modeling and Optimization* 1, 2 (1999), 17–37. Retrieved from https://www.mat.univie.ac.at/~neum/glopt/mss/BjoeH99.pdf.
[7] Florian Bürgel, Kamil S. Kazimierski, and Armin Lechleiter. 2019. Algorithm 1001: IPscatt-A MATLAB toolbox for the inverse medium problem in scattering. *ACM Transactions on Mathematical Software* 45, 4, Article 45 (Dec. 2019), 20 pages. DOI: https://doi.org/10.1145/3328525
[8] L. C. Cagnina, S. C. Esquivel, and C. A. Coello Coello. 2008. Solving engineering optimization problems with the simple constrained particle swarm optimizer. *Informatica (Ljubljana)* 32, 3 (2008), 319–326.
[9] R. G. Carter, J. M. Gablonsky, A. Patrick, C. T. Kelley, and O. J. Eslinger. 2001. Algorithms for noisy problems in gas transmission pipeline optimization. *Optimization and Engineering* 2, 2 (2001), 139–157. DOI: https://doi.org/10.1023/A:101312310266
[10] Xiaojun Chen and C. T. Kelley. 2016. Optimization with hidden constraints and embedded Monte Carlo computations. *Optimization and Engineering* 17, 1 (2016), 157–175. DOI: https://doi.org/10.1007/s11081-015-9302-1
[11] T. D. Choi, O. J. Eslinger, C. T. Kelley, J. W. David, and M. Etheridge. 2000. Optimization of automotive valve train components with implicit filtering. *Optimization and Engineering* 1, 1 (2000), 9–27. DOI: https://doi.org/10.1023/A:1010071821464
[12] M. Clerc. 1999. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*. Vol. 3, IEEE, Washington, DC, 1951–1957. DOI: https://doi.org/10.1109/CEC.1999.785513
[13] M. F. P. Costa, A. M. A. C. Rocha, and E. M. G. P. Fernandes. 2018. Filter-based DIRECT method for constrained global optimization. *Journal of Global Optimization* 71, 3 (2018), 517–536. DOI: https://doi.org/10.1007/s10898-017-0596-8
[14] Steven E. Cox, Raphael T. Haftka, Chuck A. Baker, Bernard Grossman, William H. Mason, and Layne T. Watson. 2001. A comparison of global optimization methods for the design of a high-speed civil transport. *Journal of Global Optimization* 21, 4 (2001), 415–432. DOI: https://doi.org/10.1023/A:1012782825166

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
[15] John W. David, Carl Tim Kelley, and C. Y. Cheng. 1996. Use of an Implicit Filtering Algorithm for Mechanical System Parameter Identification. Technical Report. SAE Technical Paper.

[16] D. Di Serafino, G. Liuzzi, V. Piccialli, F. Riccio, and G. Toraldo. 2011. A modified dividing RECTangles algorithm for a problem in astrophysics. Journal of Optimization Theory and Applications 151, 1 (2011), 175–190. DOI: https://doi.org/10.1007/s10957-011-9856-9

[17] Sébastien Le Digabel and Stefan M. Wild. 2015. A Taxonomy of Constraints in Simulation-Based Optimization, 1–21. arXiv:1505.07881 Retrieved from https://arxiv.org/abs/1505.07881.

[18] L. Dixon and C. Szegö. 1978. The global optimisation problem: An introduction. In Towards Global Optimization. L. Dixon and G. Szegö (Eds.), Vol. 2, North-Holland Publishing Company, Amsterdam, 1–15.

[19] D. Finkel and C. Kelley. 2004. An Adaptive Restart Implementation of DIRECT. Technical Report CRSC-TR04-30. Center for Research in Scientific Computation, North Carolina State University, Raleigh, 1–16.

[20] D. E. Finkel. 2004. MATLAB Source Code for DIRECT. Retrieved March 22, 2017 from http://www4.ncsu.edu/~ctk/Finkel_Direct/.

[21] D. E. Finkel. 2005. Global Optimization with the DIRECT Algorithm. Ph.D. Dissertation. North Carolina State University.

[22] D. E. Finkel and C. T. Kelley. 2006. Additive scaling and the DIRECT algorithm. Journal of Global Optimization 36, 4 (2006), 597–608. DOI: https://doi.org/10.1007/s10898-006-9029-9

[23] Christodoulos A. Floudas, Panos M. Pardalos, Claire S. Adjiman, William R. Esposito, Zeynep H. Gumus, Stephen T. Harding, John L. Klepeis, Clifford A. Meyer, and Carl A. Schweiger. 1999. Handbook of Test Problems in Local and Global Optimization. Nonconvex Optimization and Its Applications, Vol. 33. Springer, Boston, MA. DOI: https://doi.org/10.1007/978-1-4757-3040-1

[24] J. M. Gablonsky. 2001. Modifications of the DIRECT Algorithm. Ph.D. Dissertation. North Carolina State University.

[25] J. M. Gablonsky and C. T. Kelley. 2001. A locally-biased form of the DIRECT algorithm. Journal of Global Optimization 21, 1 (2001), 27–37. DOI: https://doi.org/10.1023/A:1017930332101

[26] J. Gillard and D. Kvasov. 2017. Lipschitz optimization methods for fitting a sum of damped sinusoids to a series of observations. Statistics and Its Interface 10, 1 (2017), 59–70. DOI: https://doi.org/10.31007/SII.2017.v10.n1.a6

[27] Ratko Grbić, Emmanuel Karlo Nyarko, and Rudolf Scitovski. 2013. A modification of the direct method for Lipschitz global optimization for a symmetric function. Journal of Global Optimization 57, 4 (2013), 1193–1212. DOI: https://doi.org/10.1007/s10898-012-0020-3

[28] Jian He, Alex Verstak, Masha Sosonkina, and Layne T. Watson. 2009. Performance modeling and analysis of a massively parallel DIRECT—Part 1. The International Journal of High Performance Computing Applications 23, 1 (2009), 14–28. DOI: https://doi.org/10.1177/1094342008098463

[29] Jian He, Alex Verstak, Layne T. Watson, and Masha Sosonkina. 2009. Performance modeling and analysis of a massively parallel DIRECT—Part 2. The International Journal of High Performance Computing Applications 23, 1 (2009), 29–41. DOI: https://doi.org/10.1177/10943420080984643

[30] Jian He, Layne T. Watson, Naren Ramakrishnan, Clifford A. Shaffer, Alex Verstak, Jing Jiang, Kyung Bae, and William H. Tranter. 2002. Dynamic data structures for a DIRECT search algorithm. Computational Optimization and Applications 23, 1 (2002), 5–25. DOI: https://doi.org/10.1023/A:1019992822938

[31] Jian He, Layne T. Watson, and Masha Sosonkina. 2009. Algorithm 897: VTDIRECT95: Serial and parallel codes for global optimization for a symmetric function. ACM Transactions on Mathematical Software 36, 3, Article 17 (July 2009), 24 pages. DOI: https://doi.org/10.1145/1527286.1527291

[32] A. Hedar. 2005. Test Functions for Unconstrained Global Optimization. Retrieved March 22, 2017 from http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedard/Hedar_files/TestGO.htm.

[33] Didier Henrion and Jean-Bernard Lasserre. 2003. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. ACM Transactions on Mathematical Software 29, 2 (June 2003), 165–194. DOI: https://doi.org/10.1145/779359.779363

[34] John H. Holland. 1975. Adaptation in Natural and Artificial Systems (2nd ed.). University of Michigan Press, Ann Arbor, MI.

[35] K. Holmstrom, A. O. Goran, and M. M. Edvall. 2010. User’s Guide for TOMLAB 7. Retrieved November 15, 2021 from https://tomopt.com/.

[36] R. Horst, P. M. Pardalos, and N. V. Thoai. 1995. Introduction to Global Optimization. Kluwer Academic Publishers, Berlin.

[37] D. R. Jones. 2001. The DIRECT global optimization algorithm. In The Encyclopedia of Optimization. Christodoulos A. Floudas and Panos M. Pardalos (Eds.), Kluwer Academic Publishers, Dordrecht, 431–440.

[38] Donald R. Jones and Joaquim R. R. A. Martins. 2021. The DIRECT algorithm: 25 years later. Journal of Global Optimization 79, 3 (2021), 521–566. DOI: https://doi.org/10.1007/s10898-020-00952-6

[39] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. 1993. Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Application 79, 1 (1993), 157–181. DOI: https://doi.org/10.1007/BF00941892

ACM Transactions on Mathematical Software, Vol. 48, No. 4, Article 41. Publication date: December 2022.
[40] Moslem Kazemi, G. Gary Wang, Shahryar Rahnamayan, and Kamal Gupta. 2011. Metamodel-based optimization for problems with expensive objective and constraint functions. *Journal of Mechanical Design* 133, 1 (2011), 014505. DOI: https://doi.org/10.1115/1.4003035

[41] J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Proceedings of the International Conference on Neural Networks*. Vol. 4, IEEE, New Jersey, NJ, 1942–1948. DOI: https://doi.org/10.1109/ICNN.1995.488968

[42] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680. DOI: https://doi.org/10.1126/science.220.4598.671

[43] Jing Liang, Thomas Runarsson, Efrén Mezura-Montes, M. Clerc, Ponnuthurai Suganthan, Carlos Coello, and Kalyan Deb. 2006. *Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization*. Technical Report, Nanyang Technological University, Singapore, 251–256.

[44] H. Liu, S. Xu, X. Chen, X. Wang, and Q. Ma. 2017. Constrained global optimization via a DIRECT-type constraint-handling technique and an adaptive metamodeling strategy. *Structural and Multidisciplinary Optimization* 55, 1 (2017), 155–177. DOI: https://doi.org/10.1007/s00158-016-1482-6

[45] H. Liu, S. Xu, X. Wang, X. Wu, and Y. Song. 2015. A global optimization algorithm for simulation-based problems via the extended DIRECT scheme. *Engineering Optimization* 47, 11 (2015), 1441–1458. DOI: https://doi.org/10.1080/0305215X.2014.971777

[46] Qunfeng Liu. 2013. Linear scaling and the DIRECT algorithm. *Journal of Global Optimization* 56, 3 (2013), 1233–1245. DOI: https://doi.org/10.1007/s10898-012-9952-x

[47] Qunfeng Liu and Wanyou Cheng. 2014. A modified DIRECT algorithm with bilevel partition. *Journal of Global Optimization* 60, 3 (2014), 483–499. DOI: https://doi.org/10.1007/s10898-013-0119-1

[48] Qunfeng Liu, Guang Yang, Zhongshi Zhang, and Jinping Zeng. 2017. Improving the convergence rate of the DIRECT global optimization algorithm. *Journal of Global Optimization* 67, 4 (2017), 851–872. DOI: https://doi.org/10.1007/s10898-016-0447-z

[49] Qunfeng Liu, Jinping Zeng, and Gang Yang. 2015. MrDIRECT: A multilevel robust DIRECT algorithm for global optimization problems. *Journal of Global Optimization* 62, 2 (2015), 205–227. DOI: https://doi.org/10.1007/s10898-014-0241-8

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

[51] Giampaolo Liuzzi, Stefano Lucidi, and Veronica Piccialli. 2010. A partition-based global optimization algorithm. *Journal of Global Optimization* 48, 1 (2010), 113–128. DOI: https://doi.org/10.1007/s10898-009-9515-y

[52] Piotr Luszczek. 2009. Parallel programming in MATLAB. *Matlab*. 1008 pages. https://se.mathworks.com/help/releases/R2020b/pdf_doc/parallel-computing/parallel-computing.pdf. (Accessed on 1 September 2020).

[53] Seyedali Mirjalili, SeyedMohammad Mirjalili, and Andrew Lewis. 2014. Grey wolf optimizer. *Advances in Engineering Software* 69, 1(2014), 46–61. DOI: https://doi.org/10.1016/j.advengsoft.2013.12.007

[54] Jonas Mockus, Remigijus Paulavičius, Dainius Rusakevičius, Dmitrij Šešok, and Julius Žilinskas. 2017. Application of reduced-set Pareto-Lipschitzian optimization to truss optimization. *Journal of Global Optimization* 67, 1–2 (2017), 425–450. DOI: https://doi.org/10.1007/s10898-015-0364-6

[55] J. J. Moré and S. M. Wild. 2009. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* 20, 1 (2009), 172–191. DOI: https://doi.org/10.1137/080724083

[56] Jonggeol Na, Younghub Lim, and Chonghun Han. 2017. A modified DIRECT algorithm for hidden constraints in an LNG process optimization. *Energy* 126, 1 (2017), 488–500. DOI: https://doi.org/10.1016/j.energy.2017.03.047

[57] Remigijus Paulavičius, Lakhdar Chiter, and Julius Žilinskas. 2018. Global optimization based on bisection of rectangles, function values at diagonals, and a set of Lipschitz constants. *Journal of Global Optimization* 71, 1 (2018), 5–20. DOI: https://doi.org/10.1007/s10898-016-0485-6

[58] Remigijus Paulavičius, Ya. D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. 2014. Globally-biased DISIMPL algorithm for expensive global optimization. *Journal of Global Optimization* 59, 2–3 (2014), 545–567. DOI: https://doi.org/10.1007/s10898-014-0180-4

[59] Remigijus Paulavičius, Ya. D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. 2020. Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. *Expert Systems with Applications* 144, C (2020), 11305. DOI: https://doi.org/10.1016/j.eswa.2019.113052

[60] R. Paulavičius and J. Žilinskas. 2006. Analysis of different norms and corresponding Lipschitz constants for global optimization. *Technological and Economic Development of Economy* 36, 4 (2006), 383–387. DOI: https://doi.org/10.1080/13928619.2006.9637758
[63] R. Paulavičius and J. Žilinskas. 2007. Analysis of different norms and corresponding Lipschitz constants for global optimization in multidimensional case. Information Technology and Control 36, 4 (2007), 383–387.

[64] R. Paulavičius and J. Žilinskas. 2008. Improved Lipschitz bounds with the first norm for function values over multidimensional simplex. Mathematical Modelling and Analysis 13, 4 (2008), 553–563. DOI: https://doi.org/10.3846/1392-6292.2008.13.553-563

[65] Remigijus Paulavičius and Julius Žilinskas. 2013. Simplicial Lipschitz optimization without the Lipschitz constant. Journal of Global Optimization 59, 1 (2013), 23–40. DOI: https://doi.org/10.1007/s10898-013-0089-3

[66] Remigijus Paulavičius and Julius Žilinskas. 2014. Simplicial Global Optimization. Springer, New York, NY. DOI: https://doi.org/10.1007/978-1-4614-9093-7

[67] Remigijus Paulavičius and Julius Žilinskas. 2016. Advantages of simplicial partitioning for Lipschitz optimization problems with linear constraints. Optimization Letters 10, 2 (2016), 237–246. DOI: https://doi.org/10.1007/s11590-014-0772-4

[68] Remigijus Paulavičius, Julius Žilinskas, and Andreas Grothey. 2010. Investigation of selection strategies in branch and bound algorithm with simplicial partitions and combination of Lipschitz bounds. Optimization Letters 4, 2 (2010), 173–183. DOI: https://doi.org/10.1155/2013/11590-009-0156-3

[69] G. Di Pillo, G. Liuzzi, S. Lucidi, V. Piccialli, and F. Rinaldi. 2016. A DIRECT-type approach for derivative-free constrained global optimization. Computational Optimization and Applications 65, 2 (2016), 361–397. DOI: https://doi.org/10.1007/s10589-016-9876-3

[70] G. Di Pillo, S. Lucidi, and F. Rinaldi. 2010. An approach to constrained global optimization based on exact penalty functions. Journal of Optimization Theory and Applications 54, 2 (2010), 251–260. DOI: https://doi.org/10.1007/s10898-010-9582-0

[71] János D. Pintér. 1996. Global Optimization in Action: Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications. Nonconvex Optimization and Its Applications, Vol. 6, Springer US, Berlin. DOI: https://doi.org/10.1007/978-1-4757-2502-5

[72] S. A. Polyak. 1967. An algorithm for finding the absolute minimum of a function. Theory of Optimal Solutions 12, 4 (1967), 57–67. DOI: https://doi.org/10.10014-5553(72)90115-2 in Russian.

[73] Franco P. Preparata and Michael Shamos. 1985. Computational Geometry, Springer-Verlag New York, Berlin. DOI: https://doi.org/10.1007/978-1-4612-1098-6

[74] Tapabrata Ray and Kim Meow Liew. 2003. Society and civilization: An optimization algorithm based on the simulation of social behavior. IEEE Transactions on Evolutionary Computation 7, 4 (2003), 386–396. DOI: https://doi.org/10.1109/TEVC.2003.814902

[75] L. M. Rios and N. V. Sahinidis. 2007. Derivative-free optimization: A review of algorithms and comparison of software implementations. Journal of Global Optimization 56, 3 (2007), 1247–1293. DOI: https://doi.org/10.1007/s10898-012-9951-y

[76] Ya. D. Sergeyev and Dmitri E. Kvasov. 2007. Global search based on diagonal partitions and a set of Lipschitz constants. SIAM Journal on Optimization 16, 3 (2006), 910–937. DOI: https://doi.org/10.1137/040621132

[77] Ya. D. Sergeyev and D. E. Kvasov. 2008. Diagonal Global Optimization Methods. FizMatLit, Moscow.

[78] Ya. D. Sergeyev and D. E. Kvasov. 2011. Lipschitz global optimization. In Wiley Encyclopedia of Operations Research and Management Science (in 8 volumes). J. J. Cochran, L. A. Cox, P. Keshkinocak, J. P. Kharoufeh, and J. C. Smith (Eds.), Vol. 4, John Wiley & Sons, New York, NY, 2812–2828.

[79] Yaroslav D. Sergeyev and Dimitri E. Kvasov. 2017. Deterministic Global Optimization: An Introduction to the Diagonal Approach. Springer, Berlin. DOI: https://doi.org/10.1007/978-1-4939-7199-2

[80] Yaroslav D. Sergeyev, Dmitri E. Kvasov, and Marat S. Mukhametzhanov. 2016. On the Least-Squares Fitting of Data by Sinusoids. Springer International Publishing, Cham, 209–226. DOI: https://doi.org/10.1007/978-3-319-29975-4_11

[81] B. O. Shubert. 1972. A sequential method seeking the global maximum of a function. SIAM Journal on Numerical Analysis 9, 3 (1972), 379–388. DOI: https://doi.org/10.1137/0709036

[82] D. E. Stoneking, G. L. Bilbro, P. A. Gilmore, R. J. Trew, and C. T. Kelley. 1992. Yield optimization using a GaAs process simulator coupled to a physical device model. IEEE Transactions on Microwave Theory and Techniques 40, 7 (1992), 1353–1363. DOI: https://doi.org/10.1109/22.146318

[83] Linas Stripinis. 2020. Parallel DIRECT-type algorithms for generally constrained global optimization problems in MATLAB, GitHub. Retrieved June 12, 2021 from https://github.com/blockchain-group/pDIRECT-GLe.

[84] Linas Stripinis and Remigijus Paulavičius. 2021. A new DIRECT-GLeh algorithm for global optimization with hidden constraints. Optimization Letters 15, 6 (2021), 1865–1884. DOI: https://doi.org/10.1007/s11590-021-01726-z

[85] Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGO: A new DIRECT-type MATLAB toolbox for derivative-free global optimization, Version v1.0.0, GitHub. Retrieved August 23, 2022 from https://github.com/blockchain-group/DIRECTGO/releases/tag/v1.0.0.
[86] Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGOLib - DIRECT Global Optimization test problems Library, Version v1.0, GitHub. Retrieved August 23, 2022 from https://github.com/blockchain-group/DIRECTGOLib/tree/v1.0.

[87] Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGOLib - DIRECT Global Optimization test problems Library, Version v1.2, GitHub. Retrieved August 23, 2022 from https://github.com/blockchain-group/DIRECTGOLib/tree/v1.2.

[88] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. 2018. Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT. Optimization Letters 12, 7 (2018), 1699–1712. DOI: https://doi.org/10.1007/s11590-017-1228-4

[89] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. 2019. Penalty functions and two-step selection procedure based DIRECT-type algorithm for constrained global optimization. Structural and Multidisciplinary Optimization 59, 6 (2019), 2155–2175. DOI: https://doi.org/10.1007/s11590-018-2181-2

[90] Linas Stripinis and Remigijus Paulavičius. 2020. DIRECTLib – a library of global optimization problems for DIRECT-type methods, Version v1.2, Zenodo. DOI: https://doi.org/10.5281/zenodo.3948890 (Accessed on April 28, 2022).

[91] Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGOLib - DIRECT Global Optimization test problems Library, Version v1.2, Zenodo. DOI: https://doi.org/10.5281/zenodo.6617799 (Accessed on August 23, 2022).

[92] Linas Stripinis and Remigijus Paulavičius. 2022. DIRECTGOLib - DIRECT Global Optimization test problems Library, Version v1.0, Zenodo. DOI: https://doi.org/10.5281/zenodo.6491863 (Accessed on August 23, 2022).

[93] Linas Stripinis, Julius Žilinskas, Leocadio G. Casado, and Remigijus Paulavičius. 2021. On MATLAB experience in accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization. Applied Mathematics and Computation 390, 4 (2021), Article 125596, 1–17. DOI: https://doi.org/10.1016/j.amc.2020.125596

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

[95] S. Surjanovic and D. Bingham. 2013. Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved March 22, 2017 from http://www.sfu.ca/~ssurjano/index.html.

[96] Ye Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin. 2017. PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]. IEEE Computational Intelligence Magazine 12, 4 (2017), 73–87. DOI: https://doi.org/10.1109/MCI.2017.2742868

[97] A. I. F. Vaz and L. N. Vicente. 2009. Pswarm: A hybrid solver for linearly constrained global derivative-free optimization. Optimization Methods and Software 24, 4–5 (2009), 669–685. DOI: https://doi.org/10.1080/10556780902909948

[98] L. T. Watson and C. A. Baker. 2001. A fully-distributed parallel global search algorithm. Engineering Computations 18, 1/2 (2001), 155–169. DOI: https://doi.org/10.1108/02644400110365851

Received 24 April 2021; revised 16 August 2022; accepted 22 August 2022