https://doi.org/10.15388/vu.thesis.138
https://orcid.org/0000-0001-9680-5847

VILNIUS UNIVERSITY

Linas
STRIPINIS

Improvement, development and implementation of derivative-free global optimization algorithms

DOCTORAL DISSERTATION

Natural Sciences,
Informatics (N 009)

Vilnius, 2021

This dissertation was written between 2016 and 2020 at Vilnius University.

Academic supervisor:

Prof. Dr. Remigijus Paulavičius (Vilnius University, Natural Sciences, Informatics – N 009).

https://doi.org/10.15388/vu.thesis.138
https://orcid.org/0000-0001-9680-5847

VILNIAUS UNIVERSITETAS

Linas
STRIPINIS

Globaliojo optimizavimo algoritmų, nereikalaujančių išvestinių informacijos, kūrimas, tobulinimas ir realizacija

DAKTARO DISERTACIJA

Gamtos mokslai,
Informatika (N 009)

Vilnius, 2021

Disertacija rengta 2016 – 2020 metais Vilniaus universitete.

Mokslinis vadovas:

Prof. Dr. Remigijus Paulavičius (Vilniaus Universitetas, gamtos mokslai, informatika – N 009).

ACKNOWLEDGMENTS

I want to express the deepest appreciation to my scientific supervisor Prof. Dr. Remigijus Paulavičius, as this work would not have been possible without his guidance and support.

I want to thank the co-authors of the research carried out during the studies, especially Prof. Dr. (HP) Julius Žilinskas, for granted knowledge and cooperation.

I am grateful to both dissertation reviewers, Prof. Habil. Dr. Gintautas Dzemyda and Prof. dr. Dmitrij Šešok, who carefully read the dissertation and provided valuable advice and critical remarks, which helped improve the manuscript's final quality.

I sincerely thank my family members and friends for their support, even in the most challenging moments, which motivates me to move forward and not give up.

SUMMARY

Due to its simplicity and efficiency, the derivative-free global-search DIRECT (DIviding RECTangles) algorithm has received much consideration from the optimization community, and various novel ideas and extensions have been proposed. However, the efficiency of many DIRECT-type algorithms solving multimodal problems and also in cases when a solution with high accuracy is required has decreased. This thesis presents the new scheme for selecting potentially optimal hyper-rectangles in the DIRECT framework, which addresses two of its weaknesses. An extensive experimental investigation revealed the potential and competitiveness of the added enhancements in our recent proposals, especially for more challenging multi-modal optimization problems.

Unfortunately, the original DIRECT algorithm addresses optimization problems only with bounds on the variables, and due to it, the application of the algorithm is limited, as various applied optimization problems often hold other types of constraints. The initial DIRECT extensions for problems with general constraints were not competitive, compared with other derivative-free global optimization methods. Only in recent years, a few promising DIRECT-type modifications were proposed. In this thesis, two different constraint handling techniques are presented, and one of these strategies can even be applied to solve problems with hidden constraints. The proposed algorithms effectively explore hyper-rectangles with infeasible midpoints close to the boundaries of feasibility and may cover feasible regions. An extensive experimental investigation revealed the potential of the proposed approaches compared with other existing DIRECT-type algorithms for constrained global optimization problems, including important engineering issues.

Contemporary problems often can not be solved with algorithms reasonably fast using a single core on the fastest computers. However, most DIRECT-type algorithms present challenges for efficient parallel implementation, and only a few parallel versions of DIRECT are known. To the best of our knowledge, all the existing parallel DIRECT-type versions are focused on box-constrained global optimization problems. Since the newly

proposed selection scheme per iteration selects a larger number of subdividing regions, the algorithms developed in this thesis look more promising for parallelization than DIRECT. Therefore, the first parallel DIRECT-type algorithms for constrained global optimization problems are also introduced in this thesis.

SANTRAUKA

Dėl paprastumo ir efektyvumo, išvestinių informacijos nereikalaujantis globalios paieškos DIRECT (DIviding RECTangles) algoritmas sulaukė daug optimizavimo bendruomenės dėmesio, buvo pasiūlytos įvairios naujos algoritmo idėjos ir modifikacijos. Tačiau daugelio DIRECT tipo algoritmų efektyvumas blogėja sprendžiant daugybę lokalių minimumų turinčias problemas arba kai norima rasti kur kas tikslesnę sprendinio reikšmę. Siekiant sumažinti algoritmo trūkumus šis darbas pateikia naują potencialiai optimalių hiper-stačiakampių atrankos schemą DIRECT tipo algoritmuose. Išsamus eksperimentinis tyrimas atskleidė sukurto algoritmo potencialą, ypač sprendžiant sudėtingesnius optimizavimo uždavinius.

Tačiau originalus DIRECT algoritmas sprendžia problemas tik su intervaliniais ribojimais ir dėl šios priežasties algoritmo pritaikomumas yra labai ribotas, kai praktinės optimizavimo problemos dažnai yra apribotos įvairaus tipo ribojimų funkcijomis. Pirmųjų algoritmo modifikacijų rezultatai nebuvo konkurencingi, palyginus su kitais išvestinių nereikalaujančiais optimizavimo metodais. Tik pastaraisiais metais buvo pasiūlyta keletas perspektyvių DIRECT tipo versijų uždaviniams su ribojimais spręsti. Šiame darbe pateikiami du skirtingi metodai uždaviniams su ribojimais spręsti, o vieną iš jų galima pritaikyti sprendžiant problemas su paslėptais apribojimais. Išsamūs eksperimentiniai tyrimai atskleidė siūlomų algoritmų potencialumą ir efektyvumų, palyginus su kitais egzistuojančiais DIRECT tipo algoritmais sprendžiant globaliojo optimizavimo problemas su įvairaus tipo ribojimais, įskaitant svarbias praktines problemas.

Šiuolaikinės problemos dažnai negali būti efektyviai išspręstos naudojant ir pačius greičiausius nuoseklius kompiuterius. Todėl lygiagretinimo technologijos galėtų išspręsti kylančią problemą. Tačiau iteracinė DIRECT tipų metodų prigimtis riboja algoritmo galimybes efektyviam lygiagretinimui, ir yra žinomos tik kelios lygiagrečios DIRECT versijos. Mūsų žiniomis, visos esamos lygiagrečios DIRECT tipo versijos yra skirtos globaliojo optimizavimo problemoms su intervaliniais ribojimais spręsti. Kadangi sukurta potencialiai optimali hiper-stačiakampių schema, atlieka atranką du kartus per iteraciją ir pasirenkamas didesnis skaičius dalinamų

sričių, algoritmai sukurti šiame darbe atrodo perspektyvesni lygiagretinti, palyginus su pirmine DIRECT algoritmo versija. Šiame darbe pristatomi pirmieji lygiagretūs DIRECT tipo algoritmai, skirti globaliojo optimizavimo problemoms su įvairaus tipo apribojimais spręsti.

NOTATION AND ABBREVIATIONS

f	objective function;
$\mathbf{g}$	vector of inequality constraint functions;
$\mathbf{h}$	vector of equality constraint functions;
φ	sum of constraint violations;
u	number of inequality constraint functions g;
v	number of equality constraint functions h;
m	number of linear constraint functions;
n	number of variables in objective function;
$\mathbb{R}^n$	n-dimensional Euclidean space;
Ω	n-dimensional region that satisfy all the constraint functions $\Omega \subset \mathbb{R}^n$;
D	hyper-rectangular optimization domain $D \subset \mathbb{R}^n$;
$\mathbf{a}$	lower bound of domain D (vector);
$\mathbf{b}$	upper bound of domain D (vector);
δ	measure(size) of the hyper-rectangle;
$\mathbf{x}$	variable(vector);
f^*	global optimum;
$\mathbf{x}^*$	variable(vector) of global optimum f^*;
f_{min}	smallest curent value of the objective function f;
$\mathbf{x}_{\text{min}}$	point of the f_{min};
f_{max}	largest curent value of the objective function f;
$f_{\text{min}}^{\text{feas}}$	smallest curent feasible value of the objective function f;
$\mathbf{x}_{\text{min}}^{\text{feas}}$	point of the $f_{\text{min}}^{\text{feas}}$;
ξ, ϕ	auxiliary functions;
r	penalty parameter;
$\bar{\cdot}$	normalized value;
$\text{card}(\cdot)$	cardinality of set;
$\mathbf{c}$	center point of normalized hyper-rectangle $\bar{D}$ (vector);
d	Euclidean distance;
I^n	identity matrix;
$\mathcal{H}$	set of all hyper-rectangles(partition);
$\mathbb{I}$	index set of hyper-rectangles;

$\tilde{L}$	Lipschitz constant;
ε	small error value $(0 \leq \varepsilon)$;
$\|\mathbf{x}\|_i$	The i-norm of n-dimensional vector $\mathbf{x}$;
T	time;
F	parallel efficiency;
S	speed-up ratio;
W	worker(slave lab);
ϖ	number of workers(slave labs);
$\mathcal{P}$	problem;
K, k	algorithmic iteration counters;
FE, fe	algorithmic function evaluation counters;
pe	percent error;
λ	performance ratio;
L	linear constraints;
NL	nonlinear constraints;
EQ	equality constraints;
f_{eval}	number of function evaluations;
POH	Potential optimal hyper-rectangle;
s	seconds;

Contents

Chapter 1

INTRODUCTION

1.1 Research Context And Motivation

The DIRECT algorithm [46] is well-known and broadly used for derivative-free global optimization problems. This algorithm is an extension to classical Lipschitz optimization [69, 70, 84, 85, 95, 106], where the Lipschitz constant is not assumed to be known. This feature made DIRECT-type approaches attractive for various real-world optimization problems [1, 2, 8, 12, 14, 26, 56, 73, 77]. Furthermore, the recent extensive study [91] on the large collection of optimization test problems showed that, on average, the performance of DIRECT-type algorithms is one of the best amongst all the tested state-of-the-art derivative-free global optimization approaches. The DIRECT-type algorithms outperformed algorithms belonging to other well-known optimization such classes, as Genetic [44], Simulated annealing [50], and Particle swarm optimization [48].

Nevertheless, earlier research has revealed that the DIRECT algorithm has two weaknesses [45, 72, 73, 123]. The first is delaying the discovery of the global minimum and the second is the slow fine-tuning of the solution with high accuracy, mainly in optimization problems with many local minima. Another disadvantage is that the originally published algorithm solves only box-constrained global optimization problems, and it does not naturally address other types of constraints. A less studied field in DIRECT is applying the algorithm for problems with general or even more complex hidden

constraints. Therefore, the main research objectives of this thesis are to develop algorithms that can address its weaknesses and solve problems with various types of constraints within the DIRECT framework.

Lipschitz-type global optimization methods are computationally intensive [41, 42, 67, 70, 77, 98, 102] and therefore a natural way to speed it up is to parallelize them [32, 75, 80, 92, 93, 99, 113, 112, 114]. However, the iterative nature of partitioning-based DIRECT-type algorithms limit possibilities for effective parallelism [25, 34, 35, 36, 37, 81, 122]. Only a few parallel versions of DIRECT-type algorithms, mainly by the same group of researchers have been developed. Most of them focused on improving parallel efficiency by creating more computations in each iteration using an "aggressive" version of DIRECT, which relaxes the selection criteria by picking and subdividing a hyper-rectangle of every diameter in every iteration [1]. From the optimization point of view, this approach is not appealing because it examines many "unnecessary" (non-potentially optimal) hyper-rectangles [18, 25]. Moreover, all the existing parallel DIRECT-type versions are focused on box-constrained global optimization problems to the best of our knowledge. Therefore, another important issue investigated in this thesis is the development of efficient parallel DIRECT-type algorithms for generally constrained global optimization problems.

1.2 Object of the Thesis

The object of the thesis is sequential and parallel DIRECT-type algorithms for global optimization problems with general constraints and their applicability for practical optimization problems.

1.3 Aims and Tasks of the Research

The aims of the thesis are:

1. To increase efficiency of state-of-the-art DIRECT-type global optimization algorithms on optimization problems with many local minima, and where the solution with high accuracy is needed;

2. To extend DIRECT-type algorithms on global optimization problems with general and hidden constraints;
3. To develop efficient open-source derivative-free algorithms by taking into account algorithmic improvements, efficient data structures, and parallelization techniques.

In order to achieve these aims, the following research tasks must be accomplished:

1. To evaluate the performance of the existing state-of-the-art DIRECT-type global optimization algorithms and determine their weaknesses;
2. To improve existing and develop new algorithms considering identified drawbacks;
3. To develop a general constraint-handling strategy in the DIRECT algorithmic framework;
4. To develop an auxiliary functions-based DIRECT-type algorithm for optimization problems with hidden constraints;
5. To implement efficient sequential and parallel versions of the proposed algorithms, and compare their performance to other related approaches;
6. To efficiently solve challenging practical (potentially black-box) optimization problems using implemented and openly accessible tools.

1.4 Research Methodology

To analyse the received scientific results in global optimization and parallelization fields, algorithmic theory, convergence analysis, parallel computing theory, information retrieval, organization, analysis, comparative analysis, and generalization methods have been used. For the interpretation of the experimental investigation, statistical analysis and performance profiles [15] were applied to evaluate the algorithms' efficiency.

1.5 Scientific Novelty of the Work

The main novelties of this dissertation are the following:

1. Based on theoretical and experimental research, it was revealed that the efficiency of the original DIRECT algorithm deteriorates on

optimization problems with many local minima and in cases where the solution with high accuracy is needed. To overcome these shortcomings, a new strategy for selecting potentially optimal hyper-rectangles was developed, and the proposed scheme does not require any extra parameters or use of local search subroutines (DIRECT-GL, DIRECT-G and DIRECT-L). This modification significantly outperforms the original DIRECT version, solving much more complicated multi-modal and high dimension optimization problems, and can increase the accuracy of the final solution using fewer function evaluations.

2. The original DIRECT algorithm addresses optimization problems only with bounds on the variables. In this thesis the following algorithmic extensions for problems with other types of constraints were developed:
   2.1. New approaches (DIRECT-GLc, DIRECT-GLce and DIRECT-GLce-min) for general-constrained global optimization problems were proposed. The algorithm works in two phases. During the first phase, the algorithm handles infeasible initial points using constraint function information. The second phase uses an auxiliary function approach that combines information on the objective and constraint functions and seeks to find a feasible global solution.
   2.2. An approach (DIRECT-GLh) for problems with hidden constraints (can also be used for problems with general constraints) were proposed. The algorithm works in two phases. During the first phase, the algorithm handles infeasible initial points uniformly dividing the hyper-rectangle. The second phase uses an auxiliary function approach that estimates the necessary penalty values for the infeasible points and seeks to find a feasible global solution.
3. Instead of traditionally used static data structures, the developed algorithms were implemented using more effective dynamic data structures. Moreover, for the first time, parallel DIRECT-type implementations (pD-GLce-parfor, pD-GLce-spmd and pD-ACe-spmd) for problems with general constraints were introduced.
4. DIRECTlib - a library of box and generally constrained test and

practical engineering global optimization problems for benchmarking of various DIRECT-type algorithms was created:
Stripinis, L. and Paulavičius, R. DIRECTLib – a library of global optimization problems for DIRECT-type methods, v1.2 (2020). 10.5281/zenodo.3948890;
URL: https://zenodo.org/record/3948890#.XyVOk5dR2Uk

The developed algorithms were implemented in the MatLab software:

- For box-constrained global optimization problems, two implementations of sequential and parallel DIRECT-GL versions, also four intermediate DIRECT-G and DIRECT-L versions;
  URL: https://github.com/LinasStripinis/DIRECT-GL
- For general-constrained global optimization problems, two implementations of the sequential and parallel DIRECT-GLce versions, also two intermediate DIRECT-GLc and DIRECT-GLce-min versions;
  URL 1: https://github.com/LinasStripinis/DIRECT-GL
  URL 2: https://github.com/blockchain-group/pDIRECT-GLce
- For global optimization problems with hidden constraints, implementation of sequential DIRECT-GLh version;
  URL: https://github.com/LinasStripinis/DIRECT-GL

1.6 Participation in Scientific Programs

The author participated in the research project "Development and Applications of Bilevel Optimization Algorithms" funded by the Lithuanian Research Council (2017 - 2020). (No. P-MIP-17-60)

1.7 Defended Statements

1. A new two-step selection strategy-based algorithm (DIRECT-GL) performs the best among DIRECT-type algorithms for the most challenging non-convex problems from DIRECTlib, and has the fastest fine-tuning solution to a high accuracy.
2. A new auxiliary function-based algorithm (DIRECT-GLce) for problems with general constraints is on average the most efficient

DIRECT-type method solving the most complex classes of optimization problems from DIRECTlib: high dimensional; non-linear constrained; and equality constrained.

3. A new auxiliary function-based algorithm for practically oriented (potentially black-box) problems with hidden constrains (DIRECT-GLh) is the most efficient DIRECT-type solver with respect to the number of function evaluations and the execution time solving optimization problems from DIRECTlib.
4. Created dynamic data structures in the developed algorithms reduce the number of necessary calculations and simplify the selection of potential optimal hyper-rectangles; this way it increases the speed of implemented algorithms more than twice.
5. Created the master-slave paradigm based SPMD-type parallel algorithm for problems with general constraints enables preserving the determinism, and a good parallel efficiency on multi-core computing systems was achieved.

1.8 Approbation of the Research

The results of this research were presented at the following conferences plenary sessions:

1. Paulavičius, R., Stripinis, L. and **Žilinskas**, J. "DIRECT-type algorithms for constrained global optimization", EUROPT 2017: 15th EUROPT Workshop on Advances in Continuous Optimization, July 12-14, 2017. Montreal, Canada.
2. **Stripinis, L.**, Žilinskas, J. and Paulavičius, R. "Improved DIRECT-type algorithm for constrained global optimization problems", EUROPT 2018: 16th EUROPT Workshop on Advances in Continuous Optimization, July 12-13, 2018. Almeria, Spain.
3. **Stripinis, L.** Paulavičius, R., and Žilinskas, J. "Importance of optimization techniques for the social sciences", The International EURO mini Conference Modelling and Simulation of Social-Behavioural Phenomena in Creative Societies, September 18–20, 2019. Vilnius, Lithuania.

The results of this research were presented at the following conferences poster sessions:

4. **Stripinis, L.** and Paulavičius, R. "Improved DIRECT-type algorithms for generally constrained global optimization problems", 9th International workshop on Data Analysis Methods for Software Systems (DAMSS), November 30 - December 2, 2017. Druskininkai, Lithuania.
5. **Stripinis, L.** and Paulavičius, R. "Improved DIRECT-type algorithms for generally constrained global optimization problems", 10th International workshop on Data Analysis Methods for Software Systems (DAMSS), November 29 – December 1, 2018. Druskininkai, Lithuania.

1.9 Structure of the Dissertation

The dissertation consists of an introduction, five chapters, bibliography and the publications list. The total scope of the dissertation is 154 pages, including 27 figures and 17 tables. The dissertation was based on 125 literature sources.

Chapter 1 describes the research context, presents the problem statement, discusses the motivation, aims, objectives of the research states, research questions, describes the research methods, and approbation of the research. Chapter 2 gives theoretical backgrounds of the DIRECT-type algorithms. Chapter 3 develops an extension of the DIRECT algorithm for box-constrained global optimization. Chapter 4 proposes an extension of the DIRECT-GL algorithm for general-constrained global optimization. Chapter 5 develops an extension of the DIRECT-GL algorithm for problems with hidden constraints. Chapter 6 discusses ways to speed up the DIRECT-type algorithms and develop the first parallel DIRECT-type algorithm for general-constrained problems. And finally, at the end of the dissertation, the main results of this thesis are summarized.

Chapter 2

Analytical part

The simplicity and efficiency of the original DIRECT algorithm by Jones et al. [45, 46] attracted considerable research interest, and many modifications and extensions have been proposed [1, 6, 20, 26, 30, 46, 53, 55, 58, 59, 64, 65, 71, 72, 73, 76, 77, 78, 97, 107, 108, 109]. DIRECT is a popular partitioning-based Lipschitz optimization algorithm [41, 70, 74, 77, 79, 112] extending the ideas of Piyavskii algorithm [85] (independently rediscovered also by Shubert [106]) to multidimensional derivative-free optimization.

The DIRECT algorithm seeks a global optimum by partitioning potentially optimal (the most promising) hyper-rectangles and evaluating the objective function at these hyper-rectangles centers. The DIRECT algorithm converges to the globally optimal function value if the objective function is continuous or at least continuous in the neighborhood of a global optimum. When the number of iterations goes to infinity, the set of points sampled by an algorithm form a dense subset of the hyper-rectangle.

As a direct search method, DIRECT produces deterministic results without derivative information or the Lipschitz constant of the objective function. Consequently, DIRECT-type methods have been successfully used in solving various multidisciplinary optimization problems [1, 2, 8, 66, 109, 125].

2.1 Global optimization

Traditionally, global optimization is a branch of applied mathematics and numerical analysis that attempts to find the functions of n continuous variables called the objective function $f : \mathbb{R}^n \to \mathbb{R}$ minimum (or maximum):

$$f^* = \min_{\mathbf{x}\in\Omega} \quad f(\mathbf{x}), \text{or} \quad f^* = \max_{\mathbf{x}\in\Omega} \quad f(\mathbf{x}) \tag{2.1}$$

where Ω is an n-dimensional feasible region $\Omega \subset \mathbb{R}^n$. Most of the time, the global optimization problem is defined only as a minimization problem because the maximization of the real-valued function $\hat{f}(\mathbf{x})$ is equivalent to the minimization of the function:

$$f(\mathbf{x}) \equiv (-1) \times \hat{f}(\mathbf{x}).$$

Besides global minimum f^* one or all global minimizers $\mathbf{x}^* f(\mathbf{x}^*) = f^*$ suppose to be founded.

2.1.1 Classification of optimization problems

The classification of optimization problems is not a formal matter, and several different classifications can be found in the literature. In [16], the authors classified optimization problems by the following criteria:

Classification by the objective function f:

- Objective function f is linear;
- Objective function f is non-linear:
  - f is differentiable, or non-differentiable;
  - f is convex, or non-convex;
  - f is uni-modal, or multi-modal;
  - f is quadratic.
- Objective function f is noisy;
- Multiple objective functions involved $\mathbf{f} = (f_1, f_2, ..., f_k)$.

Classification by the argument $\mathbf{x}$:

- One-dimensional problem $\{x \in \Omega, \Omega \subset \mathbb{R}\}$;
- Multi-dimensional problem $\{\mathbf{x} \in \Omega, \Omega \subset \mathbb{R}^n\}$;
- Some or all variables are discrete $\{\mathbf{x} \in \Omega, \Omega \subset \mathbb{Z}^n\}$.

Classification by the type of constraints:

- Only bounds on variables: $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$;
- Linear inequality $\mathbf{g}(\mathbf{x}) \leq 0$ and/or equality $\mathbf{h}(\mathbf{x}) = 0$ constraints;
- Non-linear inequality $\mathbf{g}(\mathbf{x}) \leq 0$ and/or equality $\mathbf{h}(\mathbf{x}) = 0$ constraints;
- Includes hidden constraints.

In most cases, it is not possible to predict a function's behavior in the optimization domain Ω, where it may be more than one local optimum in f. Many optimization methods that have found local optimum cannot guarantee that it is also global. Methods that guarantee the globality of optimum must be based on additional assumptions of the objective function f.

2.1.2 Classification of global optimization methods

Many different methods have been proposed for solving global optimization problems. In [118], the authors have summarized and classified global optimization methods into the following groups:

- Direct search methods:
  - random search methods;
  - clustering methods;
  - generalized descent methods.
- Indirect search methods:
  - methods approximating the level sets;
  - methods approximating the objective function.
- Methods with guaranteed accuracy:
  - covering methods.

In this thesis, the covering methods for global optimization are considered. These methods use different partition techniques of the feasible region. The partitioning is not terminated until some stopping criteria have been met.

Theoretically, the covering methods can solve some class global optimization problems with guaranteed accuracy. Detected subregions that do not contain the global minimum are discarded from further search. Covering methods can ensure that a point $\hat{\mathbf{x}}$ is found such that $f(\hat{\mathbf{x}})$ differs from f^* by no more than a specified accuracy ϵ.

2.2 Overview of the original DIRECT algorithm

The original DIRECT algorithm addresses optimization problems only with bounds on the variables:

$$\min_{\mathbf{x}\in D} \quad f(\mathbf{x}) \tag{2.2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ denotes the objective function and the feasible region is an n-dimensional hyper-rectangle

$$D = [\mathbf{a}, \mathbf{b}] = \{\mathbf{x} \in \mathbb{R}^n : a^j \leq x^j \leq b^j, j = 1, \ldots, n\}.$$

We also assume, that the objective function $f(\mathbf{x})$ is Lipschitz-continuous, but can be non-linear, non-differentiable, non-convex, and multi-modal.

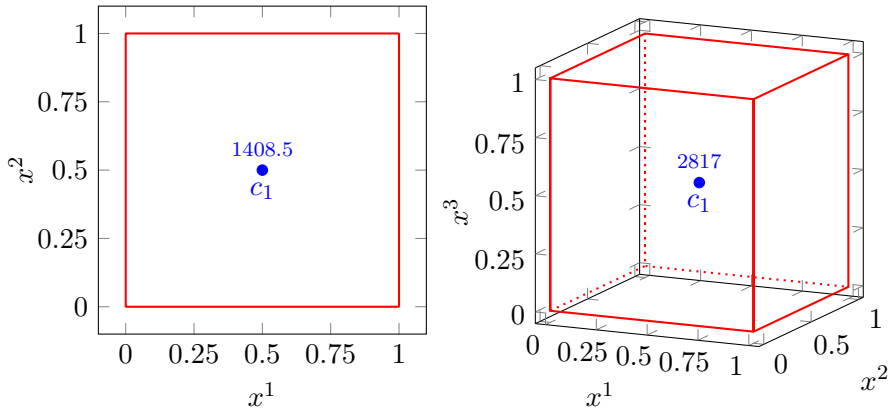

Figure 2.1: The initial evaluation at the midpoint of hyper-rectangle $\bar{D}$ solving *Rosenbrock* test function with $n = 2, 3$

In the first iteration, DIRECT has only one hyper-rectangle, which size is equal to all design space D. Algorithms normalize the domain D to be the unit hyper-rectangle $\bar{D}$ and refer to original space D only when evaluating the objective function. Regardless of the dimension, n, all calculations in the initially published DIRECT algorithm starts from the single midpoint of the hyper-rectangle $\bar{D}$. Visual representation of DIRECT algorithm initialization solving two and three-dimensional problems are given in Fig. 2.1. The first evaluation at objective function will be made at the midpoint $\mathbf{c}_1 = (\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})$ of unit hyper-rectangle $\bar{D}$. The authors in [46] proposed using hyper-rectangular partitions based on N-dimensional trisection, which

can decrease the computational cost by evaluating objective function only in the midpoints of new hyper-rectangles. The first hyper-rectangle $\bar{D}$ is trisected along all its longest dimensions as shown in Fig. 2.2, and new points are calculated $\mid 1/3I^n \pm \mathbf{c}_1 \mid$, where I^n is identity matrix and objective function is evaluated at all new $\mathbf{c}_j$ points, where $j = 2...2n$. Further, $\bar{D}$ partitioned into $2n + 1$ smaller non-intersecting different measure hyper-rectangles, and the strategy which is proposed in [46] suggest to place the lowest function values in the largest measure hyper-rectangles.

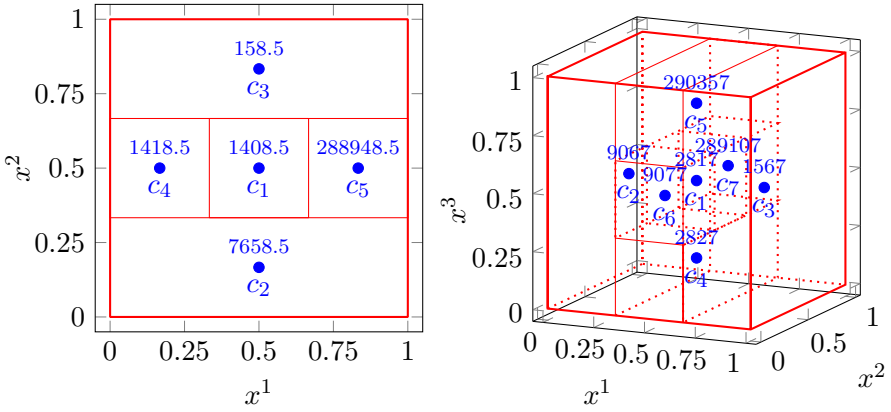

Figure 2.2: The division technique of hyper-rectangles in the DIRECT algorithm solving *Rosenbrock* test function with $n = 2, 3$

The essential step in DIRECT algorithms is the identification of potential optimal hyper-rectangles of the current partition, which at the iteration k defined as

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

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

The selection procedure at the initial step is trivial; there exist only one candidate $\bar{D}_1$. To make the selection of potentially optimal hyper-rectangles in the future iterations, DIRECT assesses the goodness based on the lower bound estimates for the objective function $f(\mathbf{c}_i)$ over each hyper-rectangle $\bar{D}_i^k$. The

requirement of potential optimality is stated formally in Definition 1.

Definition 1 (Potentially optimal hyper-rectangle) *Let* $\mathbf{c}_i$ *denote the center sampling point and* δ_i *be a measure (distance, size) of the hyper-rectangle* $\bar{D}_i^k$. *Let* $\varepsilon_{\text{poh}} > 0$ *be a positive constant and* $f_{\min}$ *be the best currently known value of the objective function. A hyper-rectangle* $\bar{D}_j^k, j \in \mathbb{I}^k$ *is said to be potentially optimal if there exists some rate-of-change (Lipschitz) constant* $\tilde{L} > 0$ *such that*

$$f(\mathbf{c}_j) - \tilde{L}\delta_j \leq f(\mathbf{c}_i) - \tilde{L}\delta_i, \quad \forall i \in \mathbb{I}^k, \tag{2.3}$$

$$f(\mathbf{c}_j) - \tilde{L}\delta_j \leq f_{\min} - \varepsilon_{\text{poh}}|f_{\min}|, \tag{2.4}$$

where the measure of the hyper-rectangle is

$$\delta_i = \frac{1}{2}\|\mathbf{b} - \mathbf{a}\|_2. \tag{2.5}$$

The hyper-rectangle $\bar{D}_j^k$ is potentially optimal if the lower Lipschitz bound for the objective function computed by the left-hand side of (2.3) is the smallest one with some positive constant $\tilde{L}$ among the hyper-rectangles of the current partition $\mathcal{H}^k$. In (2.4) the parameter ε_{poh} is used to protect from an excessive refinement of the local minima [46, 72]. Good results for ε_{poh} values ranging from 10^{-3} to 10^{-7} were obtained in [46]. A geometrical interpretation of selection procedure is shown in Fig. 2.3.

In each iteration `DIRECT`-type algorithms perform selection of such potentially optimal hyper-rectangles, which in subsequent steps are evaluated and divided. A brief description of the main steps of the `DIRECT` algorithm is given in Algorithm 1.

2.2.1 `DIRECT` algorithm for box-constrained global optimization

Two most broadly used public `MatLab` implementations of the original `DIRECT` are `DIRECT Version 4.0` [18] and `glbSolve` [6]. To store information, both implementations employing a vector-based static data memory management, contrary to a tree-structure in the original paper [46], are used. The biggest difference between both algorithms is how the selection

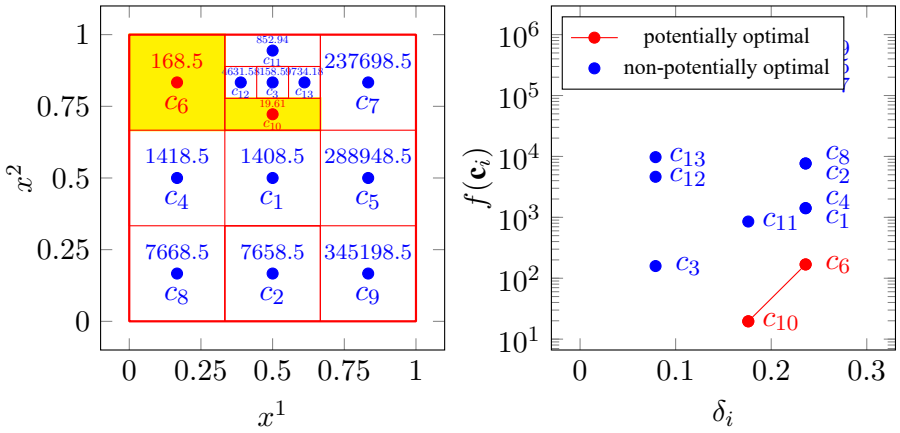

Figure 2.3: Visualization of the selected potentially optimal rectangles in the fourth iteration of the DIRECT algorithm solving *Rosenbrock* test function with $n = 2$

of potentially optimal hyper-rectangles (the lower convex hull, shown on the right side of Fig. 2.3) is implemented. The modified Graham's scan algorithm [87] is used for this in `glbSolve`, but Definition 1 is used in `DIRECT Version 4.0`. Also, the numerical tolerances used in the implementations play an important role [6]. The efficiency of the two different original DIRECT algorithm implementations (DIRECT by Jones [46] and `glbSolve`) is evaluated on nine standard test problems for box-constrained global optimization in [6], and the `glbSolve` implementation using `MatLab` software, outperforms the primer version by Jones in most cases.

Many different DIRECT algorithm extensions have been suggested, and most of them focused on improving the selection of potential optimal hyper-rectangles, while other attempts modified partitioning and sampling strategies. Most of the DIRECT-type algorithms are based on the trisection of n-dimensional potential optimal hyper-rectangles and just BIRECT, and both *DISIMPL* versions use different partitioning strategies.

The BIRECT (BIsecting RECTangles) [71] algorithm is motivated by the diagonal partitioning strategy [94, 97, 102]. The bisection is more appropriate to the trisection because of hyper-rectangles shapes, but usual sampling strategies at the center or the diagonal's endpoints are not efficient

Algorithm 1: The main steps of the DIRECT algorithms

1 **Initialization**. Normalize the search space D to be the unit hyper-rectangle $\bar{D}$, but refer to the original space D when making function calls. Evaluate the objective f at the center point $\mathbf{c}_1$. Set $f_{\min} = f(\mathbf{c}_1)$, $\mathbf{x}_{\min} = \mathbf{c}_1$, and initialize algorithmic performance measures and *stopping criteria*.

2 **while** *stopping criteria are not satisfied* **do**

3 **Selection**. *Identify* the set $\mathbb{S}$ of potentially optimal hyper-rectangles (subregions of $\bar{D}$) using Definition 1.

4 **Sampling**. For each hyper-rectangle $\bar{D}_j \in \mathbb{S}$ *sample* and *evaluate* the objective function at new domain points. Update $f_{\min}$, $\mathbf{x}_{\min}$, and algorithmic performance measures.

5 **Subdivision**. For each hyper-rectangle $\bar{D}_j \in \mathbb{S}$ subdivide (trisect) and update partitioned search space information.

6 **end**

7 **return** $f_{\min}$, $\mathbf{x}_{\min}$, and the algorithmic performance measures.

for bisection. In BIRECT, the objective function is evaluated at two points on the diagonal equidistant between themselves and a diagonal's vertices. Such a sampling strategy entitles the reuse of the sampling points in descendant hyper-rectangles. Additionally, more comprehensive information about the objective function is considered compared to the central sampling strategy used in most DIRECT-type algorithms.

In the DISIMPL [76] algorithm simplicial partitions are used. At the initial DISIMPL step, combinatorial vertex triangulation [117, 124] of $\bar{D}$ into $n!$ simplices is used. After this, all simplices share the diagonal of the feasible region and have equal hyper-volume. In [76], two different sampling strategies were proposed, both are included in the toolbox: i) DISIMPL-C evaluates the objective function at the geometric center of the simplex; ii) DISIMPL-V evaluates the objective function on all unique vertices of the simplex. For box-constrained problems, the number of initial simplices increases speedily; therefore, DISIMPL effectively can be used only for small dimensional box-constrained global optimization problems.

In [1], the authors relaxed the criteria for selecting potential optimal hyper-rectangles and proposed an "aggressive" version of the algorithm. The main idea of `Aggressive DIRECT` is to select and divide at least one hyper-rectangle from each group of different diameters (δ_i) having the lowest function value. In every iteration, the "aggressive" version of an algorithm produces new hyper-rectangle measurements (δ_i), and the number of function evaluations grows drastically with each iteration. 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. A massive supercomputer may be required to overcome the high cost of later iterations for larger-dimension problems, and such a strategy is much more appealing in a parallel environment, as was shown in [35, 36, 37, 122].

In [26] the algorithm named `DIRECT-l` was proposed. In most `DIRECT`-type algorithms, the hyper-rectangle size is measured by a half-length of a diagonal (2.5). In the `DIRECT-l`, the measure of a hyper-rectangle is evaluated by the length of its longest side. Such a measure corresponds to the infinity norm and allows the `DIRECT-l` algorithm to group more hyper-rectangles with the same measure. Thus, there are fewer distinct measures, and therefore, less potentially optimal hyper-rectangles are selected. Moreover, in the `DIRECT-l` at most one hyper-rectangle is selected from each group, even if there is more than one potentially optimal hyper-rectangle in the same group. It allows a reduction in the number of divisions within a group.

In the `PLOR` algorithm, all Lipschitz constants (here with the set of potentially optimal hyper-rectangles) are reduced to just two: the maximal and the minimal ones. In this way, the `PLOR` approach is independent of any user-defined parameters and balances equally local and global searches during the optimization process.

2.2.2 `DIRECT` algorithm for linear-constrained global optimization

Simplices may cover a feasible region defined by linear constraints. Therefore simplicial partitioning may tackle linear constraints in a very subtle

way [78]. All vertices of the feasible region are unique intersection points of all linear constraints. In this way, L-DISIMPL performs search only in the feasible region, and the algorithms have a tremendous advantage over all other DIRECT-type extensions for linear constrained global optimization problems. Two different sampling strategies are proposed: i) the Lc-DISIMPL evaluates the objective function at the geometric center of the simplex; ii) the Lv-DISIMPL evaluates the objective function on all unique vertices of the simplex. Nevertheless, the authors in [78] showed that the calculation of feasible region requires to solve $2n + m$ 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.

2.2.3 DIRECT algorithm for general-constrained global optimization

The first scheme for problems with general constraints, which can be used in DIRECT-type algorithms, was developed by one of the original DIRECT authors [45], and several years later, the investigation of three different constraint handling strategies within the DIRECT framework was carried out in [19]. However, the comparison revealed many disadvantages of handling infeasible hyper-rectangles, and only in recent years, various promising extensions of the DIRECT algorithm were introduced [3, 11, 52, 83, 82] for general global optimization problems.

In this subsection generally constrained global optimization problem is considered of the form:

$$\begin{aligned} \min_{\mathbf{x}\in D^{\text{feas}}} \; & f(\mathbf{x}) \\ \text{s.t. } & \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \end{aligned} \tag{2.6}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^u$, $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^v$ are (possibly non-linear) continuous functions. The feasible region consisting of points that satisfy all the constraints is denoted by $D^{\text{feas}} = D \cap \Omega$, where $\Omega = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \text{ and } \mathbf{h}(\mathbf{x}) = \mathbf{0}\}$. As for the box-constrained problems, it is also assumed that the objective and all constraint functions are Lipschitz-continuous (with

unknown Lipschitz constants) but can be non-linear, non-differentiable, and non-convex, and multi-modal.

Exact L1 penalty approach

One of the first DIRECT-type approaches for problems with general constraints was presented in [19]. The author extended the original DIRECT algorithm to handle constraints by using an auxiliary function that combines information of the objective and constraint functions in a special manner. An exact L1 penalty approach [21] is a transformation of the original constrained problem Eq. (2.6) to the form:

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) + \sum_{i=1}^{u} \max\{r_i g_i(\mathbf{x}), 0\} + \sum_{i=1}^{v} r_{i+u}|h_i(\mathbf{x})|, \tag{2.7}$$

where r_i are penalty parameters. The experiments in [19] showed promising results, but unfortunately, the strategy was compared only with DIRECT-type versions, which does not use all available constraint function information. It was found that the biggest drawback is the requirement for the users to set penalty parameters for each constraint function. In practice, choosing penalty parameters is a very important task and can have a huge impact on the algorithm [19, 52, 77, 78].

Two new approaches based on penalty function were recently proposed – EPGO [83] and DF-EPGO [82]. The main algorithms feature an automatic update rule for the penalty parameters, and under the weak assumptions, the penalty parameters are updated only a finite number of times.

Filter methodology

Another recently proposed DIRECT-type approach, Filter DIRECT [11], aims to simultaneously minimize the constraint violations and the objective function value. While other strategies work only with one general set of all hyper-rectangles, the Filter DIRECT algorithm adapts the filter methodology from [22] and splits the main set into three separate sets. The filtering strategy prioritizes selecting potentially optimal candidates: first, hyper-rectangles with feasible center points are selected, followed by those with infeasible but non-dominated center points, and finally by those that

have infeasible and dominated center points. The reported results show that the filter-based DIRECT method can compete with the existing DIRECT-type methods very well.

A metamodel-based constraint handling strategy

A metamodel-based [24, 103, 104] constrained DIRECT-type global optimization algorithm (eDIRECTc) was proposed in [52]. One of the algorithm's main differences and features is the employed Voronoi diagrams for partitioning the design space in Voronoi cells. Voronoi cells have irregular boundaries, and the eDIRECTc algorithm generates a set of random points to describe the cells. Thus, the algorithm is no longer deterministic and will always produce different performance results on separate algorithm runs. To speed up the convergence, the algorithm employs a pure greedy search on the objective metamodel $\hat{f}$. Also, the eDIRECTc algorithm separately handles feasible and infeasible cells.

2.2.4 DIRECT algorithm for problems with hidden constraints

In this section, the existing DIRECT-type methods for Eq. (2.2) optimization problems with hidden constraints are reviewed and summarized. Unfortunately, most of the time, descriptive information about objective functions does not exist, and no assumptions of smoothness, continuity, and where the function is defined can be made. The main challenge is convergence errors in physically infeasible regions, and then algorithms give error messages in evaluations. It can be caused when function contains noise or even may be discontinuous, and the objective function can be undefined everywhere within the given bounds, and the feasible region is unknown. Formally, the global optimization problem is defined in the following form:

$$\min_{\mathbf{x} \in D^{\text{feas}}} \quad f(\mathbf{x}), \tag{2.8}$$

where $D^{\text{feas}} = D \cap D^{\text{hidden}}$, and D^{hidden} is not given analytically, i.e., the problem has unknown hidden constraints. While solving the problem (2.2),

algorithms can use information about the feasibility only after evaluating the function at a certain point.

Barrier approach

One of the first proposed modifications for such problems was the barrier method [25]. The approach is very straightforward and places a tremendously high value on infeasible hyper-rectangles. The main issue is that edges of feasibility can be explored very slow, using such a strategy. The barrier method's significant penalties ensure that no infeasible hyper-rectangle can be potentially optimal as long as there is the same measure hyper-rectangle with the feasible midpoint. The `DIRECT-Barrier` algorithm preference will be the exploration of the regions where feasible points are founded previously. Another critical issue discovered by the author in [19] is that hyper-rectangle even with the sizeable feasible region, but the `DIRECT-Barrier` algorithm will not explore an infeasible center point in a reasonable number of function evaluations. The barrier approach is not the best fit to be used with the `DIRECT` framework [19, 25].

Neighbourhood Assignment Strategy

The second `DIRECT`-type approach can tackle problems of hidden constraints based on the Neighbourhood Assignment Strategy (NAS) [25]. This method aims to assign the value at infeasible point $\mathbf{x} \notin D^{\text{feas}}$ relative to the objective value attained in the feasible point from the neighborhood of $\mathbf{x}$. Every iteration the `DIRECT-NAS` algorithm iterates over all infeasible midpoints by creating surrounding hyper-rectangles around them by keeping the same center points. Hyper-rectangle increased by doubling the length of each dimension. If inside an enlarged region more than one feasible center point exists, the neighbourhood assignment strategy assigns the smallest function value to infeasible midpoint with small epsilon $f(\mathbf{x}) + \varepsilon_{\text{nas}} f(\mathbf{x})$, in [25] proposed to use the value $\epsilon = 10^{-6}$. If inside the enlarged region no feasible points exist the neighbourhood assignment strategy sets the largest objective function value founded so far $f_{max} + 1$. Such a strategy does not allow the `DIRECT-NAS` algorithm to move beyond the feasible region by punishing the

infeasible midpoints with larger values than f_{min}. However, the DIRECT-NAS principal concern is slow convergence caused by many additional calculations.

Incorporating sub-dividing step in the DIRECT algorithm

The most recent suggested idea to handle hidden constraints with the DIRECT algorithm framework is to use a sub-dividing step for infeasible hyper-rectangles [65] as the constraints handling method. Also, the proposed DIRECT-sub includes techniques taken from the barrier approach, and if the center points are identified as infeasible, then the function assigns an enormous penalty value to it. The sub-dividing step is performed only in specific iterations where all infeasible hyper-rectangles are identified as potential optimal and divided together with potential optimal hyper-rectangles obtained after the selection step. The sub-dividing step can decompose the boundaries of the hidden constraints and reveal the feasible region. If sub-dividing steps are performed at the proper iteration, performance can be increased effectively, but it has many limitations. The authors did not determine in which iterations this step should be used and how much it should be repeated during simulation, and experiments were performed only on the second-dimension test problem. However, hyper-rectangles with the vast infeasible region can result in many infeasible midpoints, and divisions per each sub-division step will grow drastically. Moreover, the main drawback here is dimensionality, where the number of partitions increases together with dimension, therefore the proposed sub-dividing step should be used only on low dimension optimization problems.

2.3 Implementations of DIRECT-type algorithms

The performance of DIRECT-type algorithms highly depends on the implementation. Most of the publicly available implementations use static data memory management, which is more straightforward but usually less effective due to a problematical prediction of memory requirements. The

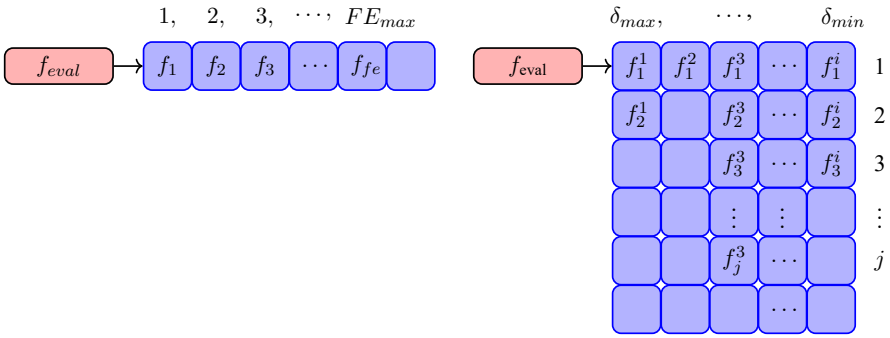

Figure 2.4: Information storage using static data structure (*left*) and dynamic data structure (*right*)

typical data structures store a collection of objective (and constraint) function values, index numbers, center point coordinates, side lengths of hyper-rectangles, and other similar data. One of the most broadly used publicly available implementation [18] of the `DIRECT Version 4.0` algorithm uses static data structures whose size is predetermined depending on stopping conditions. All information received after partitioning is stored in the contiguous blocks of memory, as shown on the left side of Fig. 2.4.

Using such data structures the algorithm can quickly and easily access the elements for further **Selection** (see Algorithm 1, Line 3), **Sampling** (see Algorithm 1, Line 4), and **Subdivision** (see Algorithm 1, Line 5) steps. Moreover, the content of the data structure can be modified without changing the memory space allocated to it. Otherwise, the algorithm should spend extra time re-allocating a more massive contiguous memory block and then copying/moving existing information into this new block. The `DIRECT Version 4.0` implementation allocates a large array to store information received after **Sampling** and **Subdivision** steps. The allocated vector size corresponds to one of the stopping conditions: the maximal number of function evaluations. However, in practice, choosing the maximum size of function evaluations is not know in advance. The main disadvantage of using static data structures is the requirement to initially reserve a large amount of memory, which slows down the algorithm and comes with an overhead proportional to the allocated vector's size.

Another disadvantage of typical static data structures used in `DIRECT`-type

algorithms is that they perform unnecessary recalculations in each iteration. One of the most critical tasks in DIRECT-type algorithms is the **Selection** step, where the algorithm decides which hyper-rectangles are potentially optimal (the most promising) for further investigation. Fig. 2.5 illustrates the selection of potentially optional hyper-rectangles using one of the DIRECT-GL schemes. Typically, all hyper-rectangles can be displayed graphically, as shown on the left side in Fig. 2.3, where data is arranged in columns and sorted according to the function value(s) obtained at each hyper-rectangle. Vertical sequences of columns are classified in the order of hyper-rectangle diameters. Usually, this procedure requires to sort all existing hyper-rectangles by the same size of diameters. Such sorting becomes inefficient when the amount of data gets larger during optimization, especially for higher dimension problems. Therefore, implementations using static data structures are often are slow, and this limits their applicability in practice.

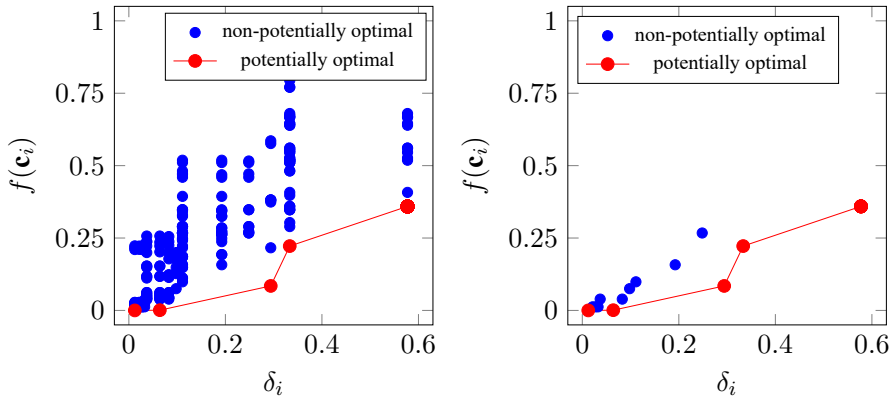

Figure 2.5: Selection of potential optimal hyper-rectangles using static data structures (*left*) and dynamic data structures (*right*) implementations in the DIRECT-GL

In [38], the authors proposed the idea to use dynamic data structures. Information received after the partitioning is sorted out by hyper-rectangle diameters and stored in columns, as shown on the right side of Fig. 2.4. All rectangles of the same diameter are stored in the column at any order. In [38], the authors have mentioned the idea to sort columns by function values in

descending order or insert all new data in sorted sequences separately, but these ideas have not been investigated further.

Using implementation based on dynamic data structures, the selection of potentially optimal hyper-rectangles in DIRECT (the **Selection** step) is simplified. Selection can be performed only in the set consisting of the best function values from each column, as shown on the right side of Fig. 2.5, which can save a lot of time compared to the previous selection strategy. Another considerable advantage comes from the fact that the **Selection** step is hardly parallelized. Therefore, the selection of potential optimal hyper-rectangles using dynamic data structures simplifies and significantly reduces the time needed for this step.

One of the obvious drawbacks of the dynamic data structures is allocating the columns, which is unpredictable. Two operations change the column size: fully processed potentially optimal hyper-rectangle needs to be removed from the previous column and new ones added to a new/existing column. Depending on the problem's dimension, the initial array is allocated a fairly large size in the usual way. If the array provides insufficient size, new blocks of columns will be reallocated as needed. In practice, only a few of these columns become large at any given time. In [38], the authors examined and determined that the dynamic data structure is more memory expensive, compared with static, and the way code is implemented has enormous impacts on running time.

2.4 Parallel DIRECT-type algorithms and design challenges

Parallel programming is often used to implement many deterministic Lipschitz global optimization methods [32, 75, 80, 92, 93, 99, 112, 113, 114]. Among them, the branch-and-bound framework is especially well suited for global optimization [13, 27, 40, 112], and different node selection strategies [79], branching rules, and bound calculations were proposed and investigated. However, the iterative nature of the partitioning-based DIRECT-type algorithms limit possibilities for effective parallelism [31, 34, 35, 36, 37, 81, 122]. Usually, the DIRECT-type algorithms

belong to the class of "Divide the best" methods [100]. Therefore, the partitioning procedure is not uniform, and more potential partitions (hyper-rectangles) are preferred. thus, the DIRECT-type methods, in some sense, can be viewed as a class of *branch without bounding* type methods.

In the development of the parallel DIRECT-type algorithms, the main challenge is a strong data dependency between different iterations and quite a small number of selected potentially optimal hyper-rectangles (even solving higher dimensionality problems) to process further, which does not allow efficient use of many computational cores. In Fig. 2.6, the growth of the number of different diameters and the number of selected potentially optimal hyper-rectangles, obtained by the DIRECT Version 4.0 algorithm is illustrated. It is clear that even solving higher dimensionality problems, the number of different diameters is limited, and therefore, the number of selected potential optimal hyper-rectangles is quite small.

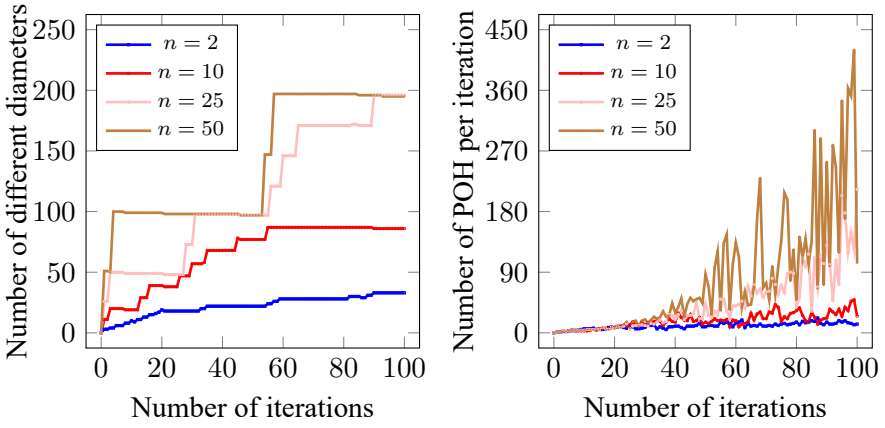

Figure 2.6: The growth of the number of different diameters (*left*) and the number of selected potential optimal hyper-rectangles (*right*), obtained by the DIRECT Version 4.0 algorithm on the *Rosenbrock* test problem with different dimensionality n.

The first design challenge comes in the very first **Initialization** step (see Algorithm 1, Line 1), where the algorithm starts from a single center point and, therefore, produces only one evaluation task for all the acquired workers. The situation improves when the algorithm progresses longer by

subdividing more hyper-rectangles. Load balancing is always a problem in the initial iterations, as there are not enough hyper-rectangles to process. Moreover, for low dimensional problems the load balancing issue is more critical, as the number of different partitions grows slower (see Fig. 2.6), compared to the problems of higher dimensionality.

Another important design challenge comes in the **Sampling** step (see Algorithm 1, Line 4), which cannot be started until the **Selection** step is finished (see Algorithm 1, Line 3) and vice versa. Only the **Sampling** and **Subdivision** steps (see Algorithm 1, Lines 4 and 5) can be parallelized efficiently. The **Selection** step is very difficult to parallelize efficiently, especially preserving the determinism of the algorithm. Usually, in the **Selection** step, a load imbalance occurs with the most workers being idle. Moreover, the cost of the **Selection** increases when the algorithm processes longer (see Fig. 6.3), thus reducing the total percentage of works which can be performed in parallel. Also, the number of selected potentially optimal hyper-rectangles in the **Selection** step cannot be predicted accurately in advance (see Fig. 2.6). When the number of selected potential optimal hyper-rectangles is small, an insufficient work will be sent to workers and processors will not be used efficiently, some of them possibly being idle.

2.4.1 Previous ideas to overcome the main challenges of parallel DIRECT-type algorithms

The authors in [34] proposed the decomposition of the initial n-dimensional hyper-rectangle D into smaller parts $\hat{D}_i, i = 1, \ldots, \gamma$. Two different hierarchical parallel schemes were proposed. The first parallel scheme operates on three different levels. At the initial level, the entire search space D is decomposed into multiple subdomains γ, and then optimization on each of them is performed independently. On each subdomain $\hat{D}_i$ master performs the **Selection** step and sends the identified potential optimal hyper-rectangles to the level two. Between levels two and three, a master-slave paradigm is used for distributing function evaluation tasks. The second introduced parallel scheme is an extension of the first one, which combines the first two levels to maintain data structures and independently performs the **Selection**

and **Subdivision** steps on their subdomain. While in the first proposed scheme, each master has a defined number of workers, in the second scheme, all masters globally share the workers' pool. The performance analysis in [34, 36, 37] revealed that the decomposition of domain D improves the load balancing, and the proposed DIRECT-type algorithm scales better on large parallel systems. However, splitting the entire search space into smaller subdomains destroys the original DIRECT-type algorithm's determinism, and using the different sizes of subdomains γ can result in an uneven algorithm performance.

Most of the previous parallel DIRECT-type versions [34, 35, 36, 37, 122] are focused on improving parallel efficiency by creating more computations in every iteration. Therefore, they use a different scheme to select potentially optimal hyper-rectangles. The authors in [1] introduced a particular scheme for the selection of potentially optimal hyper-rectangles, which is called "aggressive" DIRECT. The main idea of "aggressive" DIRECT is to select and subdivide a hyper-rectangle of every diameter in each iteration.

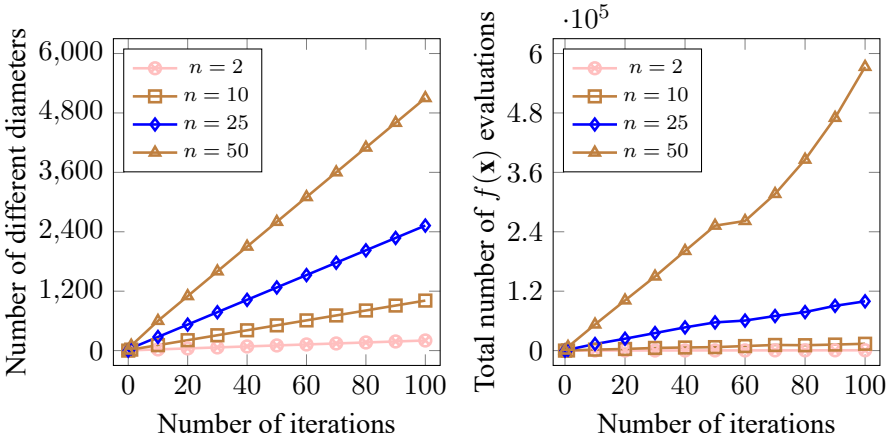

Figure 2.7: The growth of the number of different diameters (*left*) and the total number of objective function evaluations per iteration (*right*), obtained by the Aggressive DIRECT algorithm on the *Rosenbrock* test function with different dimensionality n.

The aggressive version relaxed for the criteria of selection of potentially optimal hyper-rectangles, thus assuring a much higher number of selected

potential optimal hyper-rectangles per iteration (which coincides with the number of different diameters, see Fig. 2.7), and therefore, requires a much higher number of function evaluations per iteration, compared to DIRECT. This approach does not appear to be favorable from the optimization point of view, since it is wasting function evaluations, by exploring "unnecessary" (non-potentially optimal) hyper-rectangles. For more difficult (higher dimensionality) optimization problems, iteration cost of the aggressive DIRECT version grows much faster compared to other DIRECT-type versions (see Figs. 2.6, 2.7 and 3.2). To overcome the high cost of later iterations for larger dimension problems, a massive supercomputer may be required. To reduce memory requirements and balance the cost of iteration, the authors proposed to limit the refinement of the search-space when the measure of hyper-rectangles reached some prescribed size. According to the authors, the latter technique reduces the memory usage from 10% to 70%, and thus, the algorithm can run longer without memory allocation failure. Such an approach improves parallel efficiency by creating a massive size of work for processors in every iteration, but it diminishes the optimization efficiency of the DIRECT algorithm [20, 25].

The parallel hybrid methods, combining generating set search (GSS) [51] and DIRECT, achieve better load balancing [31]. Such an approach is faster in parallel than original DIRECT, even when it requires more function evaluations.

2.5 Conclusions

This chapter described the deterministic derivative-free DIRECT-type global optimization algorithms and identified the main problems that will be considered in further chapters. First, all the originally developed DIRECT algorithm steps are reviewed in detail, followed by algorithm extensions for problems with box, general and hidden constraints. Next, the influence of data structures on the performance of the DIRECT-type algorithms is discussed. And finally, the known design challenges of the parallel DIRECT-type algorithm are investigated, and existing attempts of DIRECT algorithms parallelization are reviewed.

Chapter 3

New scheme for the selection of potential optimal hyper-rectangles

In this chapter, the newly developed potential optimal hyper-rectangles selection strategy is investigated, which addresses both weaknesses of the original DIRECT algorithm described in the first chapter. First, the proposed schemes for identifying potential optimal hyper-rectangles are described, following the algorithms convergence analysis. Next, step by step, a new algorithm for box-constrained global optimization is introduced. The results of numerical experiments are given, and a detailed comparison of these technique's efficiency is presented.

3.1 Introduction

As was mentioned before, the DIRECT algorithm has some well-known algorithmic weaknesses, e.g., delaying the discovery of the global minimum and the slow fine-tuning of the solution to high accuracy. These weaknesses are especially evident in optimization problems with many local minima and in cases where the solution with high precision is sought [45, 72, 73, 123]. Various proposals have been introduced in the literature to overcome these inefficiencies (see [57, 59, 72, 97] and references therein). In the first section

of this chapter, a new way to identify the extended set of potentially optimal hyper-rectangles is presented. In the new two-step based strategy, the set of the best hyper-rectangles is expanded 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 extension makes the algorithm work more globally (compared to the selection procedure used in DIRECT), while the second part assures faster and broader examination around the current minimum point. In this way, both original DIRECT weaknesses are addressed, staying in the same algorithmic framework.

3.2 Extended set of potentially optimal hyper-rectangles

Let $\mathbb{L}^k$ be the set of all different indices at the current partition $\mathcal{H}^k$, corresponding to the hyper-rectangles groups having the same measure (δ^k). The minimum value $l^k_{\min} \in \mathbb{L}^k$ corresponds to the hyper-rectangles groups having the smallest measure $\delta^k_{\min}$. The maximum value $l^k_{\max}$ of $\mathbb{L}^k$ corresponds to the hyper-rectangles groups having the largest measures $\delta^k_{\max}$, i. e., $l^k_{\max} = \max\{\mathbb{L}^k\} < \infty$. Finally, let $l^k_i \in \mathbb{L}^k$ be the group's index were the hyper-rectangle $\bar{D}^k_i$ belongs. In Definitions 2 and 3, new strategies for identifying an extended set of potentially optimal hyper-rectangles from the current partition $\mathcal{H}^k$ are formalized and in such a way addressing both weaknesses of DIRECT-type algorithms staying in the DIRECT algorithmic framework.

Definition 2 (Enhancing the global search)

- ***Step 1*** *Find an index $j \in \mathbb{I}^k$ and a corresponding hyper-rectangle $\bar{D}^k_j$, such that*

$$\bar{D}^k_j = \arg\max_j \{l^k_j : j = \arg\min_{i \in \mathbb{I}^k:\, l^k_{\min} \le l^k_i \le l^k_{\max}} \{f(\mathbf{c}_i)\}\}. \tag{3.1}$$

- ***Step 2*** *Set $l^k_{\min} = l^k_j + 1$. If $l^k_j \le l^k_{\max}$* ***repeat*** *from Step 1; otherwise* ***terminate****.*

At Step 1, the hyper-rectangle containing the minimum point $(\mathbf{x}_{\min})$ is selected. If there are several hyper-rectangles with the same lowest objective

value $f(\mathbf{c}_i)$, the preference is given to hyper-rectangles with the largest l_j^k value, i.e., a more significant size measure. After this, in Step 2, the minimum value $l_{\min}^k = l_j^k + 1$ is increased; thus, all hyper-rectangles from the groups with indices lower than the updated $l_{\min}^k$ (measures of these hyper-rectangles belonging to these groups are smaller than the measure of the $l_{\min}^k$ group) are not considered in the recurrent Step 1. A geometrical interpretation of the globally enhanced (`DIRECT-G`) version is shown in the left-hand side graphs in Figure 3.1. By this strategy, the number of medium-measured, potentially optimal hyper-rectangles are extended and force the `DIRECT-G` algorithm to work more globally. It can be stressed, that instead of the `Aggressive DIRECT` version, the Definition 2 (`DIRECT-G`) will not consider hyper-rectangles from the groups where the minimum function value is more significant than the minimum value from the larger groups.

Definition 3 (Enhancing the local search)

- ***Step 1*** *At each iteration k, evaluate the Euclidean distance from the current minimum point $(\mathbf{x}_{\min})$ to other sampled points:*

$$d(\mathbf{x}_{\min}, \mathbf{c}_i) = \sqrt{\sum_{j=1}^{n} (x_{\min}^j - c_i^j)^2} \tag{3.2}$$

- ***Step 2*** *Apply the procedure described in Definition 2 in* (3.1) *using distances $d(\mathbf{x}_{\min}, \mathbf{c}_i)$ instead of objective function values.*

A geometrical interpretation of the selection of potentially optimal hyper-rectangles using the locally enhanced strategy is shown on the right-hand side of Figure 3.1. This strategy extends the number of potentially optimal hyper-rectangles locating close to the current minimum point $(\mathbf{x}_{\min})$. Moreover, by this strategy, the closest hyper-rectangles from various measures are selected. Note that `DIRECT-L` identifies potential optimal hyper-rectangles not using function values at all, and the algorithm can stagnate if the global solution is too far from the current best function value. Therefore the effectiveness of `DIRECT-L` is considered only to show the method's efficiency on the particular group of optimization problems.

Comparing Figs. 2.6 and 3.2, the `DIRECT-GL` algorithm often selects a larger number of potential optimal hyper-rectangles than `DIRECT Version`

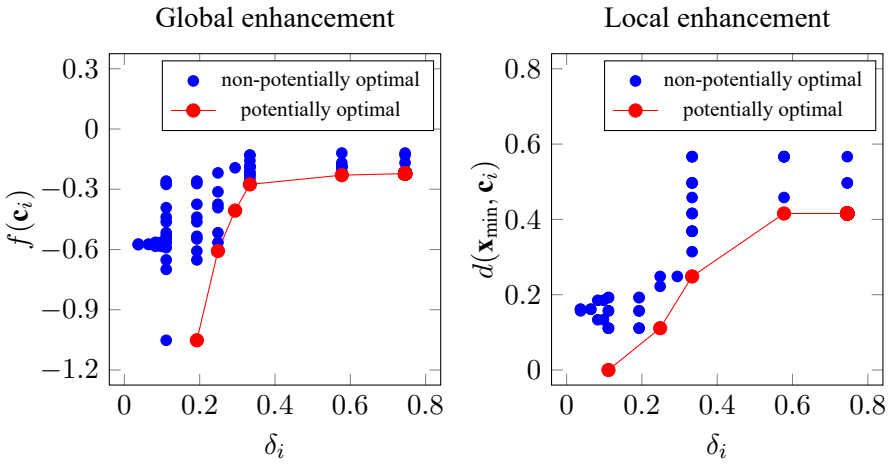

Figure 3.1: Selection of the potentially optimal hyper-rectangles using the `DIRECT-G` (on the left-hand side) and `DIRECT-L` algorithms (on the right-hand side) solving the *Shekel 5* test problem in the fifth iteration

`4.0`, solving various dimension *Rosenbrock* test functions. The Eq. (3.1) in Definition 2 limits selection to only one hyper-rectangle $\bar{D}_j^k$ from the group l_j^k with any $\forall j$, even if there are many equal objective function f values. The combined set $\mathbb{S}_3^k = \mathbb{S}_1^k \bigcup \mathbb{S}_2^k$ of potential optimal hyper-rectangles using both selection procedures Definition 2 and Definition 3, does not exceed the number of different diameters generated at any iteration (see, Fig. 3.2). `DIRECT Version 4.0` behaves entirely differently when there are many equal values of the objective function f in any group l_j^k. `DIRECT Version 4.0` selects all hyper-rectangles from the group l_j^k if they have identical values and satisfies Definition 1; thus, the algorithm can select a larger number of hyper-rectangles when different diameters exist (see, Fig. 2.6).

3.3 Algorithmic steps

The key feature of the `DIRECT-GL` algorithm is that it performs the identification of potentially-optimal hyper-rectangles twice in every iteration. First, using Definition 2, the globally enhanced set of potentially optimal candidates is determined and fully processed (sampled and partitioned). Second, by using Definition 3, the locally enhanced set is identified and fully

Algorithm 2: Pseudo code of the `DIRECT-GL` algorithm

input : $\mathcal{P}$, ε_{pe}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$;

output: $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, k, fe;

Initialize $k = 1$, $fe = 1$, $\mathbb{I}^k = \{1\}$, $f_{\min} = f(\mathbf{c}_1)$, $\mathbf{x}^k_{\min} = \mathbf{c}_1$;

while $pe > \varepsilon_{\text{pe}}$ **and** $fe < \mathsf{FE}_{\max}$ **and** $k < \mathsf{K}_{\max}$ **do** `// pe defined in Eq. (3.7)`

  Identify the "global" (index) set $\mathbb{S}^k_1 \subseteq \mathbb{I}^k$ of potentially optimal hyper-rectangles using Definition 2 ;

  Identify the "local" (index) set $\mathbb{S}^k_2 \subseteq \mathbb{I}^k$ of potentially optimal hyper-rectangles using Definition 3 ;

  Find unique union of potential optimal hyper-rectangles sets $\mathbb{S}^k_3 = \mathbb{S}^k_1 \bigcup \mathbb{S}^k_2$;

  foreach $i \in \mathbb{S}^k_3$ **do**

    Subdivide (trisect) hyper-rectangle $\bar{D}^k_i$ and update $\mathbb{I}^k$;

    Evaluate f at the centers of the new hyper-rectangles;

    Update $f_{\min}$, $\mathbf{x}^k_{\min}$, pe and fe;

  end

  if $d(\mathbf{x}^k_{\min}, \mathbf{x}^{k-1}_{\min}) \geq 10^{-6}$ **then**

    Calculate distances $\mathbf{d}(\mathbf{x}^k_{\min}, \mathbf{c}_i),\ i \in \mathbb{I}^k$ for all $i = 1, ..., fe$; `// using Eq. (3.2)`

  else

    Calculate distances $\mathbf{d}(\mathbf{x}^k_{\min}, \mathbf{c}_i)$ for all $i = fe_{\text{old}}, ..., fe_{\text{new}}$; `// using Eq. (3.2)`

  end

  Increase $k = k + 1$;

end

return $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, k, fe;

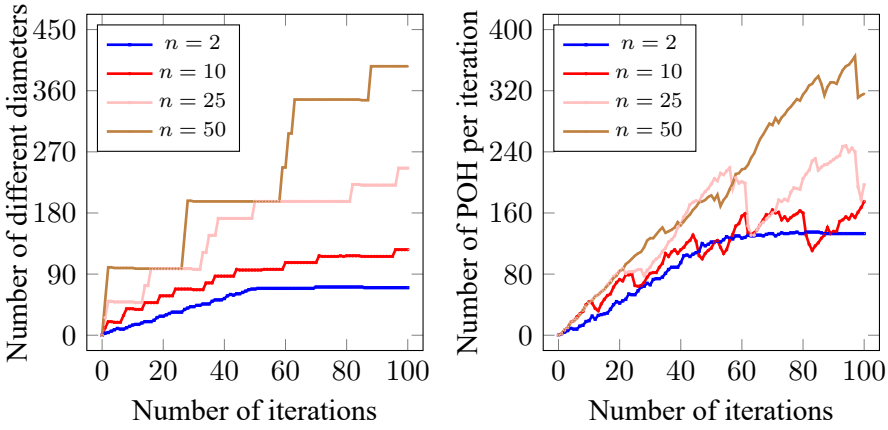

Figure 3.2: The growth of the number of different diameters (*left*) and the number of selected potential optimal hyper-rectangles (*right*), obtained by the DIRECT-GL on the *Rosenbrock* test problem with different dimensionality n.

processed (sampled and partitioned) again. Thus, the developed new approach is based on the "Divide the best" strategy [101], and it has the everywhere-dense type of convergence (like other DIRECT-type algorithms [20, 46, 71, 72, 97]).

The complete description of the DIRECT-GL algorithm is shown in Algorithm 2. The inputs for the algorithm are the problem ($\mathcal{P}$) and one (or a few) stopping criteria: required tolerance (ε_{pe}), the maximal number of function evaluations ($\text{FE}_{\max}$) and the maximal number of DIRECT-GL iterations ($\text{K}_{\max}$). After termination, the DIRECT-GL algorithm returns the found objective value $f_{\min}$ and the solution point $\mathbf{x}_{\min}$ together with algorithmic performance measures: final tolerance – percent error (pe), the number of function evaluations (fe), and the number of iterations (k).

3.4 Convergence properties of the DIRECT-GL algorithm

The convergence properties of the DIRECT-type algorithms are broadly reviewed and investigated [20, 46, 71, 72, 97]). The introduced DIRECT-GL as well as structural versions DIRECT-G and DIRECT-L algorithms

convergence properties are similar to the original DIRECT and based on "Divide the best" strategy [101] and it has the everywhere-dense type of convergence. The division strategy used by the developed algorithms is the same as in the original DIRECT and ensures that sampling occurs in every dimension of hyper-rectangles. Furthermore, in DIRECT, each point examined by the developed methods is an accumulation point, and the algorithms eventually sample arbitrarily close to global minima. This follows from Definition 2 and Definition 3, where the set of potentially optimal hyper-rectangles always includes at least one hyper-rectangle from the group of hyper-rectangles with the biggest diameter from the group $(l^k_{\max})$ with the largest measure $\delta^k_{\max}$. Since each group contains only a finite number of hyper-rectangles, any hyper-rectangles will be partitioned after a sufficient number of iterations.

The developed algorithms are guaranteed to converge to the global minimum if the objective function is continuous at least in the neighborhood of the global optimum $\mathbf{x}^*$. In Theorem 1, we state the convergence of developed algorithms to a global minimizer $\mathbf{x}^*$ within any positive tolerance $\epsilon > 0$.

Theorem 1 *For any global minima* $\mathbf{x}^* \in D$ *of the objective function* f *and any* $\epsilon > 0$ *there exists an iteration number* k *and a hyper-rectangle index* $i \in \mathbb{I}^k$ *such that:*

$$\max\{\|\mathbf{b}_i - \mathbf{x}^*\|_2, \|\mathbf{a}_i - \mathbf{x}^*\|_2\} \leq \epsilon. \tag{3.3}$$

Proof. From Definitions 2 and 3 it follows that there exists at least one potentially optimal hyper-rectangle D^k_i in every k iteration $D^k_i \in \mathbb{S}^k$, and this hyper-rectangle belong to group of hyper-rectangles $l^k_{\max}$ having the largest measures $\delta^k_{\max}$:

$$\delta^k_{\max} = \max\{\delta^k_i, i \in \mathbb{I}^k\} = \max\{\delta^k_i, i \in \mathbb{S}^k\}. \tag{3.4}$$

Since each group in $\mathbb{L}^k$ contains only a finite number of hyper-rectangles, after a sufficient number of iterations, all hyper-rectangles of the group $l^k_{\max}$ with the maximal measure $\delta^k_{\max}$ will be partitioned. The division strategy used in DIRECT-GL ensures that sampling occurs in every dimension, and each potentially optimal hyper-rectangles are fully subdivided and reduced in

size. From this follows, that such a procedure will subdivide any new group $l^k_{\max-1}$ of the largest hyper-rectangles that will appear after subdividing $l^k_{\max}$. As the results after finite number of iterations in the partition $\mathcal{H}^k$ the size measure of any hyper-rectangle $D^k_j \in l^k_{\max}$ will be $\delta^k_j \leq \epsilon$:

$$\frac{1}{2}\|\mathbf{b}_j - \mathbf{a}_j\|_2 \leq \epsilon. \tag{3.5}$$

Then, from Eq. (3.5) follows that the measure δ^k_i of the hyper-rectangle containing the global minimum $\mathbf{x}^* \in D^k_i$ is also not exceeding ϵ:

$$\max\{\|\mathbf{b}_i - \mathbf{x}^*\|_2, \|\mathbf{a}_i - \mathbf{x}^*\|_2\} \leq \delta^k_j. \tag{3.6}$$

Thus, from Eq. (3.5) and Eq. (3.6) follows Eq. (3.3).

3.5 Numerical investigation

Tested algorithms

The introduced algorithms and other DIRECT-type methods used in further experiments were implemented in the MatLab programming language. It should be noted that for the DIRECT algorithm, potentially optimal hyper-rectangles can be identified in at least two different ways: using the modified Graham's scan algorithm [6] (BIRECT, DISIMPL-V and DISIMPL-C) or the rule described in Definition 1 (DIRECT-l, PLOR and DIRECT Version 4.0). Usually, this does not impose significant differences, but occasionally it can have some differences, e.g., when higher precision is required using Definition 1 can be more useful. The selection procedure of potentially optimal hyper-rectangles in DIRECT-GL differs significantly, however, this does not have a notable difference in overall performance, compared with the procedure used in DIRECT. This means that to identify the same quantity of potentially optimal hyper-rectangles DIRECT Version 4.0 and DIRECT-GL spent a similar amount of time. A total of nine algorithms were tested.

Benchmark problems

The algorithm's efficiency on the DIRECTlib test set [110] is compared, consisting of 31 global optimization test functions. Note that all tested

problems from the DIRECTlib test set are multimodal, therefore suitable to investigate how introduced modifications help overcome the first weakness. In Table 3.1 the main features of these test problems are reported: name of the problem, source, dimensionality (n), feasible region (D), the number of local minima (if known), and the known minimum (f^*). Some of the test problems have several variants, e.g., *Bohachevsky*, *Shekel*, and some of them like *Alpine*, *Csendes*, *Griewank* and etc. can be tested for different dimensionality. In total, the algorithms were examined on 59 multimodal box-constrained global optimization test problems.

Stopping criteria

Since all the global minima f^* are known for all tested problems in advance, investigated algorithms were stopped either when the point $\mathbf{x}$ was generated such that the percent error

$$pe = 100\% \times \begin{cases} \frac{f(\mathbf{x})-f^*}{|f^*|}, & f^* \neq 0, \\ f(\mathbf{x}), & f^* = 0, \end{cases} \tag{3.7}$$

is smaller than the tolerance value ε_{pe}, or when the number of function evaluations exceeds the prescribed limit of 10^6. In flow investigation, four different values for ε_{pe} were considered: 10^{-2} and 10^{-8}. By doing this, the algorithm's ability to avoid the second weakness is investigated.

Evaluation of the performance of the algorithms

First, the comparison is based on the number of function evaluations, and the best (smallest) number for each problem is shown in bold font in the tables. Second, to evaluate the different solver's performance when running on a broad set of test problems, performance profiles [15] were applied. In this section, the algorithm's performance applying the performance profiles tool with the convergence test Eq. (3.7) is analyzed. Benchmark results generated by running a specific algorithm υ (from a set of algorithms $\mathcal{V}$ under consideration) for each problem ρ from a benchmark set $\mathcal{P}$ and recording the performance measure of interest, which could be, for example, the number of function evaluations, the computation time, the number of iterations or the

Table 3.1: Key characteristics of the `DIRECTlib` [110] test problems for box-constrained global optimization

Label	Source	n	D	No. of minima	f^*
Ackley	[39, 116]	$2, 5, 10$	$[-15, 30]_i, i = 1...n$	multimodal	0.00000000
Alpine	[10]	$5, 10, 15$	$[0, 10]_i, i = 1...n$	multimodal	-2.80813118^n
Beale	[39, 116]	2	$[-4.5, 4.5]_i, i = 1...n$	multimodal	0.00000000
Bohachevsky 1	[39, 116]	2	$[-100, 110]_i, i = 1...n$	multimodal	0.00000000
Bohachevsky 2	[39, 116]	2	$[-100, 110]_i, i = 1...n$	multimodal	0.00000000
Bohachevsky 3	[39, 116]	2	$[-100, 110]_i, i = 1...n$	multimodal	0.00000000
Branin	[39, 116]	2	$[-5, 10]_1, [10, 15]_2$	3	0.39788735
Bukin	[116]	2	$[-15, 5]_1, [-3, 3]_2$	multimodal	0.00000000
Csendes	[10]	$5, 10, 15$	$[-10, 20]_i, i = 1...n$	multimodal	0.00000000
Colville	[39, 116]	4	$[-10, 10]_i, i = 1...n$	multimodal	0.00000000
Cross-in-Tray	[116]	2	$[-10, 10]_i, i = 1...n$	multimodal	-2.06261187
Drop-Wave	[116]	2	$[-5.12, -6.12]_i, i = 1...n$	multimodal	-1.00000000
Easom	[39, 116]	2	$[-100, 100]_i, i = 1...n$	multimodal	-1.00000000
Eggholder	[116]	2	$[-512, 512]_i, i = 1...n$	multimodal	-959.64066272
Goldstein & Price	[39, 116]	2	$[-2, 2]_i, i = 1...n$	4	3.00000000
Griewank	[39, 116]	$5, 10, 15$	$[-600, 700]_i, i = 1...n$	multimodal	0.00000000
Hartman	[39, 116]	3	$[0, 1]_i, i = 1...n$	4	-3.86278214
Holder Table	[116]	2	$[-10, 10]_i, i = 1...n$	4	-19.20850000
Hump	[39, 116]	2	$[-5, 5]_i, i = 1...n$	6	-1.03162845
Langermann	[116]	2	$[0, 10]_i, i = 1...n$	6	-4.15580929
Levy	[39, 116]	$5, 10, 15$	$[-5, 5]_i, i = 1...n$	multimodal	0.00000000
McCormick	[116]	2	$[-1.5, 4]_1, [-3, 4]_2$	multimodal	-1.91322295
Michalewicz	[39, 116]	$2, 5, 10$	$[0, \pi]_i, i = 1...n$	2!	-1.80130341
Michalewicz	[39, 116]	$2, 5, 10$	$[0, \pi]_i, i = 1...n$	5!	-4.68765817
Michalewicz	[39, 116]	$2, 5, 10$	$[0, \pi]_i, i = 1...n$	10!	-9.66015171
Perm($\beta = 0.5$)	[39, 116]	5	$[-n, n]_i, i = 1...n$	multimodal	0.00000000
Perm($\beta = 10$)	[39, 116]	8	$[-n, n]_i, i = 1...n$	multimodal	0.00000000
Power Sum	[39, 116]	4	$[0, 4]_i, i = 1...n$	multimodal	0.00000000
Qing	[10]	$5, 10, 15$	$[-500, 500]_i, i = 1...n$	multimodal	0.00000000
Rastrigin	[39, 116]	$2, 5, 10$	$[-6.12, 5.12]_i, i = 1...n$	multimodal	0.00000000
Schwefel	[39, 116]	$2, 5, 10$	$[-500, 500]_i, i = 1...n$	multimodal	0.00000000
Shekel($m = 5$)	[39, 116]	4	$[0, 10]_i, i = 1...n$	5	-10.15319967
Shekel($m = 7$)	[39, 116]	4	$[0, 10]_i, i = 1...n$	7	-10.40294056
Shekel($m = 10$)	[39, 116]	4	$[0, 10]_i, i = 1...n$	10	-10.53640981
Shubert	[39, 116]	2	$[-10, 10]_i, i = 1...n$	760	-186.73090883
Styblinski-Tang	[10]	$5, 10, 15$	$[-5, 5]_i, i = 1...n$	multimodal	$-39.16616570n$
Trid	[39, 116]	6	$[-36, 36]_i, i = 1...n$	multimodal	-50.00000000
Trid	[39, 116]	10	$[-100, 100]_i, i = 1...n$	multimodal	-210.00000000
Zakharov	[39, 116]	$2, 5, 10$	$[-5, 11]_i, i = 1...n$	multimodal	0.00000000

memory used. In the following experiments, the number of function evaluations and the computation time criteria were used.

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

$$\lambda_{\rho,\upsilon} = \frac{t_{\upsilon,\rho}}{\min\{t_{\upsilon,\rho} : \rho \in \mathcal{P}\}}, \tag{3.8}$$

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

$$\chi_\upsilon(\beta) = \frac{1}{card(\mathcal{P})}\text{size}\{\rho \in \mathcal{P} : \lambda_{\rho,\upsilon} \leq \beta\}, \quad \beta \geq 1, \tag{3.9}$$

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

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

3.5.1 Group 1: low-dimensional cases

First, the algorithm's efficiency was evaluated, solving 27 low-dimensional $(n \leq 4)$ multimodal box-constrained global optimization test problems. The results of the experiments are given in Table 3.2. Here, the comma symbol "," is used to separate thousands.

First, `DIRECT-GL` performance, on the average, much better (see **Overall** row in Table 3.2) compared to all the other algorithms. Especially this is evident when a lower percentage error (pe) (higher accuracy) is sought. When $\varepsilon_{\text{pe}} = 10^{-2}$ was used, the second best algorithm `DISIMPL-V` requires on average 3.7 times more function evaluations compared to `DIRECT-GL` and when $\varepsilon_{\text{pe}} = 10^{-8}$ was used, the second best algorithm `DISIMPL-C` requires

on average 5.2 times more function evaluations compared to the same DIRECT-GL.

For small dimensional test problems $\mathbf{n} \leq \mathbf{4}$, all DIRECT-type algorithms perform quite well when the $\varepsilon_{\text{pe}} = 10^{-2}$ is used as stopping condition. However, DIRECT-type algorithms significantly suffer, when a solution with higher accuracy ($\varepsilon_{\text{pe}} = 10^{-8}$) is needed. All developed algorithms have only one loss of unsolved problem, comparing with the results, when the lower percentage error was used as the stooping condition $\varepsilon_{\text{pe}} = 10^{-2}$. The most significant recorded losses belong to BIRECT (13) and DISIMPL-V (11) of unsolved problems. DIRECT-GL failed (see **Failed** row in Table 3.2) to solve only one of the test problems $1.9\%(1/54)$ in total, while the second-best observed result was achieved by DISIMPL-C, where the algorithm fails to solve $18.5\%(10/54)$ cases accordingly. Furthermore, by the same criteria, the DIRECT-G and DIRECT-L algorithms showed better results than all other tested algorithms, by failing to solve only $5.5\%(3/54)$ cases accordingly. The developed algorithms are especially useful when the solution with high accuracy $\varepsilon_{\text{pe}} = 10^{-8}$ is needed.

Another observation is that solving simpler test problems the introduced algorithms use more function evaluations compared to other DIRECT-type methods. That is mainly because the set of potentially optimal hyper-rectangles in DIRECT-G is larger per iteration. Consequently, a greater number of function evaluations are needed.

3.5.2 Group 2: high-dimensional cases

Next, the efficiency of the algorithms was evaluated, solving 32 high-dimensional $(n \geq 4)$ multimodal box-constrained global optimization test problems. The results of the experiments are given in Table 3.3.

Once again, the developed DIRECT-GL is the most efficient optimizer and has only one loss of unsolved problem comparing with the results, when the lower percentage error was used as the stooping condition $\varepsilon_{\text{pe}} = 10^{-2}$. Meanwhile other DIRECT-type algorithms significantly suffer when a solution with higher accuracy ($\varepsilon_{\text{pe}} = 10^{-8}$) is needed. The most significant recorded losses belong to BIRECT (16) and DIRECT (9) of unsolved problems.

DIRECT-GL failed (see **Failed** row in Table 3.2) to solve only one of the test problems 10.9% (7/64) in total, while the second-best result was achieved by DIRECT, where the algorithm fails to solve more than a half test problems – 51.6% (33/64). Furthermore, by the same criteria, the DIRECT-G and DIRECT-L algorithms showed better results than all other tested algorithms, by failing to solve only 34.7% (22/64) and 39%(25/64) cases accordingly. The developed algorithms are especially useful, where the solution with high accuracy $\varepsilon_{\text{pe}} = 10^{-8}$ is needed.

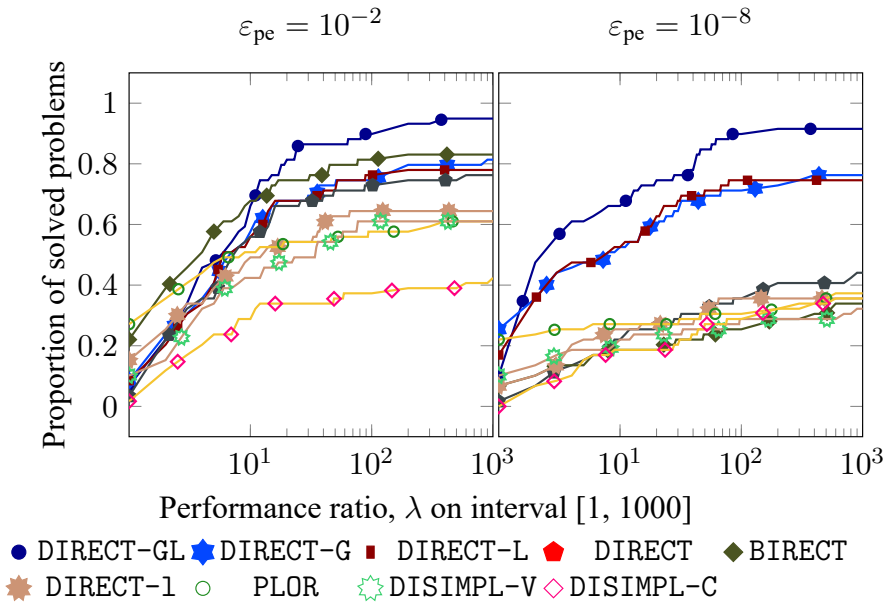

Figure 3.3: Performance profiles of DIRECT-type algorithms on the whole 59 box-constrained optimization test problems from DIRECTlib

The worst result was obtained using the simplicial partition-based DISIMPL-C algorithm, the main reason for this is the high-dimension used in the tested problems. The algorithm fails to solve approximately 93.8% (60/64) of test problems. Approximately 56.3% (18/32) of the test problems were larger than $n \geq 10$. In such situations the DISIMPL-C algorithm exceeds the maximal number of function evaluation budget in the initialization step without proceeding at least one iteration.

DIRECT-GL performance on the average is better (see **Overall** row in Table 3.2) compared to all other algorithms. When $\varepsilon_{\text{pe}} = 10^{-2}$ was used, the

second best algorithm BIRECT requires on average 1.6 times more function evaluations compared to DIRECT-GL and where $\varepsilon_{\text{pe}} = 10^{-8}$ was used, the second best algorithm PLOR requires on average 2.2 times more function evaluations compared to the same DIRECT-GL.

Table 3.2: CComparison between different DIRECT-type algorithm solving low-dimensional ($n \leq 4$) box-constrained global optimization problems from DIRECTlib

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Ackley	2	10^{-2}	1,069	851	753	255	202	**135**	143	311	578
		10^{-8}	4,525	3,489	3,379	909	21,976	473	**315**	266,004	1,126
Beale	2	10^{-2}	533	283	357	655	436	**245**	22,875	427	522
		10^{-8}	3,361	1,347	1,615	2,835	2,226	**1,143**	$> 10^6$	2,027	1,744
Bohachecsky 1	2	10^{-2}	689	435	435	327	476	205	**99**	358	456
		10^{-8}	1,955	1,129	1,133	845	4,274	511	**151**	1,323	850
Bohachecsky 2	2	10^{-2}	679	441	855	345	478	**233**	2,719	364	460
		10^{-8}	1,925	1,139	1,545	897	4,406	**551**	2,791	1,398	878
Bohachecsky 3	2	10^{-2}	719	623	**459**	693	480	573	119,217	892	462
		10^{-8}	2,609	1,795	1,595	2,099	3,672	**1,429**	706,621	2,784	2,680
Branin	2	10^{-2}	555	255	333	195	242	**159**	319	372	202
		10^{-8}	2,043	**841**	1,079	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	63,910
Bukin	2	10^{-2}	167,721	**76,507**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**458,433**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$

Continued on next page

Table 3.2 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Colville	4	10^{-2}	2,509	84,557	10,209	6,585	**794**	3,379	$> 10^6$	2,815	1,318
		10^{-8}	8,571	265,565	38,945	65,243	**2,334**	11,543	$> 10^6$	172,034	2,736
Cross-in-Tray	2	10^{-2}	299	307	187	569	120	251	199	413	**98**
		10^{-8}	833	**541**	573	46,401	$> 10^6$	21,835	24,553	$> 10^6$	58,722
Drop-Wave	2	10^{-2}	4,377	1,369	6,585	2,927	**190**	2,555	58,411	10,053	11,030
		10^{-8}	5,143	**1,873**	7,085	187,537	727,760	120,925	271,899	$> 10^6$	72,098
Easom	2	10^{-2}	467	325,955	**377**	32,859	16,420	6,903	7,015	8,429	40,462
		10^{-8}	1,491	326,529	**1,097**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Eggholder	2	10^{-2}	1,481	38,517	12,891	7,449	3,276	8,655	$> 10^6$	**1,062**	7,612
		10^{-8}	**5,425**	40,641	14,949	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	174,285	$> 10^6$
Gold. and Pr.	2	10^{-2}	325	209	269	191	274	115	85	**17**	180
		10^{-8}	1,341	789	839	828,825	$> 10^6$	791,489	288,917	**17**	119,656
Hartman 3	3	10^{-2}	685	361	313	199	352	**111**	**111**	261	334
		10^{-8}	3,097	**1,997**	2,011	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	538,892
Holder Table	2	10^{-2}	209	115	131	209	222	**59**	63	461	176
		10^{-8}	761	**379**	431	20,465	472	21,111	11,515	3,705	16,496
Hump	2	10^{-2}	367	211	215	293	334	**137**	231	375	228

Continued on next page

Table 3.2 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Hump	2	10^{-8}	1,629	1,089	**929**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	103,332
Langermannley	2	10^{-2}	281	167	197	**123**	686	173	393	1,536	912
		10^{-8}	1,431	**751**	827	653,093	$> 10^6$	774,223	440,885	$> 10^6$	3,026
McCormick	2	10^{-2}	179	129	131	113	130	75	**61**	170	128
		10^{-8}	1,015	**629**	761	479,409	$> 10^6$	$> 10^6$	216,465	$> 10^6$	3,010
Michalewics	2	10^{-2}	157	97	97	67	126	**45**	55	179	508
		10^{-8}	279	179	179	109	462	83	**79**	179	658
Power Sum	4	10^{-2}	93,201	$> 10^6$	12,561	$> 10^6$	10,902	$> 10^6$	$> 10^6$	**243**	313,036
		10^{-8}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	48,052	$> 10^6$	$> 10^6$	**243**	$> 10^6$
Rastrigin	2	10^{-2}	733	1,441	15,841	987	**180**	1,727	1,503	965	1,390
		10^{-8}	1,815	2,153	16,469	1,833	**758**	2,021	1,585	2,364	2,138
Schwefel	2	10^{-2}	591	349	807	255	236	341	**169**	472	542
		10^{-8}	1,605	1,113	1,555	1,195	1,332	679	**257**	3,188	1,226
Shekel 5	4	10^{-2}	1,311	761	725	**155**	1,200	7,185	7,133	2,485	88,064
		10^{-8}	5,715	3,721	**3,635**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Shekel 7	4	10^{-2}	1,311	753	697	145	1,180	141	**133**	719	$> 10^6$
		10^{-8}	7,871	5,341	**5,265**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$

Continued on next page

Table 3.2 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Shekel 10	4	10^{-2}	1,291	729	703	145	1,140	139	**133**	754	$> 10^6$
		10^{-8}	7,835	5,059	**5,031**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Shubert	2	10^{-2}	585	3,547	**375**	2,967	1,780	2,043	1,509	4,509	518
		10^{-8}	1,731	4,321	**1,083**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	723,387	32,926
Zakharov	2	10^{-2}	419	7,493	249	237	502	209	223	**42**	430
		10^{-8}	1,563	16,085	779	827	5,354	619	178,689	**42**	1,160
Average		10^{-2}	**10,472**	57,276	39,509	76,257	38,606	75,400	156,400	38,470	128,505
		10^{-8}	**56,815**	99,574	78,251	492,316	553,215	509,209	560,916	494,555	297,306
Median		10^{-2}	679	623	435	293	476	**233**	319	461	522
		10^{-8}	1,955	1,873	**1,555**	479,409	$> 10^6$	774,223	706,621	266,004	32,926
# of failed		10^{-2}	**0/27**	1/27	1/27	2/27	1/27	2/27	4/27	1/27	3/27
		10^{-8}	**1/27**	2/27	2/27	11/27	14/27	12/27	13/27	12/27	7/27

Concluded

Table 3.3: Comparison between different DIRECT-type algorithm solving higher-dimensional ($n \geq 5$) box-constrained global optimization problems from DIRECTlib

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Ackley	5	10^{-2}	15,655	11,103	97,401	8,845	1,268	$> 10^6$	**671**	22,881	$> 10^6$
		10^{-8}	67,647	45,013	131,161	17,757	$> 10^6$	$> 10^6$	**1,663**	$> 10^6$	$> 10^6$
	10	10^{-2}	130,949	90,487	$> 10^6$	80,927	47,792	$> 10^6$	**8,979**	$> 10^6$	$> 10^6$
		10^{-8}	502,577	339,943	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**11,163**	$> 10^6$	$> 10^6$
Alpine	5	10^{-2}	2,483	6,825	861	3,565	230	8,589	**843**	17,482	$> 10^6$
		10^{-8}	8,645	11,009	**5,051**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	62,565	$> 10^6$	233,863	$> 10^6$	**5,938**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**98,351**	$> 10^6$	258,277	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	15	10^{-2}	824,457	$> 10^6$	$> 10^6$	$> 10^6$	**137,578**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**926,677**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Csendes	5	10^{-2}	1,509	1,517	1,477	1,743	2,370	727	**257**	14,671	12,452
		10^{-8}	2,511	3,107	3,009	3,953	7,082	1,295	**337**	32,481	16,708
	10	10^{-2}	13,443	11,947	11,311	58,143	90,536	42,285	**1,187**	$> 10^6$	$> 10^6$
		10^{-8}	21,329	24,083	23,069	167,937	849,492	75,903	**1,647**	$> 10^6$	$> 10^6$
	15	10^{-2}	49,413	40,753	39,083	374,085	$> 10^6$	$> 10^6$	**2,867**	$> 10^6$	$> 10^6$
		10^{-8}	76,383	81,413	78,293	$> 10^6$	$> 10^6$	$> 10^6$	**3,953**	$> 10^6$	$> 10^6$

Continued on next page

Table 3.3 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Griewank	5	10^{-2}	$234,615$	**$205,343$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$243,063$	**$209,731$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	$> 10^6$	**$269,279$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$> 10^6$	**$298,339$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	15	10^{-2}	$232,363$	$129,745$	$> 10^6$	$> 10^6$	**$3,688$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$388,247$	**$219,181$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Levy	5	10^{-2}	$1,845$	$1,095$	$1,035$	517	872	327	**195**	$6,670$	$178,284$
		10^{-8}	$10,239$	$5,451$	$5,373$	$3,383$	$18,504$	$1,207$	**379**	$> 10^6$	$199,068$
	10	10^{-2}	$14,097$	$7,969$	$7,647$	$5,555$	$11,572$	$31,951$	**817**	$> 10^6$	$> 10^6$
		10^{-8}	$68,127$	$37,393$	$36,489$	$50,595$	$> 10^6$	$86,601$	**$1,619$**	$> 10^6$	$> 10^6$
	15	10^{-2}	$46,197$	$26,503$	$24,623$	$48,519$	$248,526$	$> 10^6$	**$1,867$**	$> 10^6$	$> 10^6$
		10^{-8}	$211,055$	$116,979$	$110,751$	$526,655$	$> 10^6$	$> 10^6$	**$3,731$**	$> 10^6$	$> 10^6$
Michalewics	5	10^{-2}	**$5,225$**	$5,459$	$> 10^6$	$14,077$	$73,866$	$26,405$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$9,755$	**$7,787$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	**$44,127$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**$57,475$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Qing	5	10^{-2}	$10,501$	$5,655$	$6,447$	$9,529$	$> 10^6$	**$3,747$**	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$28,337$	$13,817$	$16,263$	$17,479$	$> 10^6$	**$4,865$**	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	$104,299$	$> 10^6$	**$56,689$**	$837,387$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$

Continued on next page

Table 3.3 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Qing	10	10^{-8}	$241,195$	$> 10^6$	**$132,333$**	$938,403$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	15	10^{-2}	**$372,683$**	$> 10^6$	$374,419$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$838,803$	$> 10^6$	**$613,559$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Perm($\beta = 0.5$)	5	10^{-2}	**$13,505$**	$42,735$	$> 10^6$	$25,115$	$56,954$	$67,479$	$> 10^6$	$44,147$	$> 10^6$
		10^{-8}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**$240,715$**	$> 10^6$
Perm($\beta = 10$)	8	10^{-2}	$249,999$	$123,209$	$123,185$	$13,285$	$> 10^6$	$5,919$	$2,431$	**256**	$> 10^6$
		10^{-8}	$417,783$	$206,489$	$206,465$	$26,853$	$> 10^6$	$9,741$	$3,045$	**256**	$> 10^6$
Rastrigin	5	10^{-2}	$255,385$	$> 10^6$	$> 10^6$	$> 10^6$	**$1,394$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$268,773$	$> 10^6$	$> 10^6$	$> 10^6$	**$4,394$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**$40,254$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**$97,238$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Schwefel	5	10^{-2}	$401,757$	$594,077$	$> 10^6$	$27,543$	**$4,954$**	$323,683$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$413,069$	$601,795$	$> 10^6$	**$39,487$**	$95,674$	$326,653$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**$304,914$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Styblinski-Tang	5	10^{-2}	$1,905$	$1,099$	$1,167$	$3,673$	**248**	$2,083$	271	$18,616$	$> 10^6$
		10^{-8}	$3,813$	**$2,317$**	$2,455$	$126,947$	$> 10^6$	$14,253$	$2,849$	$> 10^6$	$> 10^6$
	10	10^{-2}	$12,081$	$7,059$	$7,921$	$130,669$	**$3,800$**	$> 10^6$	$3,311$	$> 10^6$	$> 10^6$
		10^{-8}	$26,103$	**$15,473$**	$17,253$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$

Continued on next page

Table 3.3 Continued from previous page

Label	n	ε_{pe}	DIRECT-GL	DIRECT-G	DIRECT-L	DIRECT *V. 4.0*	BIRECT	DIRECT-l	PLOR	DISIMPL-V	DISIMPL-C
Styblinski-Tang	15	10^{-2}	$37,571$	**$22,919$**	$25,309$	$> 10^6$	$25,444$	$> 10^6$	$60,295$	$> 10^6$	$> 10^6$
		10^{-8}	$79,805$	**$48,917$**	$53,967$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Trid	6	10^{-2}	$8,629$	$4,927$	$11,037$	$4,897$	**753**	$9,147$	$> 10^6$	$26,627$	$> 10^6$
		10^{-8}	$42,347$	**$22,833$**	$56,685$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
	10	10^{-2}	$37,849$	**$22,263$**	$230,393$	$66,615$	$30,100$	$731,693$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**$319,897$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Zakharov	5	10^{-2}	$7,737$	$> 10^6$	$5,373$	$377,737$	$21,028$	$> 10^6$	$> 10^6$	**686**	$> 10^6$
		10^{-8}	$28,211$	$> 10^6$	$21,857$	$> 10^6$	$398,062$	$> 10^6$	$> 10^6$	**686**	$> 10^6$
	10	10^{-2}	$126,749$	$> 10^6$	**$95,345$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	$396,083$	$> 10^6$	**$288,311$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Average		10^{-2}	**$197,487$**	$363,499$	$417,331$	$440,388$	$316,065$	$632,939$	$596,375$	$693,600$	$943,461$
		10^{-8}	**$306,134$**	$447,221$	$470,644$	$716,233$	$827,201$	$766,266$	$688,450$	$883,567$	$944,243$
Median		10^{-2}	$45,162$	$66,611$	$110,293$	$105,798$	**$44,023$**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
		10^{-8}	**$154,703$**	$214,456$	$232,371$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
# of failed		10^{-2}	**3/32**	10/32	12/32	12/32	9/32	19/32	19/32	22/32	30/32
		10^{-8}	**4/32**	12/32	13/32	21/32	25/32	24/32	22/32	29/32	30/32

Concluded

The overall achievements solving test problems of all algorithms are shown in Fig. 3.3 performance profiles in the intervals $[1, 1000]$(with $\varepsilon_{\text{pe}} = 10^{-2}$ and $\varepsilon_{\text{pe}} = 10^{-8}$). Even when DIRECT-GL delivers the best overall averages of function evaluations, the performance profiles in Fig. 3.3 revealed that the solution with $\varepsilon_{\text{pe}} = 10^{-2}$ is needed, BIRECT has more wins and can solve about 60% of the problems with the smaller number of function evaluations. That is mainly because the set of potentially optimal hyper-rectangles in DIRECT-GL is larger per iteration, and furthermore BIRECT divides hyper-rectangles only by one longest dimension, which results in a smaller number of sampling points. Consequently, for DIRECT-GL, a higher number of function evaluations is needed. However, when a more accurate solution $\varepsilon_{\text{pe}} = 10^{-8}$ is needed, DIRECT-GL, and structural versions of an algorithm (DIRECT-G, DIRECT-L) are most effective and outperform all other DIRECT-type methods.

3.6 Conclusions

In this chapter, a new strategy for selecting the extended set of potentially optimal hyper-rectangles in the DIRECT-type algorithmic framework is introduced. In the proposed approach, two well-known weaknesses of DIRECT-type algorithms were addressed. Extensive experimental results confirmed the well-known fact that while for simpler problems, other DIRECT-type algorithms perform well, for more challenging (higher dimensional) problems, the proposed modified DIRECT-GL performs significantly faster. Moreover, since the set of potentially optimal hyper-rectangles is larger (compared to most DIRECT-type methods), the DIRECT-GL scheme looks promising for more efficient parallelization.

Chapter 4

Auxiliary function-based DIRECT-type algorithm for constrained global optimization

Many constrained optimization problems are formed from an engineering design process, where systems are often modeled with non-linear and multi-modal behavior, being low or high dimensional, computationally cheap, or expensive. Another difficulty of real-world engineering problems is constraints, which often allow feasible solutions only in a small subset of the design space or split the feasible region into many non-intersecting subsets. In most cases, practical engineering problems are complex and difficult to solve by traditional optimization methods. Issues of engineering and applied sciences can be formulated as non-linear programming global optimization problems [4, 23, 84, 104]. In this chapter, a global solution of the general non-linear programming problem Eq. (2.6) is sought.

4.1 Introduction

The original DIRECT algorithm performs well solving box-constrained global optimization problems, but it does not naturally address constraints. Several different constraint handling strategies are offered to be used within the DIRECT framework. The meta-model-based method has proved to be

effective, but the algorithm uses random point generation techniques, so it is no longer deterministic. In this chapter, a new DIRECT-type approach to problems with general constraints is presented. The introduced DIRECT-GLce algorithm incorporates a two-step selection procedure and auxiliary function approach in the previously described DIRECT-GL method. The proposed algorithm effectively explores hyper-rectangles with infeasible centers close to the boundaries of feasibility and may cover feasible regions. An extensive experimental investigation revealed the potential of the developed approach compared with other existing DIRECT-type algorithms for constrained global optimization problems, including important engineering problems.

4.2 DIRECT-GLce algorithm for generally constrained global optimization problems

In this section, the modification for general engineering optimization problems Eq. (2.6) to the earlier introduced DIRECT-GL algorithm is presented. The developed DIRECT-GLce algorithm uses an auxiliary function strategy that combines information of the objective and constraint functions and does not require any penalty parameters. The DIRECT-GLce algorithm operates in two phases, where during the first phase the algorithm handles infeasible initial sampling points while in the second phase seeks to find a feasible global solution. A separate phase for handling infeasible initial sampling points is especially useful when the feasible region is small compared to the design space D. When feasible solutions are located, the efforts may be switched to finding better feasible solutions. The developed algorithm is evaluated on the large set of test problems by comparing it with the existing approaches. In the following two subsections, main extensions to the DIRECT-GL algorithm are described.

4.2.1 Handling cases with infeasible initial regions

In this subsection, a new way to handle hyper-rectangles with infeasible centers is presented. In the first extension of the DIRECT-GL algorithm, we consider a situation when initial sampling points are infeasible and finding at least one

feasible point can be costly. In such a situation, the DIRECT-L1 algorithm is likely to fail to find a feasible point in a reasonable number of function evolutions. For such a situation, an additional procedure in the DIRECT-GL algorithm is employed, which samples the search space and minimizes not the original objective function, but the sum of constraint violations, i.e.:

$$\min_{\mathbf{x}\in D} \varphi(\mathbf{x}), \tag{4.1}$$

where

$$\varphi(\mathbf{x}) = \sum_{i=1}^{u} \max\{g_i(\mathbf{x}), 0\} + \sum_{i=1}^{v} |h_i(\mathbf{x})|, \tag{4.2}$$

until a feasible point $\mathbf{x} \in D^{\text{feas}}_{\varepsilon_\varphi}$ is found, where

$$D^{\text{feas}}_{\varepsilon_\varphi} = \{\mathbf{x} : 0 \leq \varphi(\mathbf{x}) \leq \varepsilon_\varphi, \mathbf{x} \in D\}, \tag{4.3}$$

where ε_φ is user predefined small acceptable constraint violation. The authors of the eDIRECTc algorithm use a very similar idea, but for treating the constraints equally, they suggest to normalize every constraint function. And in the same step, they sample the search space and minimize the sum of normalized constraint violations $\bar{\varphi}(\mathbf{x})$, i.e.,

$$\min_{\mathbf{x}\in D} \bar{\varphi}(\mathbf{x}). \tag{4.4}$$

In Table 4.1 the impact of this procedure on the selected subset of test problems from DIRECTlib [110] having a small feasible region is presented. For problems *G03*, *G05*, *G10* the L1 penalty based approaches can fail to produce a feasible solution within 10^6 function evaluations. However, when DIRECT employs the extra phase by minimizing Eq. (4.1) or Eq. (4.4), we avoid such situations.

4.2.2 Improving a feasible solution

By the second extension to the DIRECT-GL algorithm, the problem Eq. (2.6) is transformed to Eq. (4.5):

$$\begin{aligned} &\min_{\mathbf{x}\in D} f(\mathbf{x}) + \xi(\mathbf{x}, f^{\text{feas}}_{\min}), \\ &\xi(\mathbf{x}, f^{\text{feas}}_{\min}) = \begin{cases} 0, & \mathbf{x} \in D^{\text{feas}}_{\varepsilon_\varphi} \\ \varphi(\mathbf{x}) + \Delta, & \text{otherwise.} \end{cases} \end{aligned} \tag{4.5}$$

Table 4.1: The number of function evaluations needed by algorithms to find a feasible point

Label	n	$u+v$	α	DIRECT-GL		DIRECT-L1		
				$\varphi(\mathbf{x})$	$\bar{\varphi}(\mathbf{x})$	$r=10$	$r=10^2$	$r=10^3$
G01	13	9	0.0111%	**4,050**	4,270	4,626	4,244	4,776
G03	10	1	0.0000%	**1,381**	**1,381**	$>10^6$	$>10^6$	$>10^6$
G05	4	5	0.0000%	6,329	**5,658**	$>10^6$	$>10^6$	$>10^6$
G06	2	2	0.0066%	**102**	**102**	1,521	547	112
G07	10	8	0.0003%	927	1,628	**449**	531	813
G10	8	6	0.0010%	3,394	**1,813**	$>10^6$	$>10^6$	$>10^6$

a is the estimated ratio between the feasible region and the search space.

Presented the auxiliary function $\xi(\mathbf{x}, f_{\min}^{\text{feas}})$ depends on the sum of constraint functions $\varphi(\mathbf{x})$ and only one parameter $\Delta = |f(\mathbf{x}) - f_{\min}^{\text{feas}}|$, which is equal to absolute value of the difference between the best feasible function value found so far $f_{\min}^{\text{feas}}$ and the objective value at an infeasible center point $\mathbf{x}$.

The main purpose of this Δ parameter is to forbid the convergence of the algorithm to infeasible regions by penalizing the objective value obtained at the infeasible points. In this way, the formulation Eq. (4.5) does not require any penalty parameters, and convergence of the algorithm to a feasible solution is guaranteed. The value of $\xi(\mathbf{x}, f_{\min}^{\text{feas}})$ is updated when a smaller value of the $f_{\min}^{\text{feas}}$ is found. The new algorithm with these two extensions is called `DIRECT-GLc`. Note that this comes with a slight performance overhead compared with the existing `DIRECT` algorithm based on the penalty function approach `DIRECT-L1`, which uses the fixed penalty values during the entire minimization process.

4.2.3 Incorporating constraint tolerance in the `DIRECT-GLce` algorithm

At the beginning of the search, the difference between $f_{\min}^{\text{feas}}$ and the global solution f^* can be huge. Therefore the value of $\xi(\mathbf{x}, f_{\min}^{\text{feas}})$ can be increased too much, which can slow down the division of the required rectangles. Therefore, $\xi(\mathbf{x}, f_{\min}^{\text{feas}})$ values can be increased too much, which can slow down the selection of the necessary hyper-rectangles. By taking this into

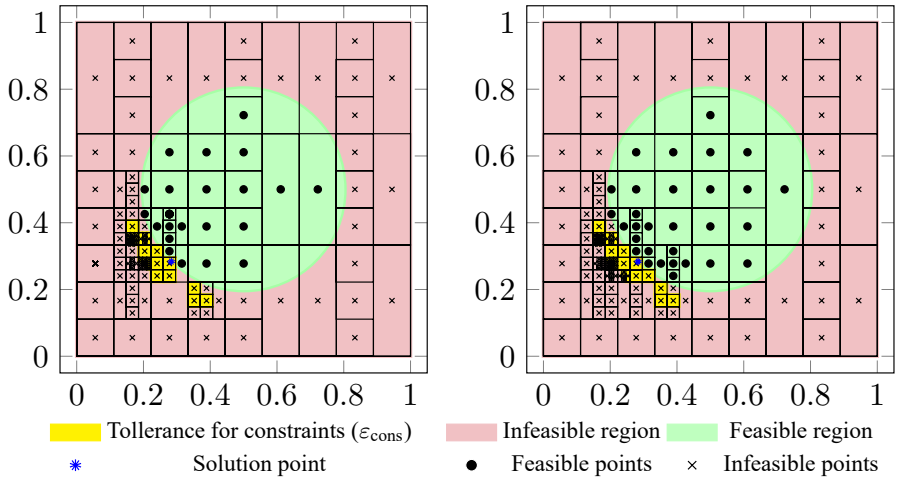

Figure 4.1: Geometric interpretation of the `DIRECT-GLce` algorithm on the *TI* ($n = 2$) test problem in the seventh iteration (left side), and the eighth iteration (right side).

account, Eq. (4.5) is modifying to Eq. (4.6):

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) + \tilde{\xi}(\mathbf{x}, f_{\min}^{\text{feas}}),$$
$$\tilde{\xi}(\mathbf{x}, f_{\min}^{\text{feas}}) = \begin{cases} 0, & \mathbf{x} \in D_{\varepsilon_\varphi}^{\text{feas}} \\ 0, & \mathbf{x} \in D_{\varepsilon_{\text{cons}}}^{\text{inf}} \\ \varphi(\mathbf{x}) + \Delta, & \text{otherwise,} \end{cases} \tag{4.6}$$

where $D_{\varepsilon_{\text{cons}}}^{\text{inf}} = \{\mathbf{x} : f(\mathbf{x}) \leq f_{\min}^{\text{feas}}, \varepsilon_\varphi < \varphi(\mathbf{x}) \leq \varepsilon_{\text{cons}}, \mathbf{x} \in D\}$ and $\varepsilon_{\text{cons}}$ is a small tolerance for constraint function sum, which automatically varies during the optimization process. More detailed behavior of $\varepsilon_{\text{cons}}$ is described in Algorithm 3, lines 20–28. With the introduction of this modification, the new `DIRECT-GLce` algorithm divides more hyper-rectangles with the center points lying close to the boundaries of the feasible region, i.e. potential solution. A geometrical illustration of $\varepsilon_{\text{cons}}$ parameter is shown in Fig. 4.1.

4.2.4 Hybridized `DIRECT-GLce-min` algorithm

Many hybridized versions have been proposed for box-constrained global optimization [45, 56, 73], and for problems with general constraints [52].

Such strategies are especially useful in solving high dimensional expensive multimodal problems. In this section, the DIRECT-GLce algorithm is enriched with a local minimization procedure (let us call the algorithm – DIRECT-GLce-min). If there is some improvement in the current best feasible minima value $f_{\min}^{\text{feas}}$, pure greedy search is performed from the current best feasible minima point $\mathbf{x}_{\min}^{\text{feas}}$. The more specific behavior of pure greedy search execution in the iteration is shown in Algorithm 3, Lines 29 and 31.

4.3 Algorithmic steps

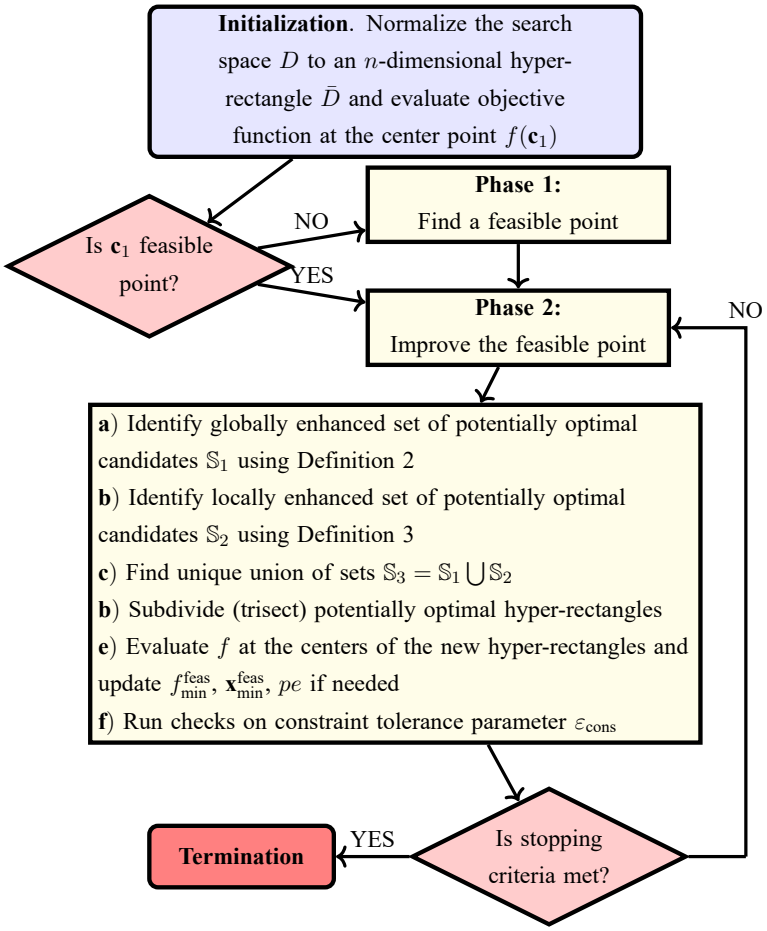

Figure 4.2: Flowchart of the DIRECT-GLce algorithm

The complete description of the DIRECT-GLce algorithm is given

in Algorithm 3 and additionally in Fig. 4.2. The input for the algorithm are the problem ($\mathcal{P}$) and one (or a few) stopping criteria: required tolerance (ε_{pe}), the maximal number of function evaluations ($\texttt{FE}_{\text{max}}$) and the maximal number of `DIRECT-GLce` iterations ($\texttt{K}_{\text{max}}$). After termination, the `DIRECT-GLce` algorithm returns the found objective value $f_{\text{min}}^{\text{feas}}$ and the solution point $\mathbf{x}_{\text{min}}^{\text{feas}}$ together with algorithmic performance measures: the final tolerance – percent error (pe), the total number of function evaluations (fe), and the total number of iterations (k).

The algorithm operates in two phases, which depends on whether a feasible point in D^{feas} is already found or not, see lines 6–10. If it is not yet found, the algorithm minimizes only the sum of constraint violation Eq. (4.2) and attempts to find a feasible point. After such a point is found, the algorithm switches to the second phase and minimizes Problem Eq. (4.6). Lines 20–28 are controlled by constraint tolerance parameter $\varepsilon_{\text{cons}}$ determining infeasible points which will not be penalized at all. In the proposed strategy, the number of such points (the cardinality of the set $D_{\varepsilon_{\text{cons}}}^{\text{inf}}$), cannot exceed $10 \times n^3$, if this happens $\varepsilon_{\text{cons}}$ should be reduced. In the opposite case when the cardinality of the set $D_{\varepsilon_{\text{cons}}}^{\text{inf}}$ is zero, $\varepsilon_{\text{cons}}$ should be increased. The boundaries for the rate of change is set to $10^{-4} \leq \varepsilon_{\text{cons}} \leq 10$.

4.4 Numerical investigation

Tested algorithms

In this section, an exhaustive comparison of the introduced `DIRECT-GLce` algorithm with the other existing `DIRECT`-type algorithms devoted to Eq. (2.6) problems is presented. First, algorithms with the `DIRECT-L1` based on the L1 penalty function approach and the simplicial partitioning-based `DISIMPL` methods are compared. In the `DIRECT-L1` algorithm, for each constraint, the penalty parameters for L1 functions are kept fixed during the optimization process. Three different penalty parameters were used ($r = 10$, $r = 10^2$, and $r = 10^3$) for all constraint functions. Next, in this thesis, the introduced algorithms are compared with the `Filter DIRECT`, `EPGO` and `DF-EPGO`. Finally, the algorithms are compared with the `eDIRECTc` version,

Algorithm 3: Pseudo code of the `DIRECT-GLce` algorithm

input : $\mathcal{P}$, ε_{pe}, ε_{φ}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$;
output: $f_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{\text{feas}}$, pe, k, fe;

Initialize $k = 1$, $fe = 1$, $\mathbf{x}_{\min}^{k} = \mathbf{c}_1$, $\varepsilon_{\text{cons}} = 1$, $\text{card}_{\text{limit}} = 10 \times n^3$, $\varsigma = 0$, $\mathbb{I}^k = \{1\}$;
if $\exists \mathbf{x}_1 \in D_{\varepsilon_{\varphi}}^{\text{feas}}$ **then**
    Update $f_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{\text{feas}}$ and pe;
end
while $pe > \varepsilon_{\text{pe}}$ **and** $fe < \mathsf{FE}_{\max}$ **and** $k < \mathsf{K}_{\max}$ **do** `// pe` (3.7)
    if $\exists \mathbf{c}_i \in D_{\varepsilon_{\varphi}}^{\text{feas}}$ **then** `// Phase II (Improve the best feasible point)`
        $F = \{f(\mathbf{c}_i) + \tilde{\xi}(\mathbf{c}_i, f_{\min}^{\text{feas}}), \mathbf{c}_i \in \bar{D}, i = 1, \ldots, fe\}$;
    else `// Phase I (Find feasible point)`
        $F = \{\varphi(\mathbf{c}), \mathbf{c}_i \in D, i = 1, \ldots, fe\}$;
    end
    Identify the "global" (index) set $\mathbb{S}_1^k \subseteq \mathbb{I}^k$ of POH on F using Definition 2 ;
    Identify the "local" (index) set $\mathbb{S}_2^k \subseteq \mathbb{I}^k$ of POH on d (3.2) using Definition 3 ;
    Find unique union of potential optimal hyper-rectangles sets $\mathbb{S}_3^k = \mathbb{S}_1^k \bigcup \mathbb{S}_2^k$;
    foreach $j \in \mathbb{S}_3^k$ **do**
        Subdivide (trisect) hyper-rectangle D_j^k and update $\mathbb{I}^k$;
        Evaluate f at the centers of the new hyper-rectangles and update fe;
    end
    if $\exists \mathbf{c}_i \in D_{\varepsilon_{\varphi}}^{\text{feas}}$ **then** `// Phase II`
        Update $f_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{k}$, pe and increase $k = k + 1$;
        if $\varepsilon_{\text{cons}} == \varepsilon_{\varphi}$ **and** $\varsigma \geq 10$ **then** `// Control model of` $\varepsilon_{\text{cons}}$
            $\varepsilon_{\text{cons}} = 1$ and extend limit of $\text{card}(D_{\varepsilon_{\text{cons}}}^{\text{inf}})$: $\text{card}_{\text{limit}} = \text{card}_{\text{limit}} \times 10$;
        else if $D_{\varepsilon_{\text{cons}}}^{\text{inf}} ==$ **and** $\varepsilon_{\text{cons}} \times 3 \leq 10$ **then**
            Increase tolerance of constraints: $\varepsilon_{\text{cons}} = \varepsilon_{\text{cons}} \times 3$;
        else if $\text{card}(D_{\varepsilon_{\text{cons}}}^{\text{inf}}) \geq \text{card}_{\text{limit}}$ **and** $\varepsilon_{\text{cons}}/3 \geq \varepsilon_{\varphi}$ **then**
            Reduce tolerance of constraints: $\varepsilon_{\text{cons}} = \varepsilon_{\text{cons}}/3$;
        else if $\text{card}(D_{\varepsilon_{\text{cons}}}^{\text{inf}}) \geq \text{card}_{\text{limit}}$ **and** $\varepsilon_{\text{cons}}/3 \leq \varepsilon_{\varphi}$ **then**
            Set tolerance of constraints: $\varepsilon_{\text{cons}} = \varepsilon_{\varphi}$;
        end
        if $| f_{\min}^{k} - f_{\min}^{k-1} | \geq 10^{-2}$ **then** `// Only in DIRECT-GLce-min`
            Perform the local search from $\mathbf{x}_{\min}^{\text{feas}}$ and update $f_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{\text{feas}}$, fe and pe;
        end
    else `// Phase I`
        Update $\mathbf{x}_{\min}^{k}$ and increase $k = k + 1$;
    end
    if $d(\mathbf{x}_{\min}^{k}, \mathbf{x}_{\min}^{k-1}) \geq 10^{-6}$ **then**
        Set $\varsigma = 0$ and calculate distances d (3.2), $\mathbf{c}_i \in D, i = 1, \ldots, fe$;
    else
        Set $\varsigma = \varsigma + 1$ and calculate distances d (3.2), $\mathbf{c}^i \in D, i = fe_{\text{old}}, \ldots, fe_{\text{new}}$;
    end
end
return $f_{\min}^{\text{feas}}$, $\mathbf{x}_{\min}^{\text{feas}}$, pe, k, fe;

and experimental investigation is concluded by applying the algorithms to five important real-world engineering problems.

DIRECTLib: collection of test and practical benchmark problems

Throughout this thesis, we aimed to compare the developed algorithms with state-of-the-art DIRECT-type algorithms under equal conditions, i.e., using the same stopping criteria and benchmark problems. Therefore, we collected a new library, DIRECTlib [110], consisting of test and practical real-world optimization problems considered by the other authors. In total, we have combined 84 test and five challenging engineering problems: *tension/compression spring*, *three-bar truss*, *NASA speed reducer*, *pressure vessel*, and *welded beam*.

Table 4.2: Key characteristics of the optimization test problems with general constraints from DIRECTlib[110]

Label	Source	n	Cons. type L	NL	EQ	D	f^*
Bunnag 1	[121]	4	1	0	0	$[0,3]_i, i=1...n$	0.1117
Bunnag 2	[121]	4	2	0	0	$[0,4]_i, i=1...n$	-6.4049
Bunnag 3	[121]	5	3	0	0	$[0,3]_1, [0,2]_{2,5}, [0,4]_{3,4}$	-16.3657
Bunnag 4	[121]	6	2	0	0	$[0,1]_{1,2,3,4,5}, [0,20]_6$	-213.0470
Bunnag 5	[121]	6	5	0	0	$[0,2]_{1,3,6}, [0,8]_2, [0,1]_{4,5}$	-11.0000
Bunnag 6	[121]	10	11	0	0	$[0,1]_i, i=1...n$	-268.0146
Bunnag 7	[121]	10	5	0	0	$[0,1]_i, i=1...n$	-39.0000
circle	[121]	3	0	10	0	$[0,10]_i, i=1...n$	4.5742
G01	[52]	13	9	0	0	$[0,10]_i, [0,100]_{10}, i=1...9,11$	-15.0000
G02	[52]	20	1	1	0	$[0,10]_i, i=1...n$	-0.8036
G03	[52]	10	0	0	1	$[0,10]_i, i=1...n$	-1.0005
G04	[52]	5	0	6	0	$[78,102]_1, [33,45]_2, [27,45]_{3,4,5}$	-30665.5386
G05	[52]	4	2	0	3	$[10,1,200]_{1,2}, [-0.55,0.55]_{3,4}$	5126.4967
G06	[52]	2	0	2	0	$[13,100]_1, [0,100]_2$	-6961.8138
G07	[52]	10	2	5	0	$[-10,10]_i, i=1...n$	24.3062
G08	[52]	2	0	2	0	$[0,10]_i, i=1...n$	-0.0958
G09	[52]	7	0	4	0	$[-10,10]_i, i=1...n$	680.6300
G10	[52]	8	3	3	0	$[100,10,000]_1, [1,000,10,000]_{2,3}$, $[10,1,000]_i, i=4...8$	7049.2480
G11	[52]	2	0	0	1	$[-1,1]_i, i=1...n$	0.7499
G12	[52]	3	0	1	0	$[0.2,10]_i, i=1...n$	-1.0000
G13	[52]	5	0	0	3	$[-2.3,2.3]_{1,2}, [-3.2,3.2]_{3,4,5}$	0.0539
G16	[115]	5	4	34	0	$[704.4148,906.3855]_1, [68.6,288.88]_2$, $[0,134.75]_3, [193,287.0966]_4, [25,84.1988]_5$	-1.9051

Continued on next page

Table 4.2 Continued from previous page

Label	Source	n	Cons. type L	NL	EQ	D	f^*
G18	[115]	9	0	13	0	$[0,10]_i, i=1...n$	-0.8660
G19	[115]	15	0	5	0	$[0,10]_i, i=1...n$	32.6555
G24	[115]	2	0	2	0	$[0,3]_1, [0,4]_2$	-5.5080
Genocop 9	[121]	3	5	0	0	$[0,10]_i, i=1...n$	-2.4714
Genocop 10	[121]	4	5	0	0	$[0,3]_1, [0,10]_{2,3}, [0,1]_4$	-4.5280
Genocop 11	[121]	6	5	0	0	$[0,5]_{1,3}, [0,8]_2, [0,1]_{4,5}, [0,2]_6$	-11.0000
Gold. & Pr.	[65]	2	0	2	0	$[-2,2]_i, i=1...n$	3.5389
Gomez	[5]	2	0	1	0	$[-1,1]_i, i=1...n$	-0.9711
Himmelblau	[7]	5	0	5	0	$[78,102]_1, [33,45]_2, [27,45]_{3,4,5}$	-31025.5602
Horst 1	[41]	2	3	0	0	$[0,3]_1, [0,2]_2$	-1.0625
Horst 2	[41]	2	3	0	0	$[0,2.5]_1, [0,2]_2$	-6.8995
Horst 3	[41]	2	3	0	0	$[0,1]_1, [0,1.5]_2$	-0.4444
Horst 4	[41]	3	4	0	0	$[0.5,2]_1, [0,3]_2, [0,2.8]_3$	-6.0858
Horst 5	[41]	3	4	0	0	$[0,1.2]_{1,2}, [0,1.7]_3$	-3.7220
Horst 6	[41]	3	7	0	0	$[0,6]_1, [0,5.0279]_2, [0,2.6]_3$	-32.5784
Horst 7	[41]	3	4	0	0	$[0,6]_1, [0,3]_{2,3}$	-52.8769
hs021	[121]	2	1	0	0	$[2,50]_1, [-50,10]_2$	-99.9599
hs021mod	[121]	7	3	0	0	$[2,50]_1, [-50,50]_2, [0,50]_3, [2,10]_4,$ $[-10,10]_5, [-10,0]_8, [0,10]_7$	4.0400
hs024	[121]	2	3	0	0	$[0,5]_i, i=1...n$	-1.0000
hs035	[121]	3	1	0	0	$[0,3]_i, i=1...n$	0.1111
hs036	[121]	3	1	0	0	$[0,20]_1, [0,11]_2, [0,15]_3$	-3300.0000
hs037	[121]	3	2	0	0	$[0,42]_i, i=1...n$	-3456.0000
hs038	[121]	4	2	0	0	$[-10,10]_i, i=1...n$	0.0000
hs044	[121]	4	6	6	0	$[0,5]_i, i=1...n$	-15.0000
hs076	[121]	4	3	0	0	$[0,1]_{1,3,4}, [0,3]_2$	-4.6818
P01	[5]	5	0	0	3	$[-5,5]_i, i=1...n$	0.0293
P02a	[5]	9	4	6	0	$[0,100]_1, [0,500]_i, 2=1...n$	-400.0000
P02b	[5]	9	4	6	0	$[0,600]_1, [0,500]_i, 2=1...n$	-600.0000
P02c	[5]	9	4	6	0	$[0,100]_1, [0,500]_i, 2=1...n$	-750.0000
P02d	[5]	10	6	6	0	$[0,300]_{1,2,6}, [0,100]_{3,5,7}, [0,200]_{4,8},$ $[0,3]_9$	-600.0000
P03a	[5]	6	0	1	4	$[0,1]_{1,2,3,4}, [10^{-5},16]_{5,6}$	0.3888
P03b	[5]	2	0	1	0	$[10^{-5},16]_i, i=1...n$	0.3888
P04	[5]	2	0	1	0	$[0,6]_1, [0,4]_2$	-6.6666
P05	[5]	3	0	2	2	$[0,9.422]_1, [0,5.903]_2, [0,267.42]_3$	201.1600
P06	[5]	2	0	1	0	$[0,115.8]_1, [10^{-5},30]_2$	376.2900
P07	[5]	2	2	2	0	$[-2,2]_i, i=1...n$	-2.8284
P08	[5]	2	1	1	0	$[-8,10]_1, [0,10]_2$	-118.7000
P09	[5]	6	9	0	0	$[10^{-5},3]_1, [10^{-5},4]_{2,3}, [0,2]_{4,5}, [0,6]_6$	-13.4020
P10	[5]	2	0	2	0	$[0,1]_i, i=1...n$	0.7417
P11	[5]	2	0	1	0	$[0,1]_i, i=1...n$	-0.5000
P12	[5]	2	0	2	0	$[0,2]_1, [0,3]_2$	-16.7390
P13	[5]	3	2	1	2	$[10^{-5},34]_1, [10^{-5},17]_2, [100,300]_3$	189.3500
P14	[5]	4	4	0	0	$[10^{-5},3]_1, [10^{-5},4]_2, [0,2]_3, [0,1]_4$	-4.51420
P15	[5]	3	0	0	3	$[10^{-5},12.5]_1, [10^{-5},37.5]_2, [0,50]_3$	0.0000
P16	[5]	5	2	4	0	$[0,1.5834]_1, [0,3.625]_2, [0,1]_3,$ $[0,3]_4, [0,4]_5$	0.7049

Continued on next page

Table 4.2 Continued from previous page

Label	Source	n	Cons. type L	NL	EQ	D	f^*
s224	[121]	2	4	0	0	$[0,6]_1,[0,11]_2$	-304.0000
s231	[121]	2	2	0	0	$[-10,10]_i, i=1...n$	0.0000
s232	[121]	2	3	0	0	$[0,100]_i, i=1...n$	-1.0000
s250	[121]	3	2	0	0	$[0,20]_1,[0,11]_2,[0,42]_3$	-3300.0000
s251	[121]	3	1	0	0	$[0,42]_i, i=1...n$	-3456.0000
s365mod	[121]	7	0	1	0	$[0,19]_i, i=1...n$	52.1399
T1 $(n=2)$	[19]	2	0	1	0	$[-4,4]_i, i=1...n$	-3.4641
T1 $(n=3)$	[19]	3	0	1	0	$[-4,4]_i, i=1...n$	-4.2426
T1 $(n=4)$	[19]	4	0	1	0	$[-4,4]_i, i=1...n$	-4.8989
T1 $(n=5)$	[19]	5	0	1	0	$[-4,4]_i, i=1...n$	-5.4772
T1 $(n=6)$	[19]	6	0	1	0	$[-4,4]_i, i=1...n$	-6.0000
T1 $(n=7)$	[19]	7	0	1	0	$[-4,4]_i, i=1...n$	-6.4807
T1 $(n=8)$	[19]	8	0	1	0	$[-4,4]_i, i=1...n$	-6.9282
zecevic2	[121]	3	3	0	0	$[0,10]_i, i=1...n$	-4.1249
zecevic3	[121]	2	3	0	0	$[0,10]_i, i=1...n$	97.3094
zecevic4	[121]	4	3	0	0	$[0,10]_i, i=1...n$	7.5575
zy2	[121]	2	2	1	0	$[0,10]_i, i=1...n$	2.0000

Concluded

In Appendix A, we provide a short description and mathematical formulations. The key characteristics and descriptions of all the test problems used in this section are given in Table 4.2, and a `MatLab` format is given in the same `DIRECTlib` [110] online resource. One of the test problems, *T1*, can be tested on any dimensionality. All the computations were carried out on a six-core computer with the 8th Generation Intel R Core^{TM} i7-8750H @ 2.20GHz Processor, 16 GB of RAM and `MatLab` R2020a. Performance analysis was carried out using physical cores only with disabled hyper-threading.

Stopping criteria

Two different stopping conditions were used for the algorithms, one of them is the same as was used in the previous chapter (pe) Eq. (3.7). In [11] the authors consider a slightly different way to calculate the percent error $(\tilde{pe})$:

$$\tilde{pe} = \frac{|f(\mathbf{x}) - f^*|}{\max\{1, |f^*|\}}. \tag{4.7}$$

4.4.1 Group 1: comparison with DIRECT-L1 and DISIMPL algorithms

Experimental results using the introduced methods are presented in Table 4.3, and additionally, Fig. 4.3 shows performance profiles on the overall achievements of the algorithm. Here, in the first column (Label), the name of the problem is reported, while in the second it is the dimensionality (n) of the problem. In the third column (Cons. type), the type of constraints is specified: linear (L) or non-linear (NL). Next, in the consecutive columns, the total number of function evaluations are reported using six different algorithms: `DIRECT-GLc`, `DIRECT-GLce`, `DIRECT-GLce-min`, `DIRECT-L1`, `Lc-DISIMPL` and `Lv-DISIMPL` accordingly. No constraint violation was allowed in this experiment and the parameter ε_{φ} was set to 0. The investigated algorithms were stopped either when the point $\mathbf{x}$ was generated such that the percent error (pe) Eq. (3.7) is smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-2}$, or when the number of function evaluations exceeds the prescribed limit of 10^6.

First, it is easy to notice that in almost all tested instances the hybridized `DIRECT-GLce-min` algorithm gives the best performance and, on average, uses $24,304$ function evaluations, i.e., more than six times less compared to the original second-best `DIRECT-GLce` algorithm. Moreover, the `DIRECT-GLce-min` algorithm failed to solve only one high dimensional 20 optimization test problem *G02*.

Next, comparing the penalty and auxiliary function-based methods, for the low-dimensional test problems ($n \leq 3$) the number of function evaluations is most often smaller for the `DIRECT-GLc` algorithm 62% $(18/29)$. The $\varepsilon_{\text{cons}}$ parameter in the `DIRECT-GLce` algorithm forces to subdivide more hyper-rectangles for easier test problems (low dimension and with linear constrains) comparing with other algorithms, but solving more complicated test problems the `DIRECT-GLce` algorithm is more promising. The main advantage of the $\varepsilon_{\text{cons}}$ parameter can be seen in solving higher-dimensional and non-linear (NL) test problems, where the `DIRECT-GLce` algorithm outperforms other methods in average function evaluations and solved problems. Also, looking in a general context, the `DIRECT-GLce` algorithm

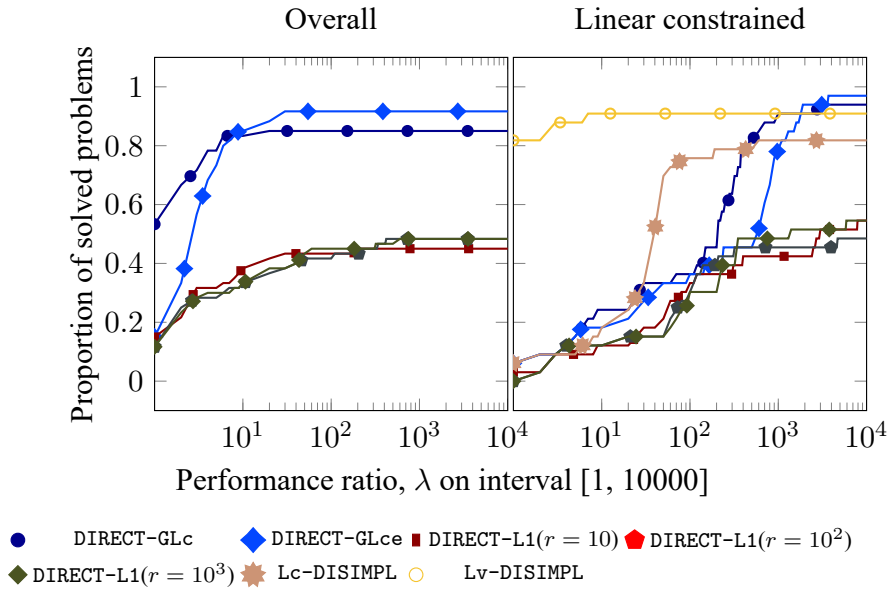

Figure 4.3: Performance profiles of DIRECT-type algorithms on the whole set of constrained optimization test problems from DIRECTlib

requires fewer function evaluations and fails to solve only five test problems.

None of the tested, fixed penalty parameters for the L1 penalty function can guarantee the convergence to the feasible solution for all tested problems. the DIRECT-L1 algorithm works better using smaller penalty parameters ($r = 10$). Larger penalty parameter values ($r = 10^3$) reduce the chance of obtaining a solution from the infeasible region. On the other hand, larger penalty values can bias the algorithm away from the feasible region's boundary where the solution is often located.

The main advantage of the L-DISIMPL algorithms is that simplices cover a feasible region bounded by linear constraints, and infeasible regions are not involved in the search. Unfortunately, an algorithm can be used to solve problems only with linear constraints. Furthermore, solving such problems are often the solutions are located on the intersection of the linear constraints, where the Lv-DISIMPL algorithm can find the solution point in the initialization step by sampling only vertices of the simplices. Performance profiles Fig. 4.3 show the Lv-DISIMPL algorithm's effectiveness, amongst all other DIRECT-type methods solving linear constrained global optimization

problems. However, the efficiency of the L-DISIMPL suffers from dimensionality. Three of the test problems were unsolved by the Lv-DISIMPL algorithm, where the dimension is greater than 10, while the second-best algorithm DIRECT-GLce fails to solve only one test problem with linear constraints.

The performance profiles in Fig. 4.3 reveal that the DIRECT-GLc algorithm has the most wins, and it can solve about 80% of the problems with the highest efficiency. However, solving more challenging problems (with non-linear constraints and $n \geq 4$), the DIRECT-GLce algorithm outperforms the other versions, and the performance difference increases as the performance ratio increases.

Table 4.3: Number of function evaluations and time (in sec.) solving problems from DIRECTlib

Label	n	Cons. type L	Cons. type NL	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1 ($r = 10$)	DIRECT-L1 ($r = 10^2$)	DIRECT-L1 ($r = 10^3$)	Lc-DISIMPL	Lv-DISIMPL
Bunnag 1	4	1	0	2,211	6,531	**31**	411,369	97,431	97,431	1,350	3,114
Bunnag 2	4	2	0	6,051	13,191	35	6,399	$> 10^6$	$> 10^6$	442	**16**
Bunnag 3	5	3	0	10,821	34,305	34,305	3,747	3,427	3,293	17,025	**34**
Bunnag 4	6	2	0	8,435	39,649	147	1,607	14,919	15,877	1,726	**72**
Bunnag 5	6	5	0	24,133	68,393	503	$> 10^6$	$> 10^6$	$> 10^6$	4,272	**97**
Bunnag 6	10	11	0	$> 10^6$	719,701	**2,763**	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$
Bunnag 7	10	5	0	98,793	98,853	**14,285**	18,375	28,307	28,419	$> 10^6$	$> 10^6$
circle	3	0	10	3,591	9,717	**1,161**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G01	13	9	0	369,163	721,509	**8,361**	7^α	7^α	7^α	$> 10^6$	$> 10^6$
G02	20	0	2	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G04	6	0	6	7,649	20,067	**49**	33^α	33^α	759	–	–
G06	2	0	2	3,773	6,833	**241**	51^α	97^α	297^α	–	–
G07	10	3	5	$> 10^6$	$> 10^6$	**2,771**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G08	2	0	2	**443**	1,007	463	327^α	453	453	–	–
G09	7	0	4	38,491	74,011	**583**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G10	8	3	3	$> 10^6$	$> 10^6$	**57,205**	57^α	57^α	205^α	–	–
G12	3	0	1	143	179	179	**111**	**111**	123	–	–

Continued on next page

Table 4.3 Continued from previous page

Label	n	Cons. type L	Cons. type NL	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1 ($r = 10$)	DIRECT-L1 ($r = 10^2$)	DIRECT-L1 ($r = 10^3$)	Lc-DISIMPL	Lv-DISIMPL
G16	5	4	34	102,811	90,385	**537**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G18	9	0	13	56,185	336,149	**673**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G19	15	0	5	$> 10^6$	$> 10^6$	**4,835**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
G24	2	0	2	877	2,677	**11**	7,783	292,259	$> 10^6$	–	–
Genocop 9	4	5	0	3,745	10,649	169	13^α	13^α	13^α	213	**6**
Genocop 10	4	5	0	5,545	21,727	155	14,849	$> 10^6$	$> 10^6$	106,550	**6**
Genocop 11	4	5	0	230,839	$> 10^6$	155	$> 10^6$	$> 10^6$	$> 10^6$	4,489	**84**
Gold. and Pr.	2	0	2	551	3,013	**65**	119^α	617	1,165	–	–
Gomez	2	0	1	795	1,323	**113**	6,093	168,001	471,967	–	–
Himmelblau's	5	0	5	5,577	20,973	**33**	67^α	67^α	$3,243^\alpha$	–	–
Horst 1	2	3	0	1,403	4,411	23	287^α	32,583	$> 10^6$	253	**7**
Horst 2	2	3	0	705	2,507	39	265^α	585	29,063	173	**5**
Horst 3	2	3	0	721	721	177	283	283	283	249	**5**
Horst 4	3	4	0	2,171	6,251	143	23,931	$> 10^6$	$> 10^6$	284	**8**
Horst 5	3	4	0	2,425	6,617	71	$4,503^\alpha$	243,471	$> 10^6$	259	**8**
Horst 6	3	7	0	4,119	9,733	19	333^α	$9,351^\alpha$	$> 10^6$	71	**12**
Horst 7	3	4	0	2,051	6,511	29	543	679	679	231	**10**
hs021	2	1	0	123	123	**5**	153	817	1,273	72	6
hs021mod	7	3	0	$> 10^6$	288,659	145	$> 10^6$	$> 10^6$	$> 10^6$	$> 10^6$	**18**

Continued on next page

Table 4.3 Continued from previous page

Label	n	Cons. type L	Cons. type NL	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1 ($r = 10$)	DIRECT-L1 ($r = 10^2$)	DIRECT-L1 ($r = 10^3$)	Lc-DISIMPL	Lv-DISIMPL
hs024	2	3	0	801	2,647	5	10,689	334,439	$> 10^6$	136	**4**
hs035	3	1	0	2,369	6,491	**25**	5,297	2,301	2,263	717	630
hs036	3	1	0	1,803	1,895	389	727	727	727	1,492	**8**
hs037	3	2	0	3,739	7,127	**17**	7^α	7^α	64,423	9,127	186
hs038	4	2	0	10,869	9,803	**159**	7,189	5,889	5,317	$> 10^6$	2,518
hs044	4	6	0	7,463	25,097	149	156,733	$> 10^6$	$> 10^6$	386	**20**
hs076	4	3	0	3,583	12,485	**31**	32,771	137,413	142,857	443	941
s224	2	4	0	413	1,395	**5**	7^α	329	1,225	463	6
s231	2	2	0	413	413	**113**	999	983	885	3,017	2,613
s232	2	3	0	1,387	5,579	145	19^α	57^α	$> 10^6$	141	**3**
s250	3	2	0	4,663	7,369	19	25^α	49^α	11,135	373	**8**
s251	3	1	0	1,185	7,655	**11**	7^α	7^α	61,401	9,127	186
s365mod	7	0	1	$> 10^6$	**325, 325**	**325, 325**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
T1	2	0	1	1,637	3,103	**33**	429	567	567	–	–
T1	3	0	1	6,573	11,187	**63**	5,315	14,777	10,251	–	–
T1	4	0	1	19,435	20,579	**127**	48,745	199,727	217,879	–	–
T1	5	0	1	$> 10^6$	89,391	**245**	196,587	473,677	647,155	–	–
T1	6	0	1	117,815	169,347	**371**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
T1	7	0	1	621,511	121,353	**303**	$> 10^6$	$> 10^6$	$> 10^6$	–	–

Continued on next page

Table 4.3 Continued from previous page

Label	n	Cons. type L	Cons. type NL	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1 ($r = 10$)	DIRECT-L1 ($r = 10^2$)	DIRECT-L1 ($r = 10^3$)	Lc-DISIMPL	Lv-DISIMPL
TI	8	0	1	$> 10^6$	363,857	**119**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
zecevic2	3	3	0	783	3,363	57	1,607	593	593	336	**34**
zecevic3	2	3	1	907	2,227	**13**	$> 10^6$	271	307	–	–
zecevic4	4	3	1	1,799	2,231	**13**	1,431	20,065	449,787	–	–
zy2	2	2	1	4,993	12,129	**53**	27,607	$> 10^6$	$> 10^6$	–	–
Average(overall)				180,141	147,307	**24,304**	566,523	551,262	554,459	–	–
Average(Non-Linear constraints)				296,132	210,632	**51,688**	677,559	635,945	659,275	–	–
Average(Linear constraints)				85,240	95,496	**1,900**	475,675	481,975	468,701	156,467	91,237
Average ($n \geq 4$)				358,245	289,442	**48,485**	629,993	732,695	720,292	–	–
Average ($n \leq 3$)				1,916	4,628	**127**	485,916	348,098	367,544	–	–
Median				4,391	10,226	**144**	$> 10^6$	$> 10^6$	$> 10^6$	–	–
# of failed				9/60	5/60	**1/60**	33/60	31/60	31/60	5/33	3/33

α - result is outside the feasible region

Concluded

4.4.2 Group 2: comparison with `Filter DIRECT`, `EPGO` and `DF-EPGO` algorithms

The proposed algorithms compared the `Filter DIRECT`, `EPGO`, and `DF-EPGO` algorithms in the second part. The same 20 global optimization test problems (*P01–P16*) used in [11] and collected from [5] were considered, which are also is included in `DIRECTlib`. The obtained experimental results are presented in Section 4.4.2. Here, in the first column (Label), the name of the problem is reported, while in the second it is the dimensionality (n) of the problem. In the third column (Cons. type), the type of constraints are specified: linear (L), non-linear (NL) or (EQ) equality. Next, in the consecutive columns, the total number of function evaluations f_{eval} required by an algorithm to reach the solution within a specified accuracy and the minimal objective function value f_{min} founded by the corresponding algorithm are reported using six different algorithms: `DIRECT-GLc`, `DIRECT-GLce`, `DIRECT-GLce-min`, `Filter DIRECT`, `EPGO` and `DF-EPGO` accordingly. The results available in [11, 82, 83], an exact penalties approach that relies on `DIRECT` to solve these test problems, together with filter-based `DIRECT`, are also included in the comparison. Note that for the `DF-EPGO` algorithm no number of function evaluations is given in the cited paper.

To provide comparison as fair as a possible, in the same vein as in [11], algebraic manipulation aiming to reduce the number of variables and equality constraints has been performed:

- Test problems *P02a*, *P02b* and *P02c* after reformulation contain 5 variables and 10 inequality constraints. In the original problem formulation, there were 9 variables, 4 equality, and 2 inequality constraints.
- Test problem *P02d* after reformulation contains 5 variables and 12 inequality constraints. In the original problem formulation, there were 10 variables, 5 equality, and 2 inequality constraints.
- Test problem *P05* after reformulation contains 2 variables, 2 equality, and 2 inequality constraints. In the original problem formulation, there were 3 variables and 3 equality constraints.
- Test problem *P09* after reformulation contains 3 variables and 9

Table 4.4: Comparison between different DIRECT-type algorithm solving generally constrained global optimization problems from DIRECTlib

Label	n	f^*	Cons. type			DIRECT-GLc		DIRECT-GLce		DIRECT-GLce-min		Filter DIRECT		EPGO		DF-EPGO
			L	NL	EQ	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	$f_{\min}$
P01	5	0.0293	0	0	3	106,395	0.0293	130,393	0.0293	**56,689**	**0.0292**	25,425	0.3989	39,575	0.0625	0.0625
P02a	9	−400.0000	4	6	0	200,000	−397.0355	200,000	−397.1475	**4,609**	**−400.0000**	697,169	−22.4449	115,107	−134.1127	−134.0983
P02b	9	−600.0000	4	6	0	200,000	−397.0347	200,000	−397.0719	200,000	−400.0000	421,197	53.6867	120,057	-768.4569^{α}	-705.1318^{α}
P02c	9	−750.0000	4	6	0	200,000	−701.4837	200,000	−701.4837	**3,797**	**−749.9997**	724,337	−38.7948	102,015	82.9774	−82.9528
P02d	10	−400.0000	6	6	0	19,193	−399.9610	58,253	−399.9712	**207**	**−400.0000**	16,715	−399.9661	229,773	−385.1704	−399.7635
P03a	6	−0.3888	0	1	4	75,757	**−0.3888**	**136,095**	**−0.3888**	**136,095**	**−0.3888**	1,109,995	−0.3832	48,647	−0.3861	0.3861
P03b	2	−0.3888	0	1	0	711	−0.3888	1,641	−0.3888	**33**	−0.3888	347	**−0.3889**	3,449	−0.3888	0.3888
P04	2	−6.6666	0	1	0	397	−6.6662	1,623	−6.6662	**27**	−6.6663	543	−6.6662	3,547	**−6.6666**	**−6.6666**
P05	3	201.1593	0	2	2	1,597	201.1593	1,597	201.1593	1,597	201.1593	**1,009**	201.1593	14,087	201.1593	**201.1592**
P06	2	376.2919	0	1	0	771	376.3253	2,613	376.3133	**33**	**376.2919**	1,323	376.3002	1,523	0.4701^{α}	376.2927
P07	2	−2.8284	2	2	0	637	−2.8282	2,211	−2.8282	**21**	**−2.8284**	1,417	−2.8282	13,187	−2.8058	−2.8059
P08	2	−118.7049	1	1	0	1,267	−118.6895	1,611	−118.6902	**65**	**−118.7049**	883	−118.7010	7,621	−118.7044	−118.7048
P09	6	−13.4018	9	0	0	2,275	−13.4018	9,261	−13.4018	**95**	−13.4019	2,203	−13.4018	68,177	**−13.4026**	−13.4013
P10	2	0.7417	0	2	0	637	0.7418	2,329	0.7418	2,329	0.7418	**587**	0.7418	6,739	0.7420	**0.7417**
P11	2	−0.5000	0	1	0	753	−0.4999	3,089	**−0.5000**	3,089	**−0.5000**	**5**	**−0.5000**	3,579	**−0.5000**	**−0.5000**
P12	2	−16.7388	0	2	0	27	−16.7387	27	−16.7386	**13**	**−16.7389**	6,665	−16.7388	3,499	**−16.7389**	−16.7388
P13	3	189.3476	2	1	2	45,589	189.3578	42,015	189.3627	**19,227**	**189.3476**	10,583	195.3399	8,085	195.9553	195.9454
P14	4	−4.5142	4	0	0	2,135	−4.5137	8,303	−4.5137	**47**	**−4.5142**	1,967	−4.5140	19,685	−4.3460	−4.35233
P15	3	0.0000	0	0	3	139	**0.0000**	139	**0.0000**	139	**0.0000**	**105**	**0.0000**	1,645	**0.0000**	**0.0000**
P16	5	0.7049	2	4	0	89	**0.7049**	93	**0.7049**	**7**	**0.7049**	151	0.7050	22,593	0.7181	0.7180
Average(overall)						43,065		50,124		**21,405**		151,131		41,629		–
Average ($n \geq 4$)						89,864		104,843		**44,616**		499,140		85,069		–
Average ($n \leq 3$)						4,775		5,354		2,415		**1,984**		6,087		–
Median						1,432		2,851		**173**		1,692		13,637		–
# of failed						3/20		3/20		**1/20**		6/20		11/20		10/20

inequality constraints. In the original problem formulation, there were 6 variables, 3 equality, and 3 inequality constraints.

- Test problem *P12* after reformulation contains 1 variable and 2 inequality constraints. In the original problem formulation, there were 2 variables and 1 equality constraints.
- Test problem *P14* after reformulation contains 3 variables and 4 inequality constraints. In the original problem formulation, there were 4 variables, 1 equality, and 2 inequality constraints.
- Test problem *P16* after reformulation contains 2 variables and 6 inequality constraints. In the original problem formulation, there were 5 variables and 3 equality constraints.

In [11] the authors stopped the considered algorithms when the point $\mathbf{x}$ was generated such that the percent error $(\tilde{pe})$ Eq. (4.7) is smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-4}$, or when the number of iterations exceeds the prescribed limit of 200. Note that although all considered algorithms belong to the DIRECT-type class, the cost of one iteration can vary significantly. Therefore, tested algorithms were stopped either when $\tilde{pe} \leq \varepsilon_{\text{pe}}$ was satisfied or when the maximal number of function evaluations equal to $200,000$ was reached.

In this thesis the introduced algorithms (DIRECT-GLc and DIRECT-GLce) given on average (Aver.(overall)) significantly better results compared to Filter DIRECT and failed to locate solution point with required tolerance Eq. (4.7) only for $3/20$ of the test problems (highlighted in red color in the colored version), and none of those three problems was solved by the Filter DIRECT algorithm among with three others. However, for simpler test problems, i.e., lower-dimensionality cases ($n \leq 3$) Filter DIRECT is a very promising option. A completely different behavior solving harder test problems, i.e., higher-dimensionality cases ($n \geq 4$) introduced approaches, gives much better results. Other penalty based approaches EPGO and DF-EPGO fail to produce a solution with the required accuracy for eleven and ten test problems accordingly. Finally, the version with an enriched local minimization procedure (DIRECT-GLce-min) failed only on the *P02b* test problem, where the algorithm converged to a local minimum point and gave the best results based on all used comparison criteria.

4.4.3 Group 3: comparison with the eDIRECTc algorithm

Next, a comparison against the recently proposed eDIRECTc algorithm is performed. The authors compared the eDIRECTc algorithm with the CORBA [90], ConstrLMSRBF [89], CiMPS [47], and DIRECT-L1 algorithms. The numerical experiments revealed the potential of the eDIRECTc algorithm for expensive constrained problems in terms of the convergence speed, the quality of final solutions, and the success rate.

To perform as fair as possible a comparison, the same 13 test problems from [52] were used, which also included in the DIRECTlib. Note that several of these test problems: *G03*, *G05*, *G11*, *G13* contain equality constraints and, in the same manner as in [52], allowed constraint violation for equality constraints only $\varepsilon_{\varphi} = 10^{-4}$ was used.

The stopping criterion is the same relative error Eq. (3.7), as was used in the previous analysis. The algorithms were stopped either when the generated point $\mathbf{x}$ was such that the percent error (pe) Eq. (3.7) was smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-2}$. In [52], the maximal allowed the number of function evaluations was allowed to $1,000$. According to the authors, the eDIRECTc algorithm was developed primarily for expensive constrained global optimization problems, in which a simulation of the problem may require several hours or even days. Thus, the eDIRECTc algorithm requires much more running time than the other compared methods, especially this is the case for higher dimensional problems. Thus, the same maximum limit equal to 10^6 function evaluations is used for the algorithms introduced in this chapter. The obtained results are given in Section 4.4.3. Note that the eDIRECTc algorithm is not a deterministic method, and can produce different results in multiple runs. The recorded values of the eDIRECTc algorithm is an average number of ten different runs. For the algorithms introduced in this thesis, there is no such requirement to run the algorithm several times.

First, the DIRECT-GLce algorithm solves $10/13$ of test problems while the eDIRECTc algorithm solves only $8/13$. When the DIRECT-GLce algorithm is combined with the local search procedure in the DIRECT-GLce-min algorithm, the hybridized algorithm outperforms the eDIRECTc version by

Table 4.5: Comparison between different DIRECT-type algorithm solving generally constrained global optimization problems from DIRECTlib

Label	n	f^*	Cons. type			DIRECT-GLc		DIRECT-GLce		DIRECT-GLce-min		eDIRECTc	
			L	NL	EQ	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$	f_{eval}	$f_{\min}$
G01	13	−15.0000	9	0	0	369,163	−14.9990	721,509	−14.9998	8,361	**−15.0000**	**148**	−14.9998
G02	20	−0.8036	1	1	0	$> 10^6$	−0.2375	$> 10^6$	−0.2330	$> 10^6$	−0.2717	$> 1,000$	−0.2480
G03	10	−1.0005	0	0	1	199,539	−1.0004	**194,807**	**−1.0006**	**194,807**	**−1.0006**	145	-0.9989^b
G04	5	−30,665.5387	0	6	0	7,649	−30,663.5710	20,067	−30663.5710	**49**	**−30,665.5390**	65	−30,665.5385
G05	4	5,126.4967	2	0	3	**8,631**	**5,126.6511**	**8,631**	**5,126.6511**	**8,631**	**5,126.6511**	413	$5,145.8149^b$
G06	2	−6,961.8139	0	2	0	3,773	−6,961.1798	6,833	−6,961.6742	241	**−6,961.8138**	**35**	−6,961.8137
G07	10	24.3062	3	5	0	$> 10^6$	24.3237	$> 10^6$	24.3176	2,771	**24.3062**	**152**	**24.3062**
G08	2	−0.0958	0	2	0	443	**−0.0958**	1,007	**−0.0958**	463	**−0.0958**	**154**	**−0.0958**
G09	7	680.6301	0	4	0	38,869	680.6940	74,011	680.6936	**583**	**680.6300**	$> 1,000$	785.6795
G10	8	7,049.2480	3	3	0	$> 10^6$	7,078.9529	$> 10^6$	7,052.0059	57,205	**7,049.2480**	**105**	7,049.2484
G11	2	0.7499	0	0	1	993	0.7499	1,533	**0.7498**	1,533	0.7499	**33**	0.7499
G12	3	−1.000	0	1	0	143	−0.9999	179	−0.9999	**52**	**−1.0000**	179	**−1.0000**
G13	5	0.0539	0	0	3	423,963	0.0539	192,053	0.0539	**59,749**	**0.0538**	$> 1,000$	0.6472
# of failed						3/13		3/13		**1/13**		5/13	

α - result is outside the feasible region

both criteria: the number solved problems $12/13$ and the quality of the final solution. Moreover, the incorporated local minimization procedure into the `DIRECT-GLce-min` algorithm significantly reduces the total number of function evaluations compared to the `DIRECT-GLce`, but the `eDIRECTc` version required the smallest number of function evaluations on the average. On the other hand, the authors in [52] stated that the `eDIRECTc` algorithm requires much more running time compared to other algorithms used in the comparison, therefore the number of function evaluations criterion alone does not represent the real performance of the algorithms very well.

4.4.4 Group 4: applying the developed algorithms on the engineering problems

In this section, the introduced algorithms were tested on five engineering design problems: *tension/compression spring*, *three-bar truss*, *NASA speed reducer*, *pressure vessel*, and *welded beam*. As the global minimums are known for all tested problems, the same stopping rules were used in the previous experiments. No constraint violation was allowed in this experiment, and the parameter ε_φ was set to 0. Note that some of the

Table 4.6: Comparison of results for *tension/compression spring design problem*

x_i, g_i	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1($r=10$)	DIRECT-L1($r=10^2$)	DIRECT-L1($r=10^3$)	eDIRECTc
x_1	0.0519	0.0516	0.0516	0.0543	0.0543	0.0543	0.0517
x_2	0.3630	0.3553	0.3553	0.4228	0.4228	0.4228	0.3567
x_3	10.9421	11.3829	11.3829	8.3127	8.3127	8.3127	11.2882
$g_1(\mathbf{x})$	-1.02×10^{-5}	-1.25×10^{-5}	-1.25×10^{-5}	-0.0042	-0.0042	-0.0042	$0.0012^{\dagger}$
$g_2(\mathbf{x})$	-1.32×10^{-5}	-6.13×10^{-5}	-6.13×10^{-5}	-0.0011	-0.0011	-0.0011	-2.61×10^{-6}
$g_3(\mathbf{x})$	-4.0627	-4.0477	-4.0477	-4.1365	-4.1365	-4.1365	-4.0568
$g_4(\mathbf{x})$	-0.7233	-0.7287	-0.7287	-0.6818	-0.6818	-0.6818	-0.7277
f_{min}	**0.012680**	**0.012680**	**0.012680**	0.012867^{α}	0.012867^{α}	0.012867^{α}	0.012666^{α}
f_{eval}	$423,209$	**178, 115**	$178,221$	10^6	10^6	10^6	292

a – result is outside the feasible region

† – constraint is violated

problems contain integer variables, thus by the same analogy to [52], integers are regarded as continuous ones. A detailed description of these engineering

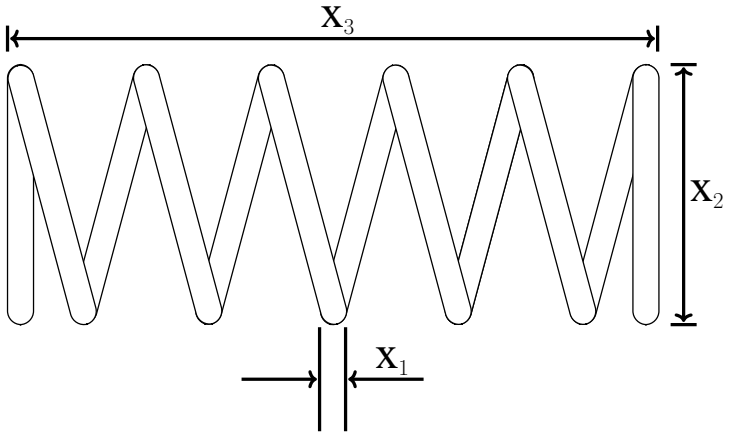

Figure 4.4: Scheme of the *Tension/Compression Spring problem*.

problems and mathematical formulations is given in Appendix A. As we tried to maintain the same number of decimals across the thesis, we acknowledge that some given rounded solution points can slightly violate constraints.

First, the *tension-compression string design problem* is considered. The problem is to minimize the string weight under constraints on deflection, shear stress, surge frequency, limits on the outside diameter. There are three variables as shown in Fig. 4.4. A detailed description of the practical problem can be found in [17, 47], while in Appendix A is given with a short description and mathematical formulation. A comparison of best solutions obtained by the algorithms is shown in Table 4.6. Only three of the algorithms solved the engineering problem, and the `DIRECT-GLce` and `DIRECT-GLce-min` were the most efficient optimizers. Both algorithms solved the problem with the required precision error using the least number of function evaluations. We note that using the `eDIRECTc` algorithm the obtained solution is better compared to ours, but the reported solution point violated constraints of the problem. The reported optimal solution point for the `eDIRECTc` algorithm violates the constrain $g_1(\mathbf{x}) : 1 - \frac{x_2^3 x_3}{71875 x_1^4} \leq 0$. At the solution point the feasible value of the first constraint should be non-positive, but the reported value is $g_1(\mathbf{x}) = 0.0012 > 0$.

Next, the *three-bar truss design problem* is considered. The problem is to minimize the volume subject to stress constraints. There are two variables as shown in Fig. 4.5. A detailed description of the practical problem can be found

Table 4.7: Comparison of results for *three-bar truss design problem*

x_i, g_i	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1($r=10$)	DIRECT-L1($r=10^2$)	DIRECT-L1($r=10^3$)	eDIRECTc
x_1	0.7921	0.7839	0.7886	0.5000	0.6111	0.7839	0.7887
x_2	0.3984	0.4218	0.4082	0.5000	0.5000	0.4218	0.4083
$g_1(\mathbf{x})$	-5.37×10^{-5}	-2.42×10^{-5}	-1.50×10^{-12}	$0.8284^{\dagger}$	$0.3949^{\dagger}$	-2.42×10^{-5}	-1.52×10^{-12}
$g_2(\mathbf{x})$	-1.4752	-1.4487	-1.4641	-0.8284	-1.1222	-1.4487	-1.4641
$g_3(\mathbf{x})$	-0.5247	-0.5512	-0.5358	-0.3431	-0.4828	-0.5512	-0.5359
f_{min}	263.9117	263.9157	**263.8958**	0.012867^{α}	262.3446^{α}	263.9157	**263.8958**
f_{eval}	727	1,055	124	5	11	179	**26**

a – result is outside the feasible region

† – constraint is violated

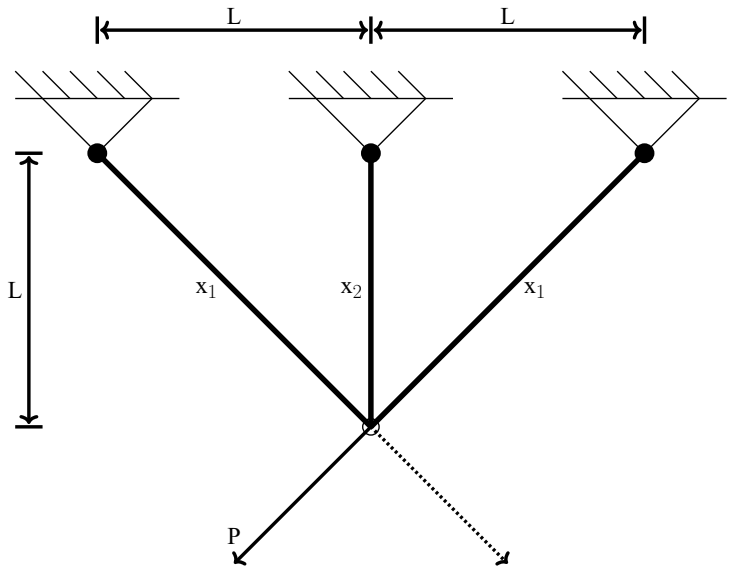

Figure 4.5: Scheme of the *three-bar truss design problem*.

in [88], while in Appendix A a short description and mathematical formulation is given. The comparison of the best solutions obtained by algorithms shown in Table 4.7. The `DIRECT-GLce-min` and `eDIRECTc` were the most efficient optimizers. However, none of the tested algorithms had any difficulty solving the latter problem, and the solution was achieved with all the algorithms.

Next, the *NASA speed reducer design problem* is considered. The problem is to minimize the overall weight subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts, and stresses in the shafts. There are seven variables as shown in Fig. 4.6. A detailed description

Table 4.8: Comparison of results for *NASA speed reducer design problem*

x_i, g_i	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1($r=10$)	DIRECT-L1($r=10^2$)	DIRECT-L1($r=10^3$)	eDIRECTc
x_1	0.7921	3.5000	3.5000	2.7666	2.7666	3.1534	3.5000
x_2	0.3984	0.7000	0.7000	0.7166	0.7166	0.7002	0.7000
x_3	0.7921	17.0002	17.0000	18.8333	18.8333	17.0075	17.0000
x_4	0.7921	7.3000	7.3000	7.8000	7.8000	7.3020	7.3000
x_5	0.7921	7.8000	7.8000	8.0500	8.0500	7.8010	7.7153^{β}
x_6	0.7921	3.3503	3.3502	3.4000	3.4000	3.3506	3.3502
x_7	0.7921	5.2868	5.2866	5.0833	5.0833	5.1800	5.2867
$g_1(\mathbf{x})$	-0.0739	-0.0739	-0.0739	$0.0088^{\dagger}$	$0.0088^{\dagger}$	$0.0267^{\dagger}$	-0.0739
$g_2(\mathbf{x})$	-0.1979	-0.1980	-0.1979	-0.2113	-0.2113	-0.1111	-0.1980
$g_3(\mathbf{x})$	-0.4409	-0.4992	-0.4991	-0.4922	-0.4922	-0.4993	-0.4992
$g_4(\mathbf{x})$	-0.8967	-0.9014	-0.9014	-0.8882	-0.8882	-0.8931	-0.9046
$g_5(\mathbf{x})$	-0.0053	-8.77×10^{-5}	-1.89×10^{-8}	-0.0439	-0.0439	-0.0003	-4.78×10^{-6}
$g_6(\mathbf{x})$	-3.93×10^{-10}	-7.10×10^{-5}	-4.86×10^{-9}	$0.1247^{\dagger}$	$0.1247^{\dagger}$	$0.0630^{\dagger}$	$2.53 \times 10^{-6\dagger}$
$g_7(\mathbf{x})$	-0.7024	-0.7024	-0.7024	-0.6625	-0.6625	-0.7022	-0.7025
$g_8(\mathbf{x})$	-8.4810×10^{-10}	-1.52×10^{-5}	-3.5514×10^{-9}	$0.2951^{\dagger}$	$0.2951^{\dagger}$	$0.1102^{\dagger}$	0.0000
$g_9(\mathbf{x})$	-0.5833	-0.5833	-0.5833	-0.6782	-0.6782	-0.6246	-0.5833
$g_{10}(\mathbf{x})$	-0.0865	-0.0513	-0.0513	-0.1025	-0.1025	-0.0515	-0.0513
$g_{11}(\mathbf{x})$	-0.0262	-0.0108	-0.0108	-0.0693	-0.0693	-0.0260	-6.48×10^{-7}
f_{min}	$3,003.3445$	$2,996.5728$	**$2,996.3482$**	$2,943.8699^{\alpha}$	$2,982.4621^{\alpha}$	$2,995.3825^{\alpha}$	2994.4711^{α}
f_{eval}	10^6	$123,175$	**$10,293$**	67	67	$26,667$	118

α – result is outside the feasible region

β – variable bound constraint violation

† – constraint is violated

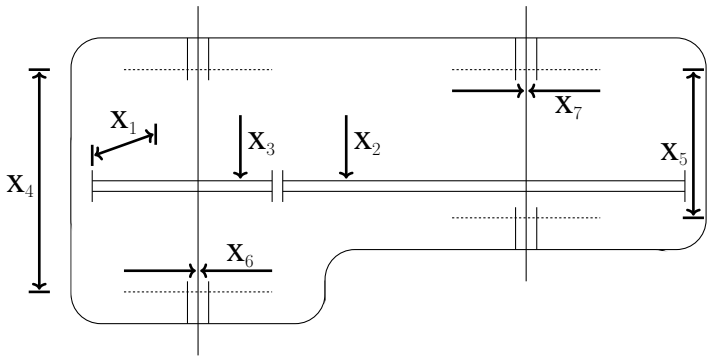

Figure 4.6: Scheme of the *NASA speed reducer design problem*.

of the practical problem can be found in [88], while in Appendix A, a short description and mathematical formulation is given. The comparison of the best solutions obtained by the algorithms shown in Table 4.8. Only two of the algorithms were able to solve the problem: `DIRECT-GLce` and `DIRECT-GLce-min`. The local search procedure reduces the number of function evaluations

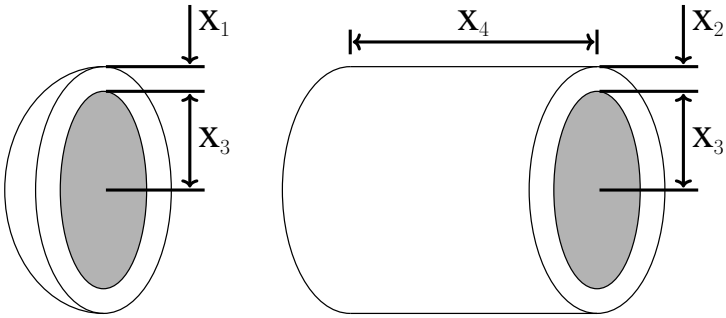

Figure 4.7: Scheme of the *pressure vessel design problem*.

approximately twelve times. DIRECT-L1 and eDIRECTc were able to return a better solution value than the best know value f_{min}, but the solution point violated the constraint functions. The variable bounds for x_5 are $7.8 \leq x_5 \leq 8.3$, however the value of x_5 from the reported optimal solution point for the eDIRECTc algorithm is equal to $x_5 = 7.71532$.

Table 4.9: Comparison of results for *pressure vessel design problem*

x_i, g_i	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1($r = 10$)	DIRECT-L1($r = 10^2$)	DIRECT-L1($r = 10^3$)	eDIRECTc
x_1	1.1457	1.1001	1.1000	1.0625	1.0625	1.0625	1.0000
x_2	0.6250	0.6250	0.6250	0.6458	0.6458	0.6458	0.6250
x_3	59.3638	56.9977	56.9948	59.7222	59.7222	59.7222	51.8135
x_4	37.9086	50.9915	51.0012	36.9444	36.9444	36.9444	84.5786
$g_1(\mathbf{x})$	-3.4162×10^{-11}	-5.12×10^{-5}	-1.93×10^{-10}	$0.0901^{\dagger}$	$0.0901^{\dagger}$	$0.0901^{\dagger}$	-2.89×10^{-14}
$g_2(\mathbf{x})$	-0.0586	-0.0812	-0.0812	-0.0760	-0.0760	-0.0760	-0.1307
$g_3(\mathbf{x})$	-1.3428×10^{-5}	-76.9748	-0.0001	$-10,242.3668$	$-10,242.3668$	$-10,242.3668$	-0.1046
$g_4(\mathbf{x})$	-202.0913	-189.0084	-188.9987	-203.0555	-203.0555	-203.0555	-155.4215
$g_5(\mathbf{x})$	-0.0457	-0.0001	-5.1253×10^{-10}	$0.0375^{\dagger}$	$0.0375^{\dagger}$	$0.0375^{\dagger}$	$0.1000^{\dagger}$
$g_6(\mathbf{x})$	-0.0250	-0.0250	-0.0250	-0.0458	-0.0458	-0.0458	-0.0250
f_{min}	$7,224.7042$	$7,164.4373$	**$7,163.7396$**	$7,025.9405^{\alpha}$	$7,037.4280^{\alpha}$	$7,152.3030^{\alpha}$	$7,006.7816^{\alpha}$
f_{eval}	10^6	$88,585$	**91**	$2,117$	$2,099$	$2,295$	412

α – result is outside the feasible region

† – constraint is violated

Next, the *pressure vessel design problem* is considered. The problem is to minimize the total cost of material, forming, and welding a cylindrical vessel. There are four variables as shown in Fig. 4.7. A detailed description of the practical problem can be found in [47], while in Appendix A, a short description and mathematical formulation is given. The comparison of the best solutions obtained by algorithms shown in Table 4.9. Like in the previous case, only two of the algorithms could solve the problem:

DIRECT-GLce and DIRECT-GLce-min. DIRECT-L1 and eDIRECTc were able to return the better solution value than the best know value f_{min}, but the solution point violated the constraint functions. The variable x_1 is bounded within $1 \leq x_1 \leq 1.375$, but the fifth constraint function $g_5(\mathbf{x}) : 1.1 - x_1 \leq 0$ reduces the feasible interval to $1.1 \leq x_1 \leq 1.375$. However, the value of x_1 for the reported optimal solution point using the eDIRECTc algorithm is $x_1 = 1$.

Table 4.10: Comparison of results for *welded beam design problem*

x_i, g_i	DIRECT-GLc	DIRECT-GLce	DIRECT-GLce-min	DIRECT-L1($r = 10$)	DIRECT-L1($r = 10^2$)	DIRECT-L1($r = 10^3$)
x_1	0.2057	0.2057	0.2057	0.2055	0.2055	0.2055
x_2	3.4701	3.4701	3.4706	3.4592	3.4592	3.4592
x_3	9.0365	9.0365	9.0366	9.0765	9.0765	9.0765
x_4	0.2057	0.2057	0.2057	0.2055	0.2055	0.2055
$g_1(\mathbf{x})$	-0.1830	-0.1830	-0.1838	-0.0615	-0.0615	-0.0615
$g_2(\mathbf{x})$	-2.3221	-2.3221	-0.5293	-236.1105	-236.1105	-236.1105
$g_3(\mathbf{x})$	0.0000	0.0000	-5.9033×10^{-6}	0.0000	0.0000	0.0000
$g_4(\mathbf{x})$	-3.4328	-3.4328	-3.4329	-3.4285	-3.4285	-3.4285
$g_5(\mathbf{x})$	-0.2355	-0.2355	-0.2355	-0.2357	-0.2357	-0.2357
$g_6(\mathbf{x})$	-1.6311	-1.6311	-0.1733	-0.8747	-0.8747	-0.8747
$g_7(\mathbf{x})$	-0.0807	-0.0807	-0.0807	-0.0805	-0.0805	-0.0805
f_{min}	1.7249	1.7249	**1.7248**	1.7284	1.7284	1.7284
f_{eval}	108, 683	104, 191	**163**	10^6	10^6	10^6

Next, the *welded beam design problem* is considered. The problem is to minimize a welded beam for minimum cost, subject to seven constraint functions. There are four variables as shown in Fig. 4.8. A detailed description of the practical problem can be found in [62, 63], while in Appendix A, a short description and mathematical formulation is given. The comparison of the best solutions obtained by the algorithms shown in Table 4.10. Three of the algorithms were able to solve the problem, and once again, the DIRECT-GLce-min algorithm was the most efficient optimizer.

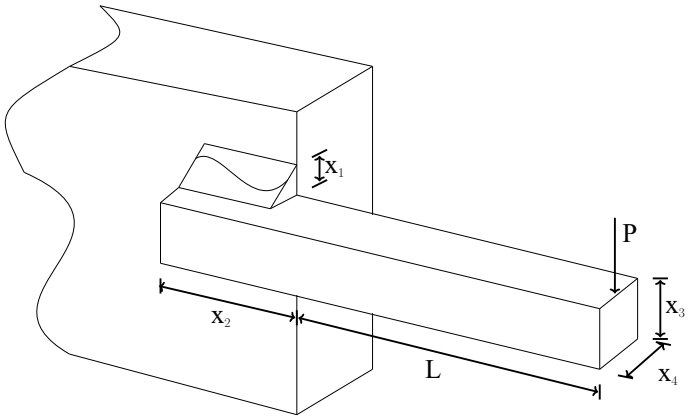

Figure 4.8: Scheme of the *welded beam design problem*.

4.5 Conclusions

This chapter introduces a new strategy for constrained global optimization problems in the DIRECT-type algorithmic framework. Two well-known weaknesses of the DIRECT-L1 algorithms were addressed in the proposed approaches. Instead of the exact L1 penalty approach, an auxiliary function-based approach in the DIRECT-GLc and DIRECT-GLce algorithms is introduced, which does not require any penalty parameters. The proposed DIRECT-GLc and DIRECT-GLce algorithms significantly outperform all the previously tested penalty function-based approaches, and the performance differences increases when the computational budget is more significant.

To improve the solution accuracy and the efficiency of solving high-dimensional problems, DIRECT-GLce has been enriched with a local minimization procedure and called the new algorithm DIRECT-GLce-min. The further experimental investigation revealed the advantage of the DIRECT-GLce and DIRECT-GLce-min algorithms over most test problems and five engineering problems compared with the recent relevant approaches.

One of the most significant challenges of the partition-based DIRECT-type approaches is deals with optimization problems with equality constraints. The proposed DIRECT-GLce showed promising results in solving such problems, but effectiveness strongly depends on the allowed equality constraints violation.

Chapter 5

Modified DIRECT-GL algorithm for global optimization with hidden constraints

The less studied field in DIRECT-type methods is the applicability of an algorithm for problems with hidden constraints, therefore in the fifth chapter of the dissertation, an extension of DIRECT-GL for problems with hidden constraints Eq. (2.8) was investigated. Many decision-making problems arising in diverse areas can be solved as global optimization problems (see, e.g., [4, 23, 28, 29, 32, 68]). Objective and constraints functions describing real-life applications are often non-linear, multi-extremal, non-differentiable, expensive to evaluate, and computed using the black-box type computer software only [91, 96]. Derivative-free optimization [91] is the key technique for finding solutions to such problems for which derivative information is unavailable, unreliable, or impractical to obtain. Such problems often occur in practical black-box optimization [21, 25, 65], therefore more efficient strategies are needed.

5.1 Introduction

To the best of our knowledge, only three DIRECT-type strategies are proposed in the literature [25, 65]. The primary purpose was to design an effective

auxiliary function-based approach for hyper-rectangles, where objective and constraint function information are not available. Two main challenges that arise are; i) detecting at least one feasible point quickly, ii) the convergence of an algorithm to a feasible solution point, especially when it lies on a boundary of the unknown feasible region. The investigated DIRECT-GLh algorithm combines two new techniques for faster discovery of feasible points, and efficient convergence, especially for problems having solutions on the edge of the unknown (hidden) feasible region. The performance of the DIRECT-GLh algorithm has been examined in a detailed numerical study using a broad set of global optimization test problems. The new DIRECT-GLh showed promising performance compared with the other existing DIRECT-type extensions for problems with hidden constraints.

5.2 New auxiliary function-based approach for problems with hidden constraints

In this section, an extension of the previously introduced DIRECT-GL algorithm for global optimization problems with hidden constraints is presented. In the developed DIRECT-GLh, each point of the search domain includes a sequential checking of the feasibility. If the objective function cannot be evaluated at a certain point, the algorithm assigns a new estimated value based on the current best feasible solution. The value is constantly changing when better minima are founded, and the algorithm rapidly adjusts to the optimization problem. The developed strategy is evaluated on the comprehensive test set by comparing it with all existing approaches, subject to function evaluations effectiveness, and algorithmic time. In the following two subsections, main extensions of the DIRECT-GL algorithm are described.

5.2.1 Finding a feasible point

In [11, 52] and the previous chapter it was shown that finding at least one feasible point while solving general optimization problems often can be very costly. For problems with hidden constraints finding a feasible midpoint can be even more challenging, as analytic information about the constraints is

unavailable. To deal with this situation, the uniform partitioning of $\bar{D}$ is performed, as stated in Definition 1.

Definition 1 *Consider $\bar{D}$, set $\mathbb{I}^k = \{1\}$, where $k = 1$, and perform the following steps:*

- ***Step 1*** *Find an index $j \in \mathbb{I}^k$ and a corresponding hyper-rectangle $\bar{D}_j$, such that*
$$\bar{D}_j = \arg\max_j \left\{ j = \arg\max_{i \in \mathbb{I}^k} \{\delta_i\} \right\}. \tag{5.1}$$
- ***Step 2*** *Subdivide (trisect) a hyper-rectangle $\bar{D}_j$ and check the feasibility at midpoints of all new hyper-rectangles. Also, set $k = k + 1$, update $\mathbb{I}^k$, and all measures δ_{j_i} of new hyper-rectangles.*
- ***Step 3*** *If $D^{\text{feas}} = \emptyset$* ***repeat*** *from Step 1; otherwise* ***terminate****.*

At Step 1, a hyper-rectangle with the largest size is chosen. If there are several hyper-rectangles with the equal maximal size, the preference is given to the hyper-rectangle with the largest index value. At Step 2, the selected hyper-rectangle is subdivided using the trisection technique, and the feasibility at the all new center points is checked. The process is continued until at least one feasible midpoint is found, i.e., $D^{\text{feas}} \neq \emptyset$. After this, all the effort is put focused on the improvement of the feasible solution.

5.2.2 Improving a feasible solution

By the second extension to `DIRECT-GL`, problem (2.8) is transformed to (5.2) as follows:

$$\begin{aligned} &\min_{\mathbf{x} \in D} \phi(\mathbf{x}, \mathbf{x}_{\min}, f_{\min}), \\ &\phi(\mathbf{x}, \mathbf{x}_{\min}, f_{\min}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \in D^{\text{feas}} \\ f_{\min} + d(\mathbf{x}_{\min}, \mathbf{x}), & \text{otherwise,} \end{cases} \end{aligned} \tag{5.2}$$

where $f_{\min}$ is the best feasible objective function value found so far, and $d(\mathbf{x}_{\min}, \mathbf{x})$ is Euclidean distance Eq. (3.2) from the current best minimum point $(\mathbf{x}_{\min})$ to the point $(\mathbf{x})$. When $\mathbf{x} \in D^{\text{feas}}$, the objective function is evaluated at $f(\mathbf{x})$, otherwise a value $f_{\min} + d(\mathbf{x}_{\min}, \mathbf{x})$ is assigned to an infeasible midpoint of the hyper-rectangle. A geometrical illustration of this

procedure is shown in Fig. 5.1. Using this strategy, the assigned values for hyper-rectangles with infeasible midpoints depends on the currently found minimum value of f_{min}, and the (Euclidean) distance from the current minimum point. This way, the convergence of `DIRECT-GLh` to a feasible solution point is assured, even when complicated shape constraints are faced. Note that this comes with a slight performance overhead. Each time when a better value of f_{min} is found, for all hyper-rectangles with infeasible midpoints the value $f_{\text{min}} + d(\mathbf{x}_{\text{min}}, \mathbf{x})$ is updated. On the other hand, the distances $d(\mathbf{x}_{\text{min}}, \mathbf{x})$ are used in the selection procedure of potential optimal hyper-rectangles in the original `DIRECT-GL` algorithm, therefore the performance overhead is minimal.

In most of the `DIRECT`-type algorithms domain D is normalized $\bar{D}$ in interval $[0, 1]$ and `DIRECT-GL` uses normalized coordinates $\mathbf{x}$ in Eq. (2.5). While in proposed `DIRECT-GL` potential optimal hyper-rectangles selection scheme, dimensionality does not attach any importance but in (5.2) formulation Euclidean distance has dependence on dimensionality n, and it is unclear how it is going to scale with objective function. The biggest issue with the formulation (5.2) is that values of objective function and distances can differ significantly. In order to treat them equally, the values are normalize as:

$$\begin{aligned} \bar{f}(\mathbf{x}) &= \begin{cases} 1, & \text{if } f_{\text{max}} == f_{\text{min}} \\ \dfrac{f(\mathbf{x}) - f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}}, & \text{otherwise.} \end{cases} \\ \bar{d}(\mathbf{x}_{\text{min}}, \mathbf{x}) &= \frac{d(\mathbf{x}_{\text{min}}, \mathbf{x}) - d_{\text{min}}}{d_{\text{max}} - d_{\text{min}}}, \end{aligned} \tag{5.3}$$

where d_{max} is the length of the diagonal of $\bar{D}$ and d_{min} is equal to 0. Again, this adds some performance overhead, as normalized objective function and distance values, should be updated each time when the hyper-rectangle with the new largest/lowest objective value at the center point is found. By taking into account (5.3) and (5.2), a new problem is derived:

$$\begin{aligned} &\min_{\mathbf{x} \in D} \bar{\phi}(\mathbf{x}, \mathbf{x}_{\text{min}}, \bar{f}_{\text{min}}), \\ &\bar{\phi}(\mathbf{x}, \mathbf{x}_{\text{min}}, \bar{f}_{\text{min}}) = \begin{cases} \bar{f}(\mathbf{x}), & \text{if } \mathbf{x} \in D^{\text{feas}}, \\ \bar{f}_{\text{min}} + \bar{d}(\mathbf{x}_{\text{min}}, \mathbf{x}), & \text{otherwise.} \end{cases} \end{aligned} \tag{5.4}$$

Note, that $\bar{f}_{\min}$ is always equal to 0, and could be omitted from (5.4).

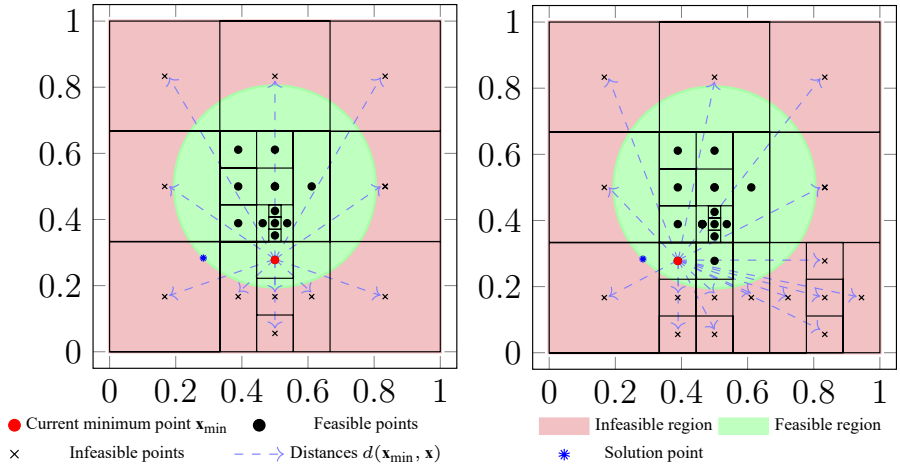

Figure 5.1: Geometric illustration of the DIRECT-GLh algorithm solving two-dimensional *T1* test problem in the third (left side), and in the fourth (right side) iterations.

5.3 Algorithmic steps

The detailed description of the DIRECT-GLh algorithm is given in Algorithm 4. The inputs are the problem ($\mathcal{P}$) and stopping conditions: required tolerance (ε_{pe}), the maximal number of function evaluations ($\text{FE}_{\max}$), and the maximal number of DIRECT-GLh iterations ($\text{K}_{\max}$). After termination, DIRECT-GLh returns the best value of the objective function $f_{\min}$ and the solution point $\mathbf{x}_{\min}$ together with algorithmic performance measures: final tolerance – percent error (pe), the number of function evaluations (fe), and the number of iterations (k).

The DIRECT-GLh algorithm starts from the single center point. If it is infeasible, the algorithm performs uniform sampling of hyper-rectangles using Definition 1 until a feasible point is found (see Lines 2–7 in Algorithm 4). DIRECT-GLh uses the two-step based strategy for the selection of potential optimal hyper-rectangles (sets $\mathbb{S}_1^k$ and $\mathbb{S}_2^k$), as was introduced in second chapter Chapter 3. During steps described in Lines 13–17, DIRECT-GLh subdivides all hyper-rectangles from the set $\mathbb{S}_3^k$, calculates

Algorithm 4: Pseudo code of the `DIRECT-GLh` algorithm

input : $\mathcal{P}$, ε_{pe}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$;
output: $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, k, fe;

Initialize $k = 1$, $fe = 1$, $\mathbb{I}^k = \{1\}$;
if $\mathbf{x}_1 \neq D^{\text{feas}}$ **then**
    Find a feasible center point; `// using Definition 1`
    Update $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, fe, $\mathbb{I}^k$;
else
    Set $f_{\min} = f(\mathbf{x}_1)$, $\mathbf{x}^k_{\min} = \mathbf{c}_1$ and update pe;
end
Calculate $d(\mathbf{x}^k_{\min}, \mathbf{c}_i)$, $\phi(\mathbf{c}, \mathbf{x}^k_{\min}, f_{\min})$ and $\bar{\phi}(\mathbf{c}, \mathbf{x}^k_{\min}, \bar{f}_{\min})$, $\mathbf{x}_i \in \bar{D}^k_i$ for all $i = 1, ..., fe$; `//` d, ϕ `and` $\bar{\phi}$ `defined in Eqs. (3.2), (5.2) and (5.4)`
while $pe > \varepsilon_{\text{pe}}$ **and** $fe < \mathsf{FE}_{\max}$ **and** $k < \mathsf{K}_{\max}$ **do** `//` pe (3.7)
    Identify the "global" (index) set $\mathbb{S}^k_1 \subseteq \mathbb{I}^k$ of POH on $\bar{\phi}(\mathbf{c}, \mathbf{x}_{\min}, \bar{f}_{\min})$ using Definition 2 ;
    Identify the "local" (index) set $\mathbb{S}^k_2 \subseteq \mathbb{I}^k$ of POH on d (3.2) using Definition 3 ;
    Find unique union of potential optimal hyper-rectangles sets $\mathbb{S}^k_3 = \mathbb{S}^k_1 \bigcup \mathbb{S}^k_2$;
    foreach $i \in \mathbb{S}^k_3$ **do**
        Subdivide (trisect) hyper-rectangle $\bar{D}^k_i$ and update $\mathbb{I}^k$;
        Calculate $\phi(\mathbf{c}, \mathbf{x}^k_{\min}, f_{\min})$ at new center points ;
        Update $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, and fe ;
    end
    if $d(\mathbf{x}^k_{\min}, \mathbf{x}^{k-1}_{\min}) \geq 10^{-6}$ **then**
        Calculate $d(\mathbf{x}^k_{\min}, \mathbf{c}_i)$, $\phi(\mathbf{c}, \mathbf{x}^k_{\min}, f_{\min})$ and $\bar{\phi}(\mathbf{c}, \mathbf{x}^k_{\min}, \bar{f}_{\min})$, $\mathbf{c}_i \in \bar{D}^k_i$ for all $i = 1, ..., fe$;
    else
        Calculate $\bar{\phi}(\mathbf{c}, \mathbf{x}^k_{\min}, \bar{f}_{\min})$, $\mathbf{c}_i \in \bar{\mathbb{D}}^k_i$ for all $i = fe_{\text{old}}, ..., fe_{\text{new}}$;
    end
    Increase $k = k + 1$;
end
return $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, k, fe;

Euclidean distances $d(\mathbf{x}^k_{\min}, \mathbf{x})$ (using (3.2)), and $\phi(\mathbf{x}, \mathbf{x}^k_{\min}, f_{\min})$ (using (5.2)). Then, if the new best objective value is found, the algorithm performs steps described in Line 19, otherwise steps in Line 21. `DIRECT-GLh` repeats all these steps until any stopping condition is unsatisfied.

5.4 Numerical investigation

Tested algorithms

In this section, the efficiency of the developed `DIRECT-GLh` algorithm is investigated and compared with the three existing `DIRECT`-type extensions for global optimization problems with hidden constraints (`DIRECT-Barrier`, `DIRECT-NAS` and `DIRECT-sub`).

Benchmark problems

Test problems from the most recent version of the library `DIRECTlib` [110] (67 in total) are used to evaluate the performance of selected algorithms. Description of all test problems used in this section in a `MatLab` format is given in the online resource [110] and Table 4.2. It is assumed that any information about the constraint functions is unavailable. In the experimental setup, the hidden searching area (D^{hidden}) is implemented as

$$D^{\text{hidden}} = \{\mathbf{x} \in D : \mathbf{g}(\mathbf{x}) \leq 0, \mathbf{h}(\mathbf{x}) = 0\}, \tag{5.5}$$

but this information is used only to determine whether a certain point is feasible or not.

Stopping criteria

Investigated algorithms were stopped either when the generated point $\mathbf{x}$ was such that the percent error (pe) Eq. (3.7) is smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-2}$, or when the number of function evaluations exceeds the prescribed limit of 10^6. Additionally, the maximal time for solving one test problem was restricted to six hours. When an algorithm exceeded this time limit, the final result was set to 10^6. All the computations were carried out on

a six-core computer with the 8th Generation Intel R Core™ i7-8750H @ 2.20GHz Processor, 16 GB of RAM and MatLab R2020a software environment.

Experimental investigation of DIRECT-GLh

The authors of [65] did not provide public access to their algorithm implementation. Moreover, detailed information on how often an extra sub-dividing step should be performed during execution is omitted. Therefore, in our own implementation of DIRECT-sub, the frequency on how often the sub-dividing step should be performed is controlled by the two parameters γ and N. In the experimental analysis, two different strategies were used, when an extra subdividing step was executed on iterations $k = \gamma^N$, where γ is equal either to 2 or 5, and $N = 1, 2, \ldots$. In practice, this means, that every two (or five) iterations and additional sub-dividing step was performed.

Table 5.1: The total number of function evaluations needed by the algorithms to find the first feasible point ($\mathbf{x} \in D^{\text{feas}}$) on a selected subset of test problems from the DIRECTlib

Label	n	Constr. type	DIRECT-GLh	DIRECT-NAS	DIRECT-Barrier	DIRECT-sub ($\gamma = 2$)	DIRECT-sub ($\gamma = 5$)
Bunnag 6	10	L	**136, 549**	363, 929	1, 240, 029	5, 614, 449	1, 951, 629
G06	2	NL	**33, 211**	39, 063	59, 049	202, 589	136, 125
G16	5	NL	**803**	2, 513	2, 673	39, 253	25, 203
Genocop 9	4	L	8, 939	**7, 543**	10, 935	42, 261	53, 695
Genocop 11	4	L	**2, 889**	7, 917	9, 477	73, 881	36, 549
hs021mod	7	L	**33, 051**	33, 309	85, 293	127, 569	54, 435
P2d	5	NL	139	137	243	**81**	231
P9	3	L	81	**51**	189	225	99
s232	2	L	**817**	2, 817	3, 645	10, 449	3, 681
Average (overall)			**24, 053**	50, 809	156, 837	678, 973	251, 294

The obtained experimental results are summarized in Tables 5.1 and 5.2. Here, the best results are given in bold. In both tables, in the first column (Label), the name of the problem is listed, while in the second – dimensionality of the problem (n). In the third column (Constr. type), the

type of hidden constraints is specified: linear (L), or non-linear (NL). Next, in the consecutive columns, the total number of function evaluations (f_{eval}) and also the total execution time (in seconds) in Table 5.2 are given.

First, the situations when finding at least one feasible point is costly are considered (see Section 5.2.1 for more information on this). For this, a subset of test problems (from `DIRECTlib` [110]) having a tiny feasible region (D^{feas}) were selected. Using the proposed uniform sampling phase in the `DIRECT-GLh`, the algorithm can locate the feasible point in six out of the nine tested problems faster (see Table 5.1). Also, the average number of function evaluations using the `DIRECT-GLh` algorithm is more than two times smaller compared with the second-best algorithm `DIRECT-NAS`.

Next, solving the complete set of 67 test instances, `DIRECT-GLh` fails to solve only $(5/67)$ test problems, while the second-best `DIRECT-NAS` fails to solve $(15/67)$ accordingly. Naturally, the low-dimensional test problems are the simplest for all tested approaches, but still, `DIRECT-GLh` requires approximately 1.3 times fewer function evaluations compared to `DIRECT-NAS`. However, the efficiency of the `DIRECT-NAS` approach decreases significantly when solving more complicated problems, i.e., larger dimensionality, and when D^{hidden} is of non-linear nature. Then, `DIRECT-GLh` algorithm requires approximately 2.4 times less function evaluations compared to the second-best `DIRECT-NAS`. The greatest advantage of the `DIRECT-GLh` algorithm is speed; the overall average time of solving all tested problems, `DIRECT-GLh` operates approximately 97.7 times faster, compared with the second best algorithm `DIRECT-NAS`. Moreover, 62/67 of the test problems were solved faster using the developed `DIRECT-GLh` algorithm.

Looking at other three tested algorithms, inconsiderable difference between the `DIRECT-sub` algorithm, which combines a barrier strategy with an extra sub-dividing step, and the original `DIRECT-Barrier` is noticed . Among them, the best performance is achieved with `DIRECT-sub` and sub-dividing steps are activated at iterations $k = 5^N, N = 1, 2, \ldots$. The sub-dividing step decomposes the edge of hidden constraints more efficiently and increases the searching region. Still, solving the higher-dimension problems, this step becomes less efficient. In later iterations, the execution of one sub-dividing step can take a considerable amount of the entire available

budget of function evaluations.

5.5 Conclusions

In recent years DIRECT has become a useful optimization tool for solving problems with bound constraints, but algorithms can now successfully handle even more complicated general constraints. This section, introduced an extended DIRECT algorithm version for problems with hidden constraints, where any information of constraints is not given, and only the evaluation of objective function can determine if the point is feasible. The numerical investigation of introduced DIRECT-GLh shows better performance results than all the existing DIRECT-type algorithms for such optimization problems.

Table 5.2: Number of function evaluations and time (in sec.) solving problems from `DIRECTlib`

Label	n	Constr. type	`DIRECT-GLh`		`DIRECT-NAS`		`DIRECT-Barrier`		`DIRECT-sub` ($\gamma = 2$)		`DIRECT-sub` ($\gamma = 5$)	
			Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}
Bunnag 1	4	L	**2.86**	**33,709**	5,653.63	288,823	23.74	49,261	23.74	199,851	174.88	177,009
Bunnag 2	4	L	**0.37**	5,215	11.28	**4,651**	159.48	$> 10^6$	10.55	85,999	152.02	$> 10^6$
Bunnag 3	5	L	**0.55**	7,933	350.42	25,105	0.86	**3,325**	85.97	$> 10^6$	1.33	5,835
Bunnag 4	6	L	**1.26**	21,039	21,600.00	$> 10^6$	3.02	**16,089**	114.67	$> 10^6$	3.92	19,657
Bunnag 5	6	L	**1.57**	**24,659**	21,600.00	$> 10^6$	68.94	$> 10^6$	190.37	$> 10^6$	63.20	$> 10^6$
Bunnag 6	10	L	278.02	$> 10^6$	21,600.00	$> 10^6$	56.93	$> 10^6$	205.56	$> 10^6$	106.77	$> 10^6$
Bunnag 7	10	L	**204.88**	**288,021**	21,600.00	$> 10^6$	53.43	$> 10^6$	255.66	$> 10^6$	102.28	$> 10^6$
circle	3	NL	**0.19**	**2,043**	14.26	12,771	21,600.00	$> 10^6$	21,600.00	$> 10^6$	21,600.00	$> 10^6$
G02	20	NL	113.02	$> 10^6$	21,600.00	$> 10^6$	111.15	$> 10^6$	134.21	$> 10^6$	104.59	$> 10^6$
G04	5	NL	**0.99**	**11,915**	2,710.45	97,109	439.11	$> 10^6$	48.83	$> 10^6$	427.15	$> 10^6$
G06	2	NL	**20.68**	**35,953**	2,813.75	61,049	506.05	$> 10^6$	182.39	$> 10^6$	354.84	$> 10^6$
G08	2	NL	**9.75**	20,117	23.05	**13,203**	416.18	$> 10^6$	180.16	$> 10^6$	313.61	$> 10^6$
G09	7	NL	**6.87**	95,463	23.05	**13,203**	91.91	$> 10^6$	36.12	$> 10^6$	286.21	$> 10^6$
G12	3	NL	**0.06**	175	0.37	483	0.12	**91**	868.14	$> 10^6$	1.22	223
G16	5	NL	**21.97**	**203,545**	21,600.00	$> 10^6$	138.62	$> 10^6$	110.98	$> 10^6$	189.54	$> 10^6$
G19	15	NL	83.04	$> 10^6$	21,600.00	$> 10^6$	112.05	$> 10^6$	106.73	$> 10^6$	125.48	$> 10^6$
G24	2	NL	**0.09**	**665**	0.61	1,097	76.82	$> 10^6$	136.00	$> 10^6$	93.98	$> 10^6$
Genocop 9	4	L	**2.49**	10,953	9.69	**9,659**	120.70	$> 10^6$	54.26	$> 10^6$	398.81	$> 10^6$
Genocop 10	4	L	**0.42**	**6,151**	26.49	8,589	616.43	$> 10^6$	196.61	$> 10^6$	22.02	91,377

Continued on next page

Table 5.2 Continued from previous page

Label	n	Constr. type	DIRECT-GLh		DIRECT-NAS		DIRECT-Barrier		DIRECT-sub ($\gamma = 2$)		DIRECT-sub ($\gamma = 5$)	
			Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}
Genocop 11	4	L	**2.47**	**32,401**	21,600.00	$> 10^6$	413.29	$> 10^6$	195.33	$> 10^6$	66.42	$> 10^6$
Gold. and Pr.	2	NL	**0.14**	**959**	0.61	1,491	8.41	25,713	1.08	6,581	7.45	24,659
Gomez	2	NL	**0.08**	**567**	0.38	825	8.41	25,713	1.07	10,525	75.15	205,763
Himmelblau's	5	NL	**1.72**	**24,087**	3,782.69	119,167	704.02	$> 10^6$	91.43	$> 10^6$	202.58	$> 10^6$
Horst 1	2	L	**0.17**	1,751	1.07	**1,295**	238.50	$> 10^6$	523.14	$> 10^6$	1,524.13	$> 10^6$
Horst 2	2	L	**0.07**	**547**	0.65	663	1,600.48	$> 10^6$	1.39	9,113	783.74	$> 10^6$
Horst 3	2	L	**0.10**	721	0.20	**237**	0.16	283	86.81	$> 10^6$	0.23	283
Horst 4	3	L	**0.21**	**2,293**	2.71	3,083	1,071.67	$> 10^6$	66.38	$> 10^6$	390.83	$> 10^6$
Horst 5	3	L	**0.22**	**2,043**	1.84	2,241	95.65	$> 10^6$	93.88	$> 10^6$	322.98	$> 10^6$
Horst 6	3	L	**0.57**	**6,251**	14.14	6,591	363.11	$> 10^6$	123.90	$> 10^6$	508.47	$> 10^6$
Horst 7	3	L	**0.18**	2,225	0.77	1,443	0.27	**679**	0.53	3,229	0.48	1,543
hs021	2	L	**0.02**	127	0.07	**107**	0.45	1,277	0.08	361	0.26	831
hs021mod	7	L	81.66	$> 10^6$	21,600.00	$> 10^6$	562.06	$> 10^6$	38.68	$> 10^6$	47.07	$> 10^6$
hs024	2	L	**0.10**	667	0.59	**647**	167.62	$> 10^6$	63.08	$> 10^6$	649.30	$> 10^6$
hs035	3	L	**0.21**	2,579	6.05	5,435	0.62	**1,791**	16.17	200,707	0.74	3,145
hs036	3	L	**0.18**	1,873	0.34	**727**	0.25	**727**	0.23	**727**	0.22	**727**
hs037	3	L	**0.31**	**3,567**	12.25	7485	9.39	36,567	151.62	$> 10^6$	8.29	36,429
hs038	4	L	**0.81**	10,967	1,081.22	44,663	1.62	**6,433**	206.66	$> 10^6$	5.39	23,805
hs044	4	L	86.57	$> 10^6$	**49.85**	**12,223**	1,041.87	$> 10^6$	145.86	$> 10^6$	621.12	$> 10^6$
hs076	4	L	**0.07**	**7,885**	1,073.36	40,557	166.56	511,919	46.56	$> 10^6$	106.78	400,141
P2d	5	NL	**3.21**	**41,807**	8,023.76	118,151	137.45	$> 10^6$	89.89	$> 10^6$	83.94	$> 10^6$

Continued on next page

Table 5.2 Continued from previous page

Label	n	Constr. type	DIRECT-GLh		DIRECT-NAS		DIRECT-Barrier		DIRECT-sub ($\gamma = 2$)		DIRECT-sub ($\gamma = 5$)	
			Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}
P3b	2	NL	**0.06**	499	0.49	595	0.15	**461**	0.32	2,151	0.36	995
P4	2	NL	**0.05**	391	0.14	227	0.12	**233**	0.24	1,759	0.11	245
P6	2	NL	**0.12**	**1,015**	1.21	1,381	122.31	223,859	6.82	66,721	103.12	226,457
P7	2	NL	**0.11**	907	0.88	2,147	0.19	**427**	0.22	1339	0.24	783
P8	2	NL	**0.13**	**1,001**	1.11	1,555	3.97	11,059	0.94	8,085	0.47	1,115
P9	3	L	**0.21**	1,867	1.47	**1,807**	229.65	$> 10^6$	54.59	$> 10^6$	550.38	$> 10^6$
P10	2	NL	**0.07**	**539**	0.52	1,151	0.22	573	73.12	$> 10^6$	0.08	573
P11	2	NL	**0.09**	**769**	1.48	2,143	0.41	5,959	3.57	34,751	1.00	3,631
P12	1	NL	0.10	27	0.09	**23**	0.16	**23**	**0.06**	**23**	0.08	29
P14	3	L	**0.24**	1,573	1.18	**1,519**	2.16	9,125	3.61	35,589	1.40	5,065
P16	2	NL	**0.03**	89	0.06	67	0.08	**63**	0.06	205	0.07	71
s224	2	L	**0.07**	**387**	0.35	479	0.97	2,131	0.31	1,559	0.87	2,063
s231	2	L	**0.11**	**551**	0.72	1,279	0.46	1,139	244.64	$> 10^6$	0.33	1,021
s232	2	L	**0.34**	**1,749**	53.54	7,045	766.84	$> 10^6$	132.66	$> 10^6$	295.12	$> 10^6$
s250	3	L	**0.44**	**3,425**	4.21	4,077	542.93	$> 10^6$	53.50	$> 10^6$	237.80	$> 10^6$
s251	3	L	**0.32**	**3,567**	10.82	7,485	7.51	36,567	37.66	$> 10^6$	9.11	36,429
T1	2	NL	**0.07**	**509**	0.97	2,419	0.28	859	0.06	801	81.83	$> 10^6$
T1	3	NL	**0.39**	4,043	0.68	**3,597**	30.76	19,343	2.44	56,723	2.03	12,551
T1	4	NL	**1.82**	**23,721**	21,600.00	$> 10^6$	58.95	140,257	244.69	$> 10^6$	2,834.21	$> 10^6$
T1	5	NL	**7.63**	**101,733**	21,600.00	$> 10^6$	474.17	$> 10^6$	132.66	$> 10^6$	691.04	$> 10^6$
T1	6	NL	**6.05**	**85,929**	21,600.00	$> 10^6$	140.53	$> 10^6$	53.51	$> 10^6$	831.48	$> 10^6$

Continued on next page

Table 5.2 Continued from previous page

Label	n	Constr. type	DIRECT-GLh		DIRECT-NAS		DIRECT-Barrier		DIRECT-sub ($\gamma = 2$)		DIRECT-sub ($\gamma = 5$)	
			Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}	Time (s)	f_{eval}
TI	7	NL	**18.81**	**245, 141**	21, 600.00	$> 10^6$	147.26	$> 10^6$	37.66	$> 10^6$	197.94	$> 10^6$
TI	8	NL	**19.54**	**254, 227**	21, 600.00	$> 10^6$	998.99	$> 10^6$	70.41	$> 10^6$	122.17	$> 10^6$
zecevic2	2	L	0.16	1, 247	2.66	2, 283	0.61	1, 403	**0.14**	2, 637	0.52	**1, 127**
zecevic3	2	NL	0.13	1, 067	0.59	799	0.14	323	**0.12**	2, 427	0.18	**281**
zecevic4	2	NL	**0.14**	1, 371	0.55	**883**	502.36	$> 10^6$	1.81	34, 399	2, 963.76	$> 10^6$
zy2	3	NL	**1.43**	16, 381	1.89	**5, 099**	119.13	$> 10^6$	53.18	$> 10^6$	234.57	$> 10^6$
# of failed			**5**		15		35		43		36	
Average (overall)			**14.74**	**99, 472**	5, 220.33	237, 875	602.64	550, 537	413.34	653, 228	581.86	562, 997
Average (NL cons.)			**9.66**	**96, 262**	6, 417.90	286, 286	969.32	551, 735	735.42	582, 621	967.62	590, 223
Average (L cons.)			**19.68**	**102, 587**	4, 057.97	190, 888	246.74	549, 374	100.73	721, 758	207.45	523, 720
Average ($n \geq 4$)			**31.67**	**218, 047**	11, 559.16	525, 766	252.57	757, 882	99.59	845, 201	379.33	757, 432
Average ($n \leq 3$)			**1.02**	**3, 330**	81.73	4, 450	886.48	382, 419	667.73	497, 573	746.07	393, 538
Median			**0.32**	**3, 425**	9.69	5, 099	91.98	$> 10^6$	54.27	$> 10^6$	83.95	$> 10^6$

Concluded

Chapter 6

Accelerating the DIRECT-GLce algorithm through dynamic data structures and parallelization

First, it should be stressed that the iterative nature and a small number of selected potential optimal hype-rectangles to feed the computational units, even for large dimensional problems, do not allow efficient parallelism of the DIRECT-type algorithms using hundreds of processors. Due to these limitations, this chapter focuses on the acceleration of DIRECT-type algorithms through dynamic data structures and parallelization. The choice of the MatLab software is based mainly on the fact, that most widely used implementations of the original DIRECT [18, 25] algorithm, as well as various later introduced DIRECT-type extensions, were developed within MatLab. This allowed us to carry out the most detailed and fair comparison under the same conditions, i.e., by running competing algorithms (also implemented in MatLab) under the same environment. Moreover, the MatLab software is one of the most widely used mathematical computing environments in scientific and technical computing [9]. Also, we want to stress, that while developed parallel algorithms were tested within the MatLab, all proposed acceleration techniques could be implemented in any programming language. Let us note,

that the developed parallel algorithms (presented in this chapter) showed very promising parallel performance even within MatLab environment. Nevertheless, using more appropriate programming languages for the parallelization, like C++ or Fortran, even better parallel efficiency could be obtained. But due to the limitations of DIRECT-type for an efficient parallelization on numerous processors, and seeking a broader applicability of the developed algorithms, we believe, that the MatLab software is also a good trade-off in this situation.

Most of the existing DIRECT-type implementations use a static data structure to store the search space partitioning's information. This can result in failure of the code if the array is insufficient to hold the necessary information. Some implementations reallocate the array to be significantly larger than needed, but it can cost a considerable amount of overhead in execution time and computer memory usage [38]. New versions with dynamic data structures have been proposed to overcome this problem.

Lipschitz-type global optimization methods are computationally intensive [41, 42, 70, 77, 98, 102] and therefore a natural way to speed up is to parallelize them [32, 75, 80, 92, 93, 99, 111, 112, 113, 114]. Nevertheless, just a few parallel implementations of DIRECT-type algorithms are known, and they are developed mainly by the same group of researchers [25, 34, 35, 36, 37, 81, 122]. The previous schemes improve workload distribution, but they have some drawbacks:

- The total number of function evaluations significantly increases in comparison with the sequential algorithm, and a large amount of these evaluations are performed at "*redundant points*" [32, 93, 114];
- The authors used their serial version for comparison, instead of identifying the best existing sequential DIRECT-type algorithm to evaluate the parallel efficiency;
- To the best of our knowledge, all the existing parallel DIRECT-type versions are focused on box-constrained global optimization problems.

6.1 Introduction

In this chapter, two different speedup techniques for a deterministic DIRECT-type global optimization algorithm DIRECT-GLce are considered. The dynamic data structures [38] are included in the sequential MatLab DIRECT-GLce algorithm, resulting in one of the most effective DIRECT-type methods. This chapter studies the shared and distributed parallel implementations of the DIRECT-GLce algorithm and a distributed parallel version for the Aggressive DIRECT counterpart. The first three parallel DIRECT-type algorithms pD-GLce-parfor, pD-GLce-spmd, and pD-ACe-spmd for generally constrained global optimization problems are introduced, which are *non-redundant* [32, 93, 114], i.e., the number of sampled points is the same in the sequential and the parallel version. The load balancing used in the developed MatLab master-worker parallel schemes can be easily adapted to be used by other sequential MatLab DIRECT-type algorithms like [6, 20, 25, 52, 53, 54, 77].

The efficiency of DIRECT-type parallel versions is evaluated solving box and generally constrained global optimizations problems with varying complexity. Furthermore, to evaluate the parallel efficiencies, the best found sequential DIRECT-type approaches implemented in MatLab were used. Numerical results showed a great efficiency, especially for the distributed parallel version of the DIRECT-GLce on a multi-core PC.

6.2 Implementation of parallel DIRECT-GLce approaches

To overcome the difficulties described in the previous sections (see Section 2.4 for more information on this), three parallel implementations of the DIRECT-GLce algorithm in the MatLab software environment [61] are presented. As various applications can benefit from parallel computing, many parallel MatLab extensions have been created [9]. Until the presence of official MathWorks extensions to the MatLab language, according to [105], the most notable extensions were pMATLAB [119], MatlabMPI [49],

MultiMATLAB [120], and bcMPI [43]. This chapter presents the functionality available from the MathWorks official extension to the MatLab language – the focus is on the Parallel Computing Toolbox [61]. The Parallel Computing Toolbox addresses the main challenge of making code to work efficiently in multi-core systems by providing several parallel programming paradigms [60], like threads, parallel for-loops, and SPMD (Single Program Multiple Data) constructs, and so on.

To avoid ambiguity, the same terminology as in the official MatLab documentation [61], and in [105] is used. The term *workers* will be used as a generic term for the MatLab computational engine processes that function completely independently of each other and execute assigned tasks in parallel. A MatLab client is responsible for the distribution of these tasks through the functions in the Parallel Computing Toolbox. However, when the communication infrastructure is required for using the message-passing functions, these connected workers will be called "*labs*" [105]. Like threads, labs are executed on processor cores, but unlike threads, they do not share a memory, therefore they can run also on separate computers connected via a network.

In the following sections, two parallel versions of the sequential DIRECT-GLce algorithm are presented implemented using parallel for-loops and the message-passing paradigm, and the parallel implementation of the pD-ACe-spmd variation, implemented using only the message-passing paradigm. Note that similar parallel schemes may be easily adapted for other existing sequential DIRECT-type implementations.

6.2.1 Parallel pD-GLce-parfor algorithm

The first parallel version of DIRECT-GLce (pD-GLce-parfor) is implemented using a parfor (parallel for)-loops. Two main types of applications that can benefit from parfor-loops will normally involve repetitive segments of code and include many iterations that do not take long to execute or will include a few iterations, which take long to execute [86]. In this sense, it is quite similar to the shared memory parallel programming standard OpenMP, but has some critical differences and limitations [105]:

- *parfor* loops do not provide any communication between multiple workers. Without coordination between workers, it is not possible to get exclusive access to a shared variable for the duration of the construct;
- *parfor* loop requires iterations to be completely independent of each other, i.e., the result should not depend on the order in which iterations are executed;

The pseudocode of the `pD-GLce-parfor` algorithm using `parfor`-loops is illustrated in Algorithm 5. By default, parallel language functions automatically create a parallel pool of `MatLab` workers. The default number of workers is one per physical CPU core using a single computational thread [61]. In the `pD-GLce-parfor` implementation, the number of workers can be adjusted by specifying the input parameter ϖ.

Algorithm 5: Pseudo code of the `pD-GLce-parfor` algorithm

input : $\mathcal{P}$, ε_{pe}, ε_{φ}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$, ϖ `// ϖ is the total number of workers`
output: $f_{\min}$, $\mathbf{x}_{\min}$, pe, k, fe;
Initialization step `// only one worker - MatLab client is active`
while $pe > \varepsilon_{\text{pe}}$ **and** $fe < \mathsf{FE}_{\max}$ **and** $k < \mathsf{K}_{\max}$ **do** `//` pe `defined in Eq. (3.7)`
  Selection step: *Identify* the index set $\mathbb{S}_3^k$ of potential optimal hyper-rectangles using Definition 2 and Definition 3
  for $i = 1$: length of $\mathbb{S}_3^k$ `// slice information for further calculations`
    Make sliced input variables of potential optimal hyper-rectangles with index i
  end
  parfor $\{i = 1$: length of $\mathbb{S}_3^k,\ \varpi\}$ `// run parallel loop on ϖ workers`
    Sampling step: *Evaluate* objective and constraint functions at the centers of new hyper-rectangles
    Subdivision step: *Trisect* hyper-rectangle with index i
  end
  for $i = 1$: length of $\mathbb{S}_3^k$ `// update MatLab client's global variables`
    Applying dynamic data structures, update partitioned search space information using sliced output variable with index i
  end
  Update $f_{\min}$, $\mathbf{x}_{\min}$, pe, k, fe
end
return $f_{\min}$, $\mathbf{x}_{\min}^k$, pe, k, fe

At the beginning (**Initialization** step), only one worker – a `MatLab` client,

is active. Here, the initialization of the variables, the normalization of the domain D, and the first evaluation of the objective function is accomplished. The stopping condition is also controlled by the `MatLab` client, based on the required tolerance ε_{pe}, the maximal number of function evaluations $\text{FE}_{\max}$ and the maximal number of iterations $\text{K}_{\max}$. The next **Selection** step cannot be parallelized using `parfor`-loops since the final result (selection of potentially optimal hyper-rectangles using `DIRECT-GL` strategy) depends on the order in which iterations are executed during this step. Therefore this part is also performed only by the `MatLab` client. On the left side of Fig. 6.1, different algorithmic parts cost is illustrated by running a sequential `DIRECT-GLce` algorithm. In a particular case, about 30% of the total work per iteration (yellow color) is performed only by the Matlab client.

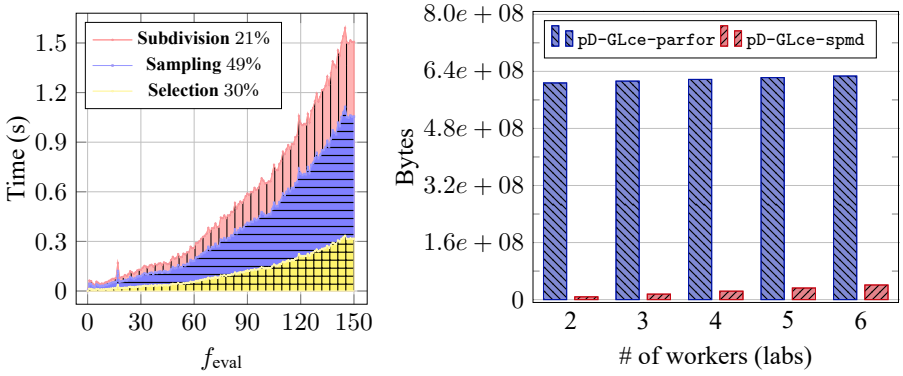

Figure 6.1: Cost of algorithmic steps per iteration on the left side and the amount of data transferred to and from a workers (labs) on the right side solving *Sphere* test problem, with $n = 50$

To reduce the communication time between the `MatLab` client and workers, the `MatLab` client must prepare sliced variables for each worker (see Algorithm 5, Line 5). Sliced variables are arrays whose segments are handled by different workers on separate loop iterations. The returned data from workers is also sliced and must be stored on the `MatLab` client using an information storing approach (see Algorithm 5, Line 12). Let us stress that static data structures are more suitable for `parfor`-loops, where sliced input and output variables could be handled more efficiently. However, significantly effective dynamic data structures here are adapted for

information storage (see the right side of Fig. 2.4 and Fig. 6.3), but this is not possible using `parfor`-loops. Therefore, synchronizing the partitioned information to global variables is done outside `parfor`-loop (see Algorithm 5, Line 12). All these replications are involved in the **Subdivision** step, and in a particular case, about 21% more of the total work per iteration (red color) is performed only by the Matlab client, see the left side of Fig. 6.1.

As pointed out in Section 2.4, the main design challenge of the parallel `DIRECT`-type algorithms is a strong data dependency. Therefore, without proper mechanisms to coordinate multiple workers using the `parfor` paradigm, the total percentage of work performed in parallel is limited. Using `parfor`-loops, only the **Sampling** step can be executed in parallel (see Algorithm 5, Lines 8 and 9). Note that even after 150 iterations(see left side of the Fig. 6.1), only about 49% of the total work per iteration (blue color) can be performed in parallel. Taking into account that operating a loop in parallel adds additional runtime costs, the `MatLab` client has to start the workers, calculations must be divided among them and later collected from the workers, there is an obvious overhead. The low cost of computations and a large amount of moving data between workers (see the right side of Fig. 6.1), can result in a noticeable overhead, especially at initial iterations (see Section 6.3 for a detailed performance analysis). Next, the `MatLab` client issues the `parfor` command and coordinates with the existing workers to execute the loop iterations in parallel. At the final termination, the best objective function value ($f_{\min}$), the point ($\mathbf{x}_{\min}$), and performance metrics (pe, k, fe) are returned by the `MatLab` client.

The low cost of computations and a large amount of moving data between workers (see the right side of Fig. 6.1), can result in a noticeable overhead, especially at initial iterations (see Section 6.3 for the detailed performance analysis).

6.2.2 Parallel `pD-GLce-spmd` algorithm

Better functionality and flexibility to parallel programming in `MatLab` is offered by *Message Passing Interface* (`MPI`). The `MatLab` message-passing paradigm uses the `smpd` keyword and additional functions described in the

point-to-point communications standard (MPI-2) [33]. Therefore, the second parallel version of pD-GLce-parfor is implemented using MPI functionality within MatLab, which is used to allocate the work across multiple labs in the MatLab software environment. Each lab stores information on their main memory block and data is exchanged through message passing over the interconnection network. The master-slave paradigm is used to implement dynamic load balancing in pD-GLce-parfor. The flowchart of the parallel algorithm is illustrated in Fig. 6.2. One lab is a master (denoted by M). The

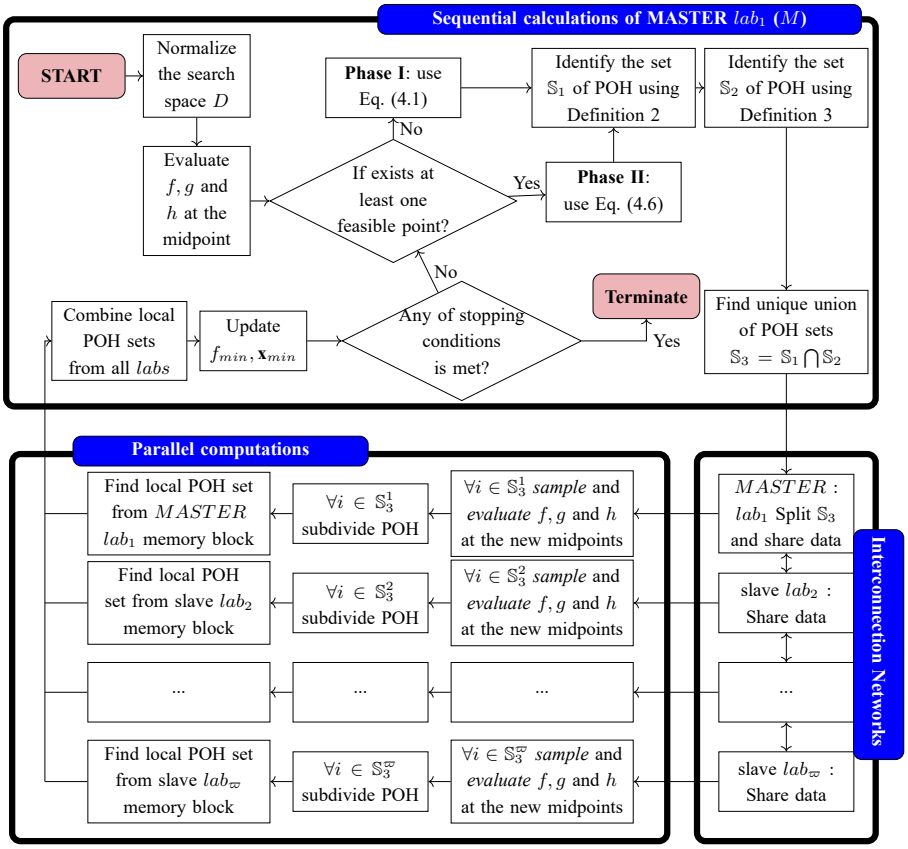

Figure 6.2: Flowchart diagram for the pD-GLce-spmd algorithm.

master lab decides which hyper-rectangles will be sampled and subdivided, and how these tasks will be distributed among all available slave labs. Additionally, the master lab is responsible for stopping the algorithm. The master lab also performs load balancing by distributing the selected hyper-rectangles to the rest of the slave labs. When the slave labs

$W_i, i = 1, \ldots, \varpi - 1$ receive tasks from the master lab, each of them performs the **Sampling** and **Subdivision** steps sequentially, and then finds a local set of potential optimal hyper-rectangles and sends a local data for the further global **Selection** step back to the master lab and becomes idle until further instructions will be received. If any of the termination conditions are satisfied, all slave labs receive the notification that the master lab has become inactive, and the slave labs will terminate themselves without further messaging.

In order to preserve determinism, each iteration must be done in a sequence. Moreover, many calculations per iteration must be done sequentially and some of them only by one lab. This situation causes parallel overhead by keeping other computational resources to stand idle. Sequential and parallel parts of `pD-GLce-parfor` are separated into blocks (see flowchart diagram for the `pD-GLce-parfor` algorithm in Fig. 6.2). The sequential section is performed only by the master lab, which organizes every iteration by making decisions and initial calculations such as the normalization of the domain D, and the first evaluation of the objective function. The master lab runs the identification of potentially optimal hyper-rectangles and creates tasks for all the labs. The new tasks are distributed to all labs, and every lab evaluates the tasks in their own set. This approach problem is that differences in evaluation times can cause some labs to finish their tasks early and become idle, while other labs continue to work on their assignments. For better load balancing, the master lab gives instructions to the slave labs who have an excess of tasks to share them with those who have the deficit. Shared data includes all relevant information regarding certain hyper-rectangles, which must be subdivided into the current iteration. The total amount of data transferred to and from all labs is shown in Fig. 6.1 right side. After the **Sampling** and **Subdivision** steps, shared data with all new data received after trisection is stored inside the new lab's memory blocks. Each lab finds its own local set of the potential optimal hyper-rectangles. The master lab gathers all of the local sets from the other slave labs and finds the global set (which is used in the global **Selection** step), updates the information, and checks the stopping conditions.

The pseudocode in Algorithm 6, shows the control mechanism between the

Algorithm 6: Pseudo code of the pD-GLce-spmd algorithm

input : $\mathcal{P}$, ε_{pe}, ε_{φ}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$, ϖ ; `// ϖ is the total number of labs`
output: $f_{\min}$, $\mathbf{x}_{\min}$, pe, k, fe;

spmd $\{\varpi\}$ `// creates parallel pool and uses all ϖ labs`
  if M **then** `// if labindex = 1`
    M receives parameters (Problem $\mathcal{P}$, D, and stopping conditions ε_{pe}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$);
    Perform **Initialization** step ; `// set tag = 1 if stop, otherwise 0`
    labSend $(\{P, D\}, W_i, tag)$; `// sends P, D, and tag to` $W_i, \forall i$
  else `// if labindex ≠ 1`
    $[\{\mathcal{P}, D\}, M, tag]$ = **labReceive** ; `// receives P, D, ang tag from M`
    if $tag == 0$ **then** `// if stopping condition is not met`
      labSend $(\emptyset, M)$; `// sends a handshaking message to M`
    end
  end
  while $pe > \varepsilon_{\text{pe}}$ **and** $fe < \mathsf{FE}_{\max}$ **and** $k < \mathsf{K}_{\max}$ **do** `// pe defined in Eq. (3.7)`
    if M **then** `// if labindex = 1`
      $[\hat{\mathbb{S}}_i, W_i]$ = **labReceive** ; `// receives information from` $W_i, \forall i$
      if *any of stopping condition is met* **then**
        $tag = 1$;
        labSend $(\emptyset, W_i, tag)$; `// terminate slave labs`
        output: $f_{\min}$, $\mathbf{x}_{\min}$, pe, k, fe;
        exit;
      else
        Perform global **Selection** step and find POH set $\mathbb{S}$;
        Split $\mathbb{S}$ among all labs equally $\mathbb{S}^{equal}$; `// finds # of POH,` $\forall i$
        Find index sets $\mathbb{S}^{poh} = \mathbb{S} \cap \hat{\mathbb{S}}$; `// locate` $\mathbb{S}$ `elements,` $\forall i$
        labSend $(W_i, \{\mathbb{S}^{equal}, \mathbb{S}^{poh}\}, tag)$; `// tell` $W_i, \forall i$`, which POH to process`
        $G.shared$ = **Distributor** $(\mathbb{S}^{equal}, \mathbb{S}^{poh}, \hat{\mathbb{S}})$; `// distribute workload, Algorithm 7`
        labBarrier ; `// block execution until all labs reach this point`
        Perform **Sampling** and **Division** steps and store information locally;
        Perform the local **Selection** step and find the local POH set $\hat{\mathbb{S}}_1$;
      end
    else `// if labindex ≠ 1`
      $[\{\mathbb{S}^{equal}, \mathbb{S}^{poh}\}, M, tag]$ = **labReceive** ; `// receive instructions from` M
      if $tag == 0$ **then**
        $\mathbb{S}.shared$ = **Distributor** $(\mathbb{S}^{equal}, \mathbb{S}^{poh}, \hat{\mathbb{S}}_i)$; `// distribute workload,Algorithm 7`
        labBarrier ; `// wait tilll all labs reach this point`
        Perform **Sampling** and **Division** steps, store information locally;
        Perform the local **Selection** step and find POH set $\hat{\mathbb{S}}_i$ on the W_i memory ;
        labSend $(\mathbb{S}_i, M)$; `// send local POH set data to` M
      else
        exit;
      end
    end
  end
end
return $f_{\min}$, $\mathbf{x}^k_{\min}$, pe, k, fe;

Algorithm 7: Workload distribution among the labs in the `pD-GLce-spmd` algorithm

function *Distributor* ($\mathbb{S}^{equal}, \mathbb{S}^{poh}, \hat{\mathbb{S}}_i$)
  $\mathbb{S}.shared = \emptyset$; `// receive information from other labs`
  if $\mathbb{S}_i^{equal} <$ card$\{\mathbb{S}_i^{poh}\}$ **then** `// if lab has an excess of potential optimal hyper-rectangles`
    $\xi =$ card$\{\mathbb{S}_i^{poh}\} - \mathbb{S}_i^{equal}$; `// find an excessive number` ξ `of potential optimal hyper-rectangles`
    $\mathbb{S}.excess = \{s_j : s_j \in \hat{\mathbb{S}}_i, j \in \mathbb{S}_i^{poh}, j = 1, \ldots, \eta\}$; `// slice excessive potential optimal hyper-rectangles from local memory`
    Using $\mathbb{S}^{equal}$ and $\mathbb{S}^{poh}$ find lab index exc, which has the largest deficit of potential optimal hyper-rectangles;
    labSend ($\mathbb{S}.excess, W_{exc}$) ; `// share excessive potential optimal hyper-rectangles`
  else if $\mathbb{S}_i^{equal} >$ card$\{\mathbb{S}_i^{poh}\}$ **then** `// if the lab has an deficit of potential optimal hyper-rectangles`
    Using $\mathbb{S}^{equal}$ and $\mathbb{S}^{poh}$ find sets of lab indexes def and exc, which has excess and deficit of potential optimal hyper-rectangles ;
    Sort exc in descending order ;
    if $exc(i) ==$ max(exc) **then** `// if the lab has largest deficit of potential optimal hyper-rectangles`
      [$\mathbb{S}.excess, W_{def}$] = **labReceive** ; `// wait untill other lab(s) share potential optimal hyper-rectangles`
      Take the needed amount of information form $\mathbb{S}.shared = \mathbb{S}.excess \cap exc(i)$;
      if card$\{exc\} \sim 1$ **then** `// is there any more labs with the deficit?`
        $\mathbb{S}.excess = \mathbb{S}.excess - exc(i)$;
        labSend ($\mathbb{S}.excess, W_{exc(2)}$) ; `// share excessive potential optimal hyper-rectangles`
      end
    else
      for $j = 1 :$ card$\{exc\} - 1$
        [$\mathbb{S}.excess, W_{exc(j)}$] = **labReceive** ; `// wait untill other lab(s) share potential optimal hyper-rectangles`
        Take the needed amount of information form $\mathbb{S}.shared = \mathbb{S}.excess \cap exc(i)$;
        if *card*$\{exc\} - 1 < j$ **then**
          $\mathbb{S}.excess = \mathbb{S}.excess - exc(i)$;
          labSend ($\mathbb{S}.excess, W_{exc(j+1)}$) ; `// share excessive potential optimal hyper-rectangles`
        end
      end
    end
  end
end
Return $\mathbb{S}.shared$

master lab M and the slave labs $W_i, i = 1, \ldots, \varpi - 1$, while Algorithm 7 shows how the workload is dynamically distributed among all the labs. In order to preserve the determinism, each iteration must be done in a sequence. The `pD-GLce-spmd` is a single start algorithm, and only the master lab M performs the **Initialization** step. Only optimization problems and the domain D are shared with the slave labs W_i. The stopping condition is also controlled only by the master lab M, which is based on the required tolerance ε_{pe}, the maximal number of function evaluations FE_{max} and the maximal number of iterations K_{max}. Whenever the master lab detects that any of stopping conditions are met, it terminates all W_i by sending a message to them and returns the best objective function value (f_{min}), the point ($\mathbf{x}_{\text{min}}$), and performance metrics (pe, k, fe).

Before each iteration, M collects local sets of the potential optimal hyper-rectangles from every slave lab $W_i, i = 1, \ldots, \varpi - 1$ including itself. M combines the results from all local sets, and finds the global set and performs the global **Selection** step. The master lab splits the whole set of potential optimal hyper-rectangles equally among all $\varpi - 1$ slave labs and itself, with the priority that potential optimal hyper-rectangles stored inside any W_i or M remain there and will be subdivided first. All labs $W_{exc}, exc \in \{1, \ldots, \varpi - 1\}$, which have an excess of potential optimal hyper-rectangles, share them with those that have the deficit $W_{def}, def \in \{1, \ldots, \varpi - 1\}$. When all W_i collect necessary instructions and data, they perform the **Sampling** and **Subdivision** steps simultaneously. All information obtained during the last two steps is stored by the labs. Finally, all W_i find their own local potential optimal hyper-rectangles sets and send it to M.

6.2.3 Parallel `pD-ACe-spmd` algorithm

An aggressive selection strategy [1] is oriented to generating more hyper-rectangles, while significantly sacrificing the performance of the sequential `DIRECT`-type algorithm (see Section 6.3). To achieve this, the approach selects and divides at least one hyper-rectangle from every existing different measure. The incorporation of such a selection strategy within a parallel algorithm may help to balance the workload. The `pD-ACe-spmd`

algorithm uses the same parallel scheme presented in Fig. 6.2, but with the aggressive selection, instead of two-step selection procedure used in `pD-GLce-spmd`. The `pD-ACe-spmd` algorithm selects potential optimal hyper-rectangles using only values of the objective and auxiliary function for each size of hyper-rectangles, i.e., the best hyper-rectangle for each size is selected. This way, the `pD-ACe-spmd` algorithm performs the same amount of work as in the version proposed in [35]. By the same analogy, instead of starting from a single hyper-rectangle (as in `pD-GLce-parfor`), the `pD-ACe-spmd` algorithm performs the initial partitioning of the search domain D without evaluating the objective function and selecting potential optimal hyper-rectangles until a number of hyper-rectangles becomes large enough to distribute to all slave labs. Furthermore, identically to [34], to reduce the memory requirement, a technique for restricting the number of columns is used, and the maximal number of diameters was allowed to be one thousand (as it was suggested in [18]).

6.3 Numerical investigation

Because of the `DIRECT`-type algorithms' iterative nature, the parallelization has not been widely investigated previously, and a comparison with the existing versions is limited. A parallel `DIRECT`-type algorithm proposed and extended in [34, 35, 36, 37, 38], looks promising for expensive global optimization problems with large dimensionality ($n = 150$). However, due to the usage of an aggressive selection scheme and additional step, which generates extra function evaluation tasks for just keeping labs busy as much as possible, such a strategy has obvious drawbacks from the sequential optimization perspective. A massively parallel `DIRECT`-type algorithm divides more sub-optimal regions, therefore increases the number of function evaluations, but significantly reduces the optimization efficiency of the algorithm (see Table 6.1). Therefore such an approach requires a massively parallel supercomputer when solving higher-dimensional problems [18, 25, 34].

Tested algorithms

First, two different implementations based on static and dynamic data structures of `DIRECT-GLce` algorithms are investigated. Three different parallel algorithms were investigated. Two of them are shared (`pD-GLce-parfor`) and distributed (`pD-GLce-spmd`) parallel implementations of the original `DIRECT-GLce` algorithm, and one is a distributed parallel version for the `Aggressive DIRECT` counterpart (`pD-ACe-spmd`).

Benchmark problems

The performance evaluation of the parallel `DIRECT`-type algorithms were made using test problems from the `DIRECTlib`. First, the performance test using box-constrained global optimization test problems is considered. Five test problems(*Ackley*, *Griewank*, *Michalewicz*, *Rosenbrock*, *Sphere*) were used with different dimensionality $n = 10, 20, 30, 40, 50$. Next, five problems with general constraints (*G02*, *G07*, *G16*, *G19*), including one practical problem (*NASA speed reducer problem*) are considered. The experimental results were carried out on a multi-core computer with 8th Generation Intel R Core^{TM} i7-8750H @ 2.20GHz Processor, which has 6 cores and 16 GB of RAM and hyperthreading disabled. Performance analysis was carried out using physical cores and enabled hyper-threading.

Stopping criteria

Since all the global minima f^* of the all test problems are known, the parallel algorithms stopped when a percent error pe Eq. (3.7) smaller than the tolerance value $\varepsilon_{\text{pe}} = 1$ or the execution time exceeds 10^4 seconds.

Evaluation of the performance of the parallel algorithms

To evaluate parallel algorithms' efficiency, the speed-up ratio is used, which shows how much faster the algorithm runs in parallel. If T_1 is the time of the best sequential algorithm and the parallel algorithm on ϖ processors takes T_ϖ

time, then the speed-up ratio is given by the formula:

$$S(\varpi) = \frac{T_1}{T_\varpi}. \tag{6.1}$$

Another metric to measure the performance of a parallel algorithm is the efficiency of parallelization. The efficiency is defined as the ratio of speed-up and the number of processors. The efficiency ratio measures how much available processing power is being used and it is defined as

$$F(\varpi) = \frac{S}{\varpi}. \tag{6.2}$$

6.3.1 Group 1: DIRECT-GLce performance with static and dynamic data structures

The DIRECT-GLce algorithm was implemented using both static and dynamic data structures. Static Data Structures are employed in S-DIRECT-GLce, whereas dynamic Data Structures are used in D-DIRECT-GLce. S-DIRECT-GLce is based on the improved Finkel's implementation [18]. Both implementations are stopped when the number of function evaluations exceeds 10^6. The test problem *G19* with $n = 15$ and 5 non-linear inequality constraint functions from DIRECTlib is used in the experimental comparison. In Fig. 6.3 the comparison of time cost of different versions is illustrated. As it was expected, the implementation based on dynamic data structures (D-DIRECT-GLce) outperforms implementation (S-DIRECT-GLce) with static data structures. After the termination, D-DIRECT-GLce version requires 37% less total execution time compared to S-DIRECT-GLce version, and the difference grows with the increase of the number of function evaluations. The main advantage of D-DIRECT-GLce is that the algorithm working only with the least values from each column as shown in Fig. 2.3, while S-DIRECT-GLce implementation makes unnecessary recalculations, and always works with all data received at previous **Sampling** and **Division** steps. The evaluation of objective (and auxiliary) functions depend on the complexity of test problems. For the current G19 problem using D-DIRECT-GLce selection of potentially optimal hyper-rectangles (**Selection** step) costs 26% of the total time in D-DIRECT-GLce, which is about 36

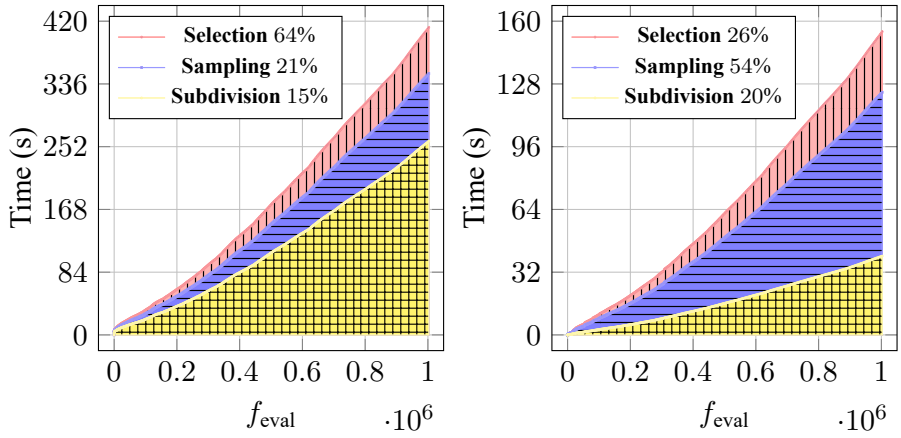

Figure 6.3: Geometric interpretation of running time (s) of the two different `DIRECT-GLce` algorithm implementations: with static data structures (*left*) [18] and with dynamic data structures (*right*) solving the *G19* test problem

seconds, while it takes around 7 times more – around 265 seconds using `S-DIRECT-GLce`. **Sampling** step takes about 85 seconds (which corresponds to 54% of the total execution time) in `D-DIRECT-GLce` and about 87 seconds (which corresponds to only 21% of the total time) using `S-DIRECT-GLce`. The **Subdivision** step, in `S-DIRECT-GLce` version works requires approximately 15% ($\sim$ 62 seconds) of the total execution time, while `D-DIRECT-GLce` handles the step $\sim$ 2 times more efficiently and requires only 20%($\sim$ 31 seconds) of time.

6.3.2 Group 2: performance test on box-constrained optimization problems

Next, the performance test using box-constrained global optimization test problems is considered. First, the best sequential algorithm among `DIRECT`, `DIRECT-GL`, and `Aggressive DIRECT` is determined. Table 6.1 shows in bold the best sequential algorithm time T_1, which will be used to measure the parallel performance. The introduced `DIRECT-GL` algorithm significantly outperforms `DIRECT` and `Aggressive DIRECT` in most of the cases. Therefore, for further experiments, the best sequential algorithm time T_1

(given in bold) is used.

Table 6.1: Results of the DIRECT, DIRECT-GL and Aggressive DIRECT algorithms solving four box-constrained global optimization test problems from the DIRECTlib

Label	n	DIRECT			DIRECT-GL			Aggressive DIRECT		
		T_1(s)	f_{eval}	k	T_1(s)	f_{eval}	k	T_1(s)	f_{eval}	k
Ackley	10	10^4	N/A	N/A	**2.09**	43,061	77	10.04	317,905	76
	20	10^4	N/A	N/A	**13.85**	331,347	153	83.93	2,700,473	152
	30	10^4	N/A	N/A	**65.59**	1,091,065	228	296.35	6,604,939	227
	40	10^4	N/A	N/A	**301.68**	2,518,409	304	856.85	12,288,961	303
	50	10^4	N/A	N/A	**895.96**	4,817,507	380	3,357.73	19,599,909	379
Griewank	10	**0.79**	3,107	36	1.25	24,719	36	0.86	29,575	23
	20	**3.07**	14,591	60	5.42	152,279	60	11.64	393,415	43
	30	9.39	35,363	84	**9.38**	260,761	64	44.77	1,460,359	63
	40	**17.94**	47,573	87	25.73	604,797	87	111.86	3,113,503	86
	50	10^4	N/A	N/A	**44.67**	836,027	107	238.96	5,107,193	106
Rosenbrock	10	7.75	23,547	119	**1.19**	37,593	55	2.09	67,867	35
	20	10^4	N/A	N/A	**7.03**	199,141	88	26.36	913,793	68
	30	10^4	N/A	N/A	**28.07**	617,991	128	90.89	2,904,299	109
	40	10^4	N/A	N/A	**96.14**	1,417,105	169	256.32	5,780,293	149
	50	10^4	N/A	N/A	**298.99**	2,648,993	208	604.18	9,516,989	189
Sphere	10	7.75	16,481	29	**0.35**	10,495	29	1.08	34,857	25
	20	10^4	N/A	N/A	**2.45**	83,301	60	17.83	637,283	55
	30	10^4	N/A	N/A	**10.44**	268,189	90	70.18	2,182,329	86
	40	10^4	N/A	N/A	**31.14**	616,831	120	166.21	4,383,403	116
	50	10^4	N/A	N/A	**95.33**	1,170,819	150	376.79	7,286,753	147

N/A – Solution point was not reached.

The amount of work is mainly determined by the number of potential optimal hyper-rectangles in each iteration. In order to have a good parallel efficiency, this number has to be large enough to feed the process units. Figs. 2.7 and 3.2 show the number of potential optimal hyper-rectangles for aggressive DIRECT and DIRECT-GL algorithms for the *Rosenbrock* test problem. All test problems used in comparison Table 6.1 are cheap, and the average cost of function evaluation is approximately equal to 10^{-6} seconds. The number of hyper-rectangle diameters increases equally in each iteration of pD-ACe-spmd. However, the number of potential optimal hyper-rectangles

selected by DIRECT-GL per iteration has significant variance, and massive amounts of evaluations are not assured, even in the last iterations.

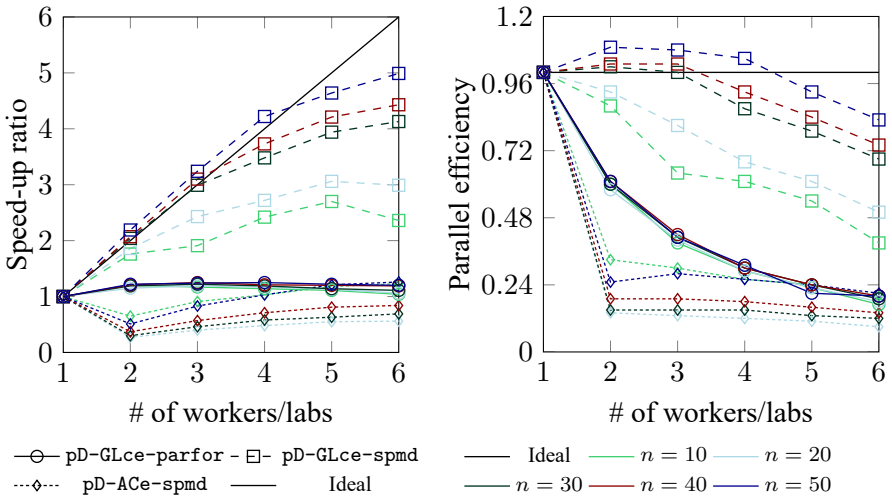

Figure 6.4: Parallel speed-up (*left*) and efficiency (*right*) on the *Sphere* problem and dimensions n

Figure 6.4 plots the parallel speed-up and efficiency of pD-GLce-parfor, pD-GLce-spmd, and pD-ACe-spmd on the *Sphere* test problem with different dimensions $n = 10, 20, 30, 40, 50$. pD-GLce-spmd achieved ideal efficiency for small dimension instances ($n = 10, 20$) using up to two labs. These instances of the problem do not exhibit enough work, causing a low increase in the speed-up when more labs are used. The parallel performance of pD-GLce-spmd improves as n increases, which is close to linear speed-up with five labs for $n = 50$, even though function evaluations are cheap. It is worth noting that the operating system and other system tasks are running in one out of the PC's six cores.

The pD-GLce-parfor algorithm achieved an almost flat speed-up, just above 1, due to the data exchange cost, no matter how many cores are used. Meanwhile, the efficiency of pD-ACe-spmd is even worse because it always lasts more time than the best sequential algorithm.

Fig. 6.5, shows the performance results of all three parallel implementations solving problems from DIRECTlib with dimension $n = 50$. Again, pD-GLce-parfor suffers from data movement and pD-ACe-spmd from oversampling,

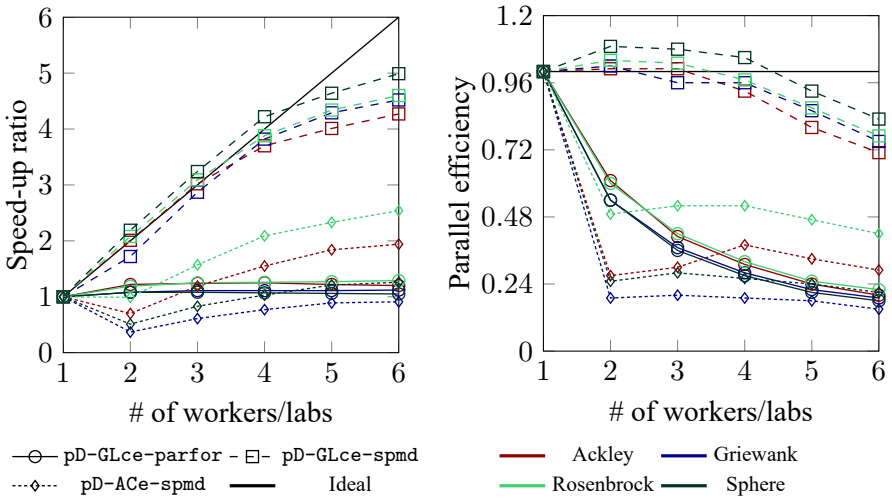

Figure 6.5: Parallel speed-up (*left*) and efficiency (*right*) on four box-constrained optimization test problems. $n = 50$

but results improve for `pD-ACe-spmd`. The `pD-GLce-spmd` shows an excellent parallel efficiency using two, three, and four labs, but it starts to decline using more labs. The `pD-GLce-spmd` is a single start, and the different number of hyper-rectangles' size increases slowly, which results in a small number of evaluations per iteration.

Fig. 6.6 shows how the efficiency of `pD-GLce-spmd` increases with the first 100 iterations on *Michalewicz* test problem with $n = 25$ and $T_{delay} = 0$. In the initial 10 iterations, there is not enough work for all available workers (labs), and the parallel overhead dominates in the early stages of `pD-GLce-spmd`. However, the results are improving when the number of iterations is increasing, as the algorithm produces more different box diameters, and as a result, more function evaluation tasks. Moreover, when a larger number of labs is used a slower start of the algorithm is seen. The parallel efficiency has a significant variance as the number of iterations grows, which can be explained as the result of different sizes of the potential optimal hyper-rectangles set using `DIRECT-GL` selection scheme. Even in the late iterations, such a scheme can leave many labs to stand idle by producing only a few potential optimal hyper-rectangles per iteration. Nevertheless, in long iterative progress, the total efficiency ratio of the algorithm grows, and

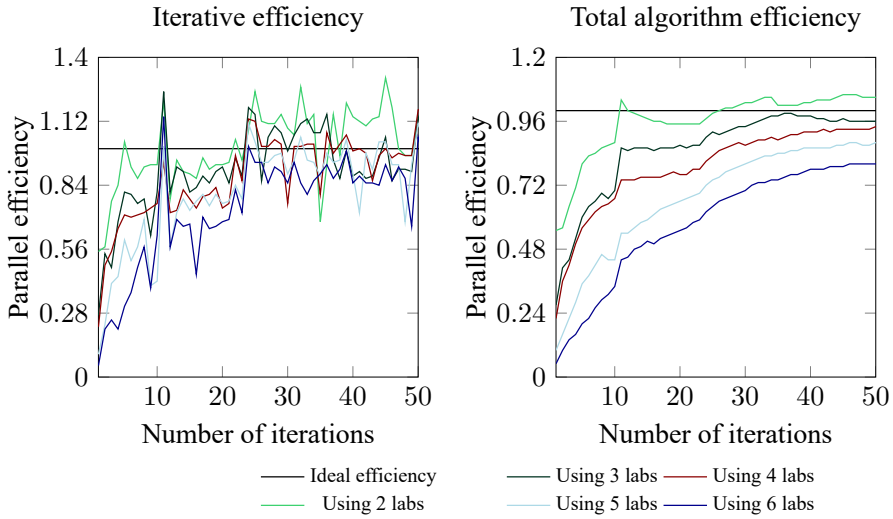

Figure 6.6: Efficiency of the `pD-GLce-spmd` algorithm on the *Michalewicz* test problem in the first 50 iterations. n=25

approaches close to the ideal parallelization.

One of the critical parameters to get a good parallel performance is the cost of the objective function. In [34, 35, 36, 37, 38] the authors add a $T_{delay} = 10^{-1}$ (in seconds) to increase the evaluation cost. Fig. 6.7 shows that when the evaluation cost of the *Sphere* test problem with $n = 10$ is increased ($T_{delay} = 10^{-4}$, 10^{-3}, 10^{-2}, 10^{-1}), the parallel efficiency increases as well, for both `pD-GLce-parfor` and `pD-GLce-spmd` algorithms. The parallel performance is almost ideal for $T_{delay} = 10^{-1}$. The efficiency of `pD-ACe-spmd` is entirely different, and such a cost of evaluations does not give tangible benefits.

6.3.3 Group 3: performance test on generally-constrained optimization problems

Unlike the massively parallel `DIRECT`-type algorithm from [35], `DIRECT-GLce` presented in this thesis can handle general constraints natively. The constraint handling technique raises the cost of the algorithm with an extra calculations and constraint functions evaluations. These additional computations may increase the efficiency of parallel `DIRECT`-type algorithms.

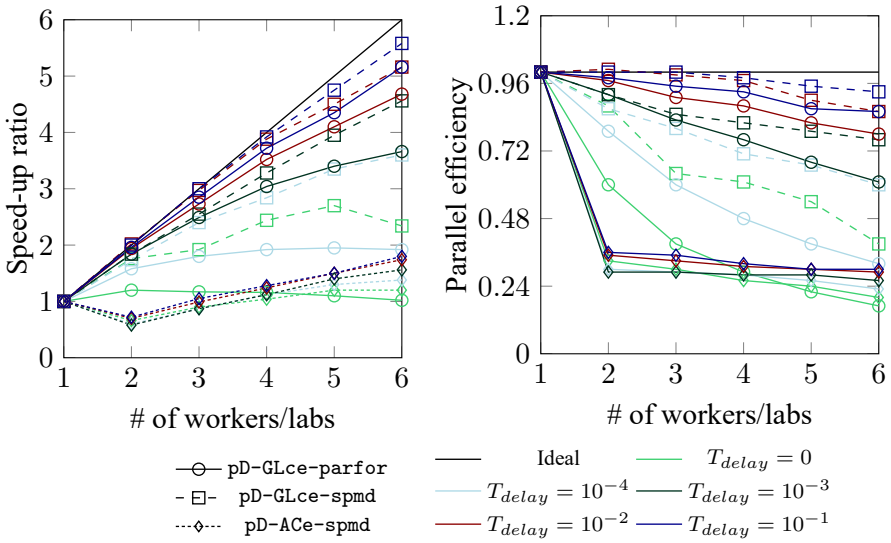

Figure 6.7: Parallel efficiency on the *Sphere* test problem with $n = 10$ and different values of T_{delay}

Next, five problems with general constraints, including one practical problem are considered. The sequential `DIRECT-GLce` algorithm requires less than a second to solve low dimensional test problems ($n \leq 10$). Therefore, the efficiency of its parallel versions on such problems cannot be expected to be good. Thus, the four most challenging test problems from [109], and the practical NASA speed reducer design problem from [88] are selected. The cost of their objective function evaluation is similar to previously used test problems, which is approximately equal to 10^{-6}s., while the constraint functions are from 10 to 100 times more expensive.

Table 6.2 shows the results of the sequential `DIRECT-L1` [18], `DIRECT-GLce` and `Aggressive DIRECT-Ce` algorithms, while Fig. 6.8 shows the performance of all three parallel versions solving the selected five problems. Although the number of function evaluations per iteration is significantly bigger in `pD-ACe-spmd` (see the results of sequential `Aggressive DIRECT-Ce` in Table 6.2), its oversampling results in the worst parallel performance. Again, `pD-GLce-parfor` suffer from data movement when the number of labs is greater than two and `pD-GLce-spmd` shows a good efficiency, which decreases as the number of labs increases.

Table 6.2: Results of the DIRECT-L1, DIRECT-GLce and Aggressive DIRECT-Ce algorithms on test problems with constraints

Label	n	DIRECT-L1			DIRECT-GLce			Aggressive DIRECT-Ce		
		T_1(s)	f_{eval}	k	T_1(s)	f_{eval}	k	T_1(s)	f_{eval}	k
G02	20	10^4	N/A	N/A	**486.36**	1,898,413	817	10^4	N/A	N/A
G07	10	10^4	N/A	N/A	**6.69**	37,661	79	56.26	289,687	59
G16	5	10^4	N/A	N/A	**8.22**	13,915	77	49.17	86,603	58
G19	15	10^4	N/A	N/A	**1725.23**	4,581,613	1509	10^4	N/A	N/A
NASA	7	10^4	N/A	N/A	**4.37**	25,879	103	9.46	61,185	35

N/A – Solution point was not reached.

Finally, let us note that speed-up is not always monotonically increasing using more workers/labs (see Figs. 6.7 and 6.8). The reason is that the number of selected potentially optimal hyper-rectangles in each iteration is unpredictable. For example, in solving complex multi-modal optimization problems, it may happen that in a late iteration, the best value of the objective function will be found in the largest hyper-rectangle, which will result in a minimal number of selected hyper-rectangles, and a large parallel overhead in at least a few subsequent iterations will be experienced. Furthermore, such processes can repeat many times when dealing with strongly multi-modal problems.

6.4 Conclusions

In this chapter, the first three parallel DIRECT-type algorithms were introduced for generally constrained global optimization problems. The experiments reveal that the aggressive selection of potential optimal hyper-rectangles results in a weak parallel performance on the set of test problems used here. This is mainly because pD-ACe-spmd wastes function evaluations on suboptimal regions, and optimization efficiency (based on the number of function evaluations) is significantly worse than the one achieved with the selection used in pD-GLce-parfor and pD-GLce-spmd.

The version pD-GLce-spmd, based on MatLab spmd, is the most efficient and significantly outperforms pD-GLce-parfor, based on parfor-loops, due

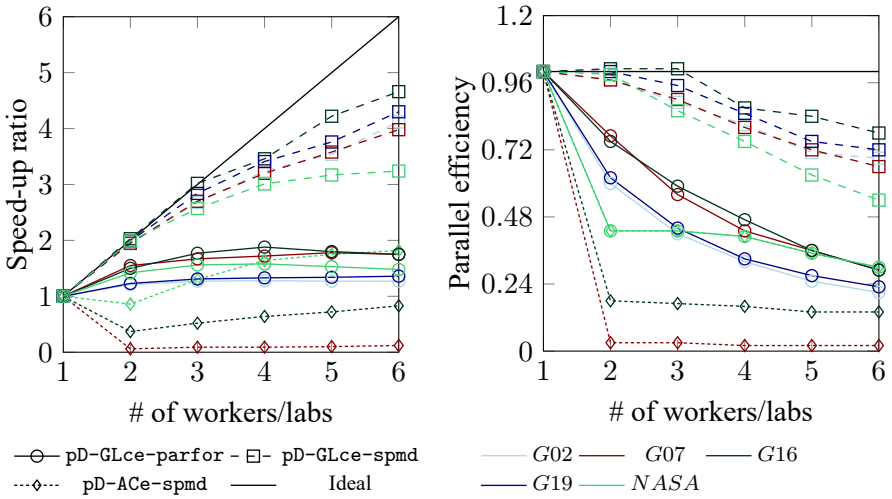

Figure 6.8: Parallel speed-up (*left*) and efficiency (*right*) on the *NASA speed reducer design problem* (see Appendix A), and test problems

to the dynamic load balancing of the former and a larger cost of the data movement in the latter. In a general context, the `MatLab parfor`-loops approach only looks reasonable when expensive global optimization problems are solved.

Finally, all introduced parallel versions preserve the determinism, while previous attempts to parallelize `DIRECT`-type algorithms lose it.

PUBLICATIONS BY THE AUTHOR

Articles in the reviewed scientific periodical publications:

1. Stripinis, L., Paulavičius, R., and Žilinskas, J. Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT, Optimization letters. Heidelberg : Springer. ISSN 1862-4472. eISSN 1862-4480. 2018, vol. 12, no 7, p. 1699-1712. DOI: 10.1007/s11590-017-1228-4.
2. Stripinis, L., Paulavičius, R., and Žilinskas, J. Penalty functions and two step selection procedure based DIRECT-type algorithm for constrained global optimization, Structural and multidisciplinary optimization. Heidelberg : Springer-Verlag. ISSN 1615-147X. eISSN 1615-1488. 2019, vol. 59, no 1, p. 2155-2175. DOI: 10.1007/s00158-018-2181-2.
3. Stripinis, L., Casado L. G., Žilinskas, J. and Paulavičius, R. On MATLAB experience in accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization, Applied Mathematics and Computation. ISSN: 0096-3003. 2021, vol. 390, p. 1-17. DOI: 10.1016/j.amc.2020.125596
4. Stripinis, L. and Paulavičius, R. A modified DIRECT-GL algorithm for global optimization with hidden constraints, Optimization letters. (2020) 1–15 submitted.

GENERAL CONCLUSIONS

1. First, a new selection scheme for the selection of potentially optimal hyper-rectangles based algorithm `DIRECT-GL` has been developed. Using the proposed approach two well-known weaknesses of `DIRECT`-type algorithms were effectively addressed. Extensive experimental studies have shown the potential of the developed `DIRECT-GL` algorithm compared to other existing `DIRECT`-type algorithms:
   1.1 Overall, `DIRECT-GL` requires around 59% less function evaluations to reach $\varepsilon_{\text{pe}} = 10^{-2}$ precision from the global solution compared

with the second best BIRECT algorithm.

1.2 Where a more accurate solution is needed ($\varepsilon_{\text{pe}} = 10^{-8}$), the developed algorithm in the overall required $\sim 72\%$ less function evaluations compared with the second best PLOR algorithm.

1.3 Overall, DIRECT-GL solved $\sim 32\%$ more box-constrained test problems compared with the second-best algorithm, DIRECT Version 4.0, in this class.

2. Next, a new auxiliary function-based algorithm DIRECT-GLce has been developed to address global optimization problems with general constraints in the DIRECT-framework. For the considered test and practical engineering problems, DIRECT-GLce performance is for all problems among the best ones, and on average, is the best one among all state-of-the-art DIRECT-type algorithms:

2.1 Developed DIRECT-GLc algorithm has the most wins, and solved $\sim 55\%$ of the problems with the highest efficiency.

2.2 Overall, DIRECT-GLce solved $\sim 43\%$ more test problems compared with the second-best algorithm DIRECT-L1.

2.3 Hybridized DIRECT-GLce-min is the most effective algorithm among all pure and hybridized DIRECT-type methods, and overall, solved the largest number of tested problems $(85/88)$, out of them $(62/88)$ with the highest efficiency.

3. A new auxiliary function-based algorithm DIRECT-GLh has been developed to address global optimization problems with hidden constraints. Experimental studies revealed the potential of developed algorithm compared with all existing DIRECT-type algorithms in this class:

3.1 Developed DIRECT-GLh algorithm has the most wins, and solved $\sim 52\%$ of the test problems with the lowest number of function evaluations, among all other DIRECT-type approaches.

3.2 Overall, DIRECT-GLh required $\sim 42\%$ less function evaluations compared with the second best DIRECT-NAS algorithm.

4. Created dynamic data structures for a better data storage and organization, increased the developed algorithms' speed by approximately $\sim 62\%$.

5. The proposed `pD-GLce-spmd` is the first deterministic parallel `DIRECT`-type algorithm for general global optimization problems, and achieves a good parallel efficiency on a multi-core infrastructure.

Bibliography

[1] C. A. Baker, L. T. Watson, B. Grossman, W. H. Mason, and R. T. Haftka. Parallel global aircraft configuration design space exploration. In A. Tentner, editor, *High Performance Computing Symposium 2000*, pages 54–66. Soc. for Computer Simulation Internat, 2000.

[2] M. C. Bartholomew-Biggs, S. C. Parkhurst, and S. P. Wilson. Using DIRECT to solve an aircraft routing problem. *Computational Optimization and Applications*, 21(3):311–323, 2002.

[3] A. Basudhar, C. Dribusch, S. Lacaze, and S. Missoum. Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization*, 46(2):201–221, 2012.

[4] Lorenz T. Biegler and Ignacio E. Grossmann. Retrospective on optimization. *Computers & Chemical Engineering*, 28(8):1169–1192, 2004.

[5] E. G. Birgin, C. A. Floudas, and J. M. Martínez. Global minimization using an augmented lagrangian method with variable lower-level constraints. *Mathematical Programming*, 125(1):139–162, 2010.

[6] Mattias Björkman and Kenneth Holmström. Global optimization using the DIRECT algorithm in Matlab. *Advanced Modeling and Optimization*, 1(2):17–37, 1999.

[7] L. C. Cagnina, S. C. Esquivel, and C. A. Coello Coello. Solving engineering optimization problems with the simple constrained particle swarm optimizer. *Informatica (Ljubljana)*, 32(3):319–326, 2008.

[8] R. G. Carter, J. M. Gablonsky, A. Patrick, C. T. Kelley, and O. J. Eslinger. Algorithms for noisy problems in gas transmission pipeline optimization. *Optimization and Engineering*, 2(2):139–157, 2001.

[9] Ron Choy and Alan Edelman. Parallel MATLAB: Doing it right. *Proceedings of the IEEE*, 93(2):331–341, 2005.

[10] Maurice Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, 1999.

[11] M. F. P. Costa, A. M. A. C. Rocha, and E. M. G. P. Fernandes. Filter-based direct method for constrained global optimization. *Journal of Global Optimization*, 71(3):517–536, 2018.

[12] Steven E. Cox, Raphael T. Haftka, Chuck A. Baker, Bernard Grossman, William H. Mason, and Layne T. Watson. A comparison of global optimization methods for the design of a high-speed civil transport. *Journal of Global Optimization*, 21(4):415–432, 2001.

[13] Teodor Gabriel Crainic, Bertrand Le Cun, and Catherine Roucairol. *Parallel Branch-and-Bound Algorithms*, chapter 1, pages 1–28. John Wiley & Sons, Ltd, 2006.

[14] D. Di Serafino, G. Liuzzi, V. Piccialli, F. Riccio, and G. Toraldo. A modified DIviding RECTangles algorithm for a problem in astrophysics. *Journal of Optimization Theory and Applications*, 151(1):175–190, 2011.

[15] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[16] Gintautas Dzemyda, Vydūnas Šaltenis, and Vytautas Tiešis. *Optimizavimo metodai: vadovėlis informatikos krypties doktorantams ir magistrantams*. Vilnius: Mokslo aidai, 2007.

[17] A. ERDOĞAN YILDIRIM and A. KARCI. Application of three bar truss problem among engineering design optimization problems using

artificial atom algorithm. In *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, pages 1–5, 2018.

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

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

[20] D. E. Finkel and C. T. Kelley. Additive scaling and the DIRECT algorithm. *Journal of Global Optimization*, 36(4):597–608, 2006.

[21] R. Fletcher. *Practical Methods of Optimization*. John and Sons Chichester, 2nd edition, 1987.

[22] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239—269, 2002.

[23] Christodoulos A Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Springer US, 1999.

[24] A. I. J. Forrester and A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1):50–79, 2009.

[25] J. M. Gablonsky. *Modifications of the DIRECT Algorithm*. PhD thesis, North Carolina State University, 2001.

[26] J. M. Gablonsky and C. T. Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21(1):27–37, 2001.

[27] Bernard Gendron and Teodor Gabriel Crainic. Parallel branch-and-branch algorithms: Survey and synthesis. *Operations research*, 42(6):1042–1066, 1994.

[28] V. P. Gergel, K. A. Barkalov, and A. V. Sysoyev. Globalizer: A novel supercomputer software system for solving time-consuming global optimization problems. *Numerical Algebra, Control and Optimization*, 8(1):47–62, 2018.

[29] V. P. Gergel, V. A. Grishagin, and A. V. Gergel. Adaptive nested optimization scheme for multidimensional global search. *Journal of Global Optimization*, 66(1):35–51, 2016.

[30] Ratko Grbić, Emmanuel Karlo Nyarko, and Rudolf Scitovski. A modification of the DIRECT method for Lipschitz global optimizatio n for a symmetric function. *Journal of Global Optimization*, 57(4):1193–1212, 2013.

[31] Joshua D Griffin and Tamara G Kolda. Asynchronous parallel hybrid optimization combining direct and gss. *Optimization Methods & Software*, 25(5):797–817, 2010.

[32] Vladimir A Grishagin, Yaroslav D Sergeyev, and Roman G Strongin. Parallel characteristical algorithms for solving problems of global optimization. *Journal of Global Optimization*, 10(2):185–206, 1997.

[33] William Gropp, Ewing Lusk, and Rajeev Thakur. Using MPI-2: Advanced features of the message-passing interface. *Computers & Mathematics with Applications*, 2000.

[34] J. He, A. Verstak, L. T. Watson, and M. Sosonkina. Design and implementation of a massively parallel version of direct. *Computational Optimization and Applications*, 2008.

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

[36] Jian He, Alex Verstak, Masha Sosonkina, and Layne T Watson. Performance modeling and analysis of a massively parallel DIRECT–Part 2. *The International Journal of High Performance Computing Applications*, 23(1):29–41, 2009.

[37] Jian He, Alex Verstak, Layne T Watson, and Masha Sosonkina. Performance modeling and analysis of a massively parallel DIRECT–part 1. *The International Journal of High Performance Computing Applications*, 23(1):14–28, 2009.

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

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

[40] Juan F.R. Herrera, José M.G. Salmerón, Eligius M.T. Hendrix, Rafael Asenjo, and Leocadio G. Casado. On parallel Branch and Bound frameworks for Global Optimization. *Journal of Global Optimization*, 69(3):547–560, 2017.

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

[42] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer, Berlin, 1996.

[43] David E Hudak, Neil Ludban, Vijay Gadepally, and Ashok Krishnamurthy. Developing a computational science ide for hpc systems. In *Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC'07)*, pages 1–5. IEEE, 2007.

[44] H John. Adaptation in natural and artificial systems. *The University of Michigan Press, Ann Arbor*, 1975.

[45] D. R. Jones. The DIRECT global optimization algorithm. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *The Encyclopedia of Optimization*, pages 431–440. Kluwer Academic Publishers, Dordrect, 2001.

[46] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian

optimization without the Lipschitz constant. *Journal of Optimization Theory and Application*, 79(1):157–181, 1993.

[47] M. Kazemi, G. G. Wang, S. Rahnamayan, and K. Gupta. Metamodel-based optimization for problems with expensive objective and constraint functions. *Journal of Mechanical Design*, 133(1):14505, 2011.

[48] J. Kennedy and R. Eberhart. Particle swarm optimization. *In Proceedings of the IEEE international conference on neural networks IV*, 1995.

[49] Jeremy Kepner and Stan Ahalt. MatlabMPI. *Journal of Parallel and Distributed Computing*, 2004.

[50] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 1983.

[51] Tamara G Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[52] H. Liu, S. Xu, X. Chen, X. Wang, and Q. Ma. Constrained global optimization via a direct-type constraint-handling technique and an adaptive metamodeling strategy. *Structural and Multidisciplinary Optimization*, 55(1):155–177, 2017.

[53] Qunfeng Liu and Wanyou Cheng. A modified DIRECT algorithm with bilevel partition. *Journal of Global Optimization*, 60(3):483–499, 2014.

[54] Qunfeng Liu, Guang Yang, Zhongzhi Zhang, and Jinping Zeng. Improving the convergence rate of the DIRECT global optimization algorithm. *Journal of Global Optimization*, 67(4):851–872, 2017.

[55] Qunfeng Liu, Jinping Zeng, and Gang Yang. MrDIRECT: a multilevel robust DIRECT algorithm for global optimization problems. *Journal of Global Optimization*, 62(2):205–227, 2015.

[56] G. Liuzzi, S. Lucidi, and V. Piccialli. A DIRECT-based approach exploiting local minimizations for the solution of large-scale global optimization problems. *Computational Optimization and Applications*, 45:353–375, 2010.

[57] Giampaolo Liuzzi, Stefano Lucidi, and Veronica Piccialli. A DIRECT-based approach exploiting local minimizations for the solution for large-scale global optimization problems. *Computational Optimization and Applications*, 45(2):353–375, 2010.

[58] Giampaolo Liuzzi, Stefano Lucidi, and Veronica Piccialli. A partition-based global optimization algorithm. *Journal of Global Optimization*, 48(1):113–128, 2010.

[59] Giampaolo Liuzzi, Stefano Lucidi, and Veronica Piccialli. Exploiting derivative-free local searches in DIRECT-type algorithms for global optimization. *Computational Optimization and Applications*, 65:449–475, 2016.

[60] Piotr Luszczek. Parallel programming in MATLAB. *International Journal of High Performance Computing Applications*, 23(3):277–283, 2009.

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

[62] Seyedali Mirjalili and Andrew Lewis. The Whale Optimization Algorithm. *Advances in Engineering Software*, 2016.

[63] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey Wolf Optimizer. *Advances in Engineering Software*, 2014.

[64] Jonas Mockus, Remigijus Paulavičius, Dainius Rusakevičius, Dmitrij Šešok, and Julius Žilinskas. Application of Reduced-set Pareto-Lipschitzian Optimization to truss optimization. *Journal of Global Optimization*, 67(1-2):425–450, 2017.

[65] J. Na, Y. Lim, and C. Han. A modified DIRECT algorithm for hidden constraints in an LNG process optimization. *Energy*, page 488–500, 2017.

[66] T. D. Panning, L. T. Watson, N. A. Allen, K. C. Chen, C. A. Shaffer, and J. J. Tyson. Deterministic parallel global parameter estimation for a model of the budding yeast cell cycle. *Journal of Global Optimization*, 2008.

[67] P. M. Pardalos and H. E. Romeijn, editors. *Handbook of Global Optimization*, volume 2. Kluwer Academic Publishers, Dordrecht, 2002.

[68] R Paulavičius, J Gao, P-M Kleniati, and C. S Adjiman. BASBL: Branch-And-Sandwich BiLevel solver: Implementation and computational study with the BASBLib test set. *Computers & Chemical Engineering*, 132:106609, 2020.

[69] R. Paulavičius and J. Žilinskas. Analysis of different norms and corresponding Lipschitz constants for global optimization. *Technological and Economic Development of Economy*, 36(4):383–387, 2006.

[70] R. Paulavičius and J. Žilinskas. Analysis of different norms and corresponding Lipschitz constants for global optimization in multidimensional case. *Information Technology and Control*, 36(4):383–387, 2007.

[71] Remigijus Paulavičius, Lakhdar Chiter, and Julius Žilinskas. Global optimization based on bisection of rectangles, function values at diagonals, and a set of Lipschitz constants. *Journal of Global Optimization*, 71(1):5–20, 2018.

[72] Remigijus Paulavičius, Ya. D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. Globally-biased DISIMPL algorithm for expensive global optimization. *Journal of Global Optimization*, 59(2-3):545–567, 2014.

[73] Remigijus Paulavičius, Ya. D. Sergeyev, Dmitri E. Kvasov, and Julius Žilinskas. Globally-biased BIRECT algorithm with local accelerators for expensive global optimization. *Expert Systems with Applications*, 144:11305, 2020.

[74] Remigijus Paulavičius and Julius Žilinskas. Global optimization using the branch-and-bound algorithm with a combination of Lipschitz bounds over simplices. *Technological and Economic Development of Economy*, 15(2):310–325, 2009.

[75] Remigijus Paulavičius and Julius Žilinskas. Parallel branch and bound algorithm with combination of Lipschitz bounds over multidimensional simplices for multicore computers. In *Parallel Scientific Computing and Optimization*, volume 15, pages 93–102. Springer New York, 2009.

[76] Remigijus Paulavičius and Julius Žilinskas. Simplicial Lipschitz optimization without the Lipschitz constant. *Journal of Global Optimization*, 59(1):23–40, 2013.

[77] Remigijus Paulavičius and Julius Žilinskas. *Simplicial Global Optimization*. SpringerBriefs in Optimization. Springer New York, New York, NY, 2014.

[78] Remigijus Paulavičius and Julius Žilinskas. Advantages of simplicial partitioning for Lipschitz optimization problems with linear constraints. *Optimization Letters*, 10(2):237–246, 2016.

[79] Remigijus Paulavičius, Julius Žilinskas, and Andreas Grothey. Investigation of selection strategies in branch and bound algorithm with simplicial partitions and combination of Lipschitz bounds. *Optimization Letters*, 4(2):173–183, 2010.

[80] Remigijus Paulavičius, Julius Žilinskas, and Andreas Grothey. Parallel branch and bound for global optimization with combination of Lipschitz bounds. *Optimization Methods and Software*, 26(3):487–498, 2011.

[81] Remigijus Paulavičius, Julius Žilinskas, Juan F.R. Herrera, and Leocadio G. Casado. A Parallel DISIMPL for Pile Placement

Optimization in Grillage-Type Foundations. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 525–530. IEEE, 2013.

[82] G. Di Pillo, G. Liuzzi, S. Lucidi, V. Piccialli, and F. Rinaldi. A DIRECT-type approach for derivative-free constrained global optimization. *Computational Optimization and Applications*, 65(2):361–397, 2016.

[83] G. Di Pillo, S. Lucidi, and F. Rinaldi. An approach to constrained global optimization based on exact penalty functions. *Journal of Optimization Theory and Applications*, 54(2):251–260, 2010.

[84] János D Pintér. *Global optimization in action: continuous and Lipschitz optimization: algorithms, implementations and applications*, volume 6 of *Nonconvex Optimization and Its Applications*. Springer US, 1996.

[85] S. A. Piyavskii. An algorithm for finding the absolute minimum of a function. *Theory of Optimal Solutions*, 2:13–24, 1967. in Russian.

[86] Nikolaos Ploskas and Nikolaos Samaras. *GPU Programming in MATLAB*. Morgan Kaufmann, 2016.

[87] Franco P. Preparata and Michael Shamos. *Computational Geometry*. Monographs in Computer Science. Springer-Verlag New York, 1985.

[88] Tapabrata Ray and Kim Meow Liew. Society and civilization: An optimization algorithm based on the simulation of social behavior. *IEEE Transactions on Evolutionary Computation*, 7(4):386–396, 2003.

[89] R. G. Regis. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. *Computers and Operations Research*, 38(5):837–853, 2011.

[90] R. G. Regis. Constrained optimization by radial basis function interpolation for high-dimensional expensive black-box problems with infeasible initial points. *Engineering Optimization*, 46(2):218–243, 2014.

[91] Luis Miguel Rios and Nikolaos V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2012.

[92] Ya D Sergeev and RG Strongin. A global minimization algorithm with parallel iterations. *USSR Computational Mathematics and Mathematical Physics*, 29(2):7–15, 1989.

[93] Ya D Sergeyev and VA Grishagin. A parallel method for finding the global minimum of univariate functions. *Journal of optimization theory and applications*, 80(3):513–536, 1994.

[94] Ya. D. Sergeyev and D. E. Kvasov. *Diagonal Global Optimization Methods*. FizMatLit, Moscow, 2008. In Russian.

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

[96] Ya. D. Sergeyev, D. E. Kvasov, and M. S. Mukhametzhanov. On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. *Nature Scientific Reports*, 8, 2018. article 453.

[97] Ya. D. Sergeyev and Dmitri E. Kvasov. Global search based on diagonal partitions and a set of Lipschitz constants. *SIAM Journal on Optimization*, 16(3):910–937, 2006.

[98] Ya. D. Sergeyev, R. G. Strongin, and D. Lera. *Introduction to Global Optimization Exploiting Space-Filling Curves*. SpringerBriefs in Optimization. Springer, New York, 2013.

[99] YaD Sergeyev and VA Grishagin. Sequential and parallel global optimization algorithms. *Optim. Methods Softw*, 3:111–124, 1994.

[100] Yaroslav D Sergeyev. On convergence of ”divide the best” global optimization algorithms. *Optimization*, 44(3):303–325, 1998.

[101] Yaroslav D Sergeyev. On convergence of “divide the best” global optimization algorithms. *Optimization*, 44(3):303–325, 1998.

[102] Yaroslav D Sergeyev and Dmitri E Kvasov. *Deterministic Global Optimization: An Introduction to the Diagonal Approach*. SpringerBriefs in Optimization. Springer, 2017.

[103] S. Shan and G. G. Wang. Metamodeling for high dimensional simulation-based design problems. *Journal of Mechanical Design*, 132(5):051009, 2010.

[104] S. Shan and G. G. Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, 2010.

[105] Gaurav Sharma and Jos Martin. MATLAB®: A language for parallel computing. *International Journal of Parallel Programming*, 37(1):3–36, 2009.

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

[107] Linas Stripinis and Remigijus Paulavičius. Extension of DIRECT-GLalgorithm for problems with hidden constraints. *Optimization Letters*, pages 1–12, 2020.

[108] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. Improved scheme for selection of potentially optimal hyper-rectangles in DIRECT. *Optimization Letters*, 12(7):1699–1712, 2018.

[109] Linas Stripinis, Remigijus Paulavičius, and Julius Žilinskas. Penalty functions and two-step selection procedure based DIRECT-type algorithm for constrained global optimization. *Structural and Multidisciplinary Optimization*, 59(6):2155–2175, 2019.

[110] Linas Stripinis and Remigijus Paulavičius. DIRECTLib – a library of global optimization problems for DIRECT-type methods, v1.2, 2020.

[111] Linas Stripinis, Julius Žilinskas, Leocadio G. Casado, and Remigijus Paulavičius. On MATLAB experience in accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization, 2021.

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

[113] Roman G Strongin and Yaroslav D Sergeyev. Global multidimensional optimization on parallel computer. *Parallel Computing*, 18(11):1259–1273, 1992.

[114] Roman G Strongin and Yaroslav D Sergeyev. Global optimization: fractal approach and non-redundant parallelism. *Journal of Global Optimization*, 27(1):25–50, 2003.

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

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

[117] M. J. Todt. *The Computation of Fixed Points and Applications*, volume 24 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 1976.

[118] Aimo Törn and Antanas Žilinskas. *Global optimization*, volume 350. Springer, 1989.

[119] N Travinin Bliss and Jeremy Kepner. pMATLAB Parallel MATLAB Library. *The International Journal of High Performance Computing Applications*, 21(3):336–359, 2007.

[120] Anne E Trefethen, Vijay S Menon, Chi-Chao Chang, Grzegorz Czajkowski, Chris Myers, and Lloyd N Trefethen. MultiMATLAB: MATLAB on multiple processors. Technical report, Cornell University, 1996.

[121] A.I.F. Vaz and L.N. Vicente. Pswarm: a hybrid solver for linearly constrained global derivative-free optimization. *Optimization Methods and Software*, 24(4–5):669–685, 2009.

[122] L. T. Watson and C. A. Baker. A fully-distributed parallel global search algorithm. engineering computations. *Engineering Computations*, 18(1/2):155–169, 2001.

[123] A. Žilinskas. On strong homogeneity of two global optimization algorithms based on statistical models of multimodal objective functions. *Applied Mathematics and Computation*, 218(16):8131–8136, 2012.

[124] J. Žilinskas. Branch and bound with simplicial partitions for global optimization. *Mathematical Modelling and Analysis*, 13(1):145–159, 2008.

[125] J. W. Zwolak, J. J. Tyson, and L. T. Watson. Globally optimised parameters for a model of mitotic control in frog egg extracts. *IEE Proceedings - Systems Biology*, 2005.

Appendix A

A mathematical formulations of engineering problems

Tension/compression spring design problem

The design variables are the number of the wire diameter x_1, the winding diameter x_2, and active coils of the spring x_3. The objective function and the mechanical constraints are given by:

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

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

Three-bar truss design problem

This problem has two design variables and three constraints. The optimization problem formulated as follows:

$$\begin{aligned}
\min\ & f(\mathbf{x}) = L \times (2\sqrt{2}x_1 + x_2) \\
\text{s.t. } & g_1(\mathbf{x}) = \frac{\sqrt{2}x_1 + x_2}{\sqrt{2}x_1^2 + 2x_1x_2}P - 2 \leq 0, \\
& g_2(\mathbf{x}) = \frac{x_2}{\sqrt{2}x_1^2 + 2x_1x_2}P - 2 \leq 0, \\
& g_3(\mathbf{x}) = \frac{1}{x_1 + \sqrt{2}x_2}P - 2 \leq 0
\end{aligned}$$

where $0 \leq x_1 \leq 1,\ 0 \leq x_2 \leq 1$, $L = 100$, $P = 2KN/cm^2$. The best known solution $\mathbf{x}^* = (0.78867512, 0.40824832)$, where $f(\mathbf{x}^*) = 263.89584535$. One of the constraint function is active (g_1).

NASA speed reducer design problem

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

$$
\begin{aligned}
\min\ & f(\mathbf{x}) = 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) \\
& - 1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) \\
& + 0.7854(x_4x_6^2 + x_5x_7^2) \\
\text{s.t. } & g_1(\mathbf{x}) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0,\ g_2(\mathbf{x}) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \leq 0, \\
& g_3(\mathbf{x}) = \frac{1.93x_4^3}{x_2x_3x_6^4} - 1 \leq 0,\ g_4(\mathbf{x}) = \frac{1.93x_5^3}{x_2x_3x_7^4} - 1 \leq 0, \\
& g_5(\mathbf{x}) = \frac{((\frac{745x_4}{x_2x_3})^2 + 16.9 \times 10^6)^{0.5}}{110x_6^3} - 1 \leq 0, \\
& g_6(\mathbf{x}) = \frac{((\frac{745x_5}{x_2x_3})^2 + 157.5 \times 10^6)^{0.5}}{85x_7^3} - 1 \leq 0, \\
& g_7(\mathbf{x}) = \frac{x_2x_3}{40} - 1 \leq 0,\ g_8(\mathbf{x}) = \frac{5x_2}{x_1} - 1 \leq 0, \\
& g_9(\mathbf{x}) = \frac{x_1}{12x_2} - 1 \leq 0,\ g_{10}(\mathbf{x}) = \frac{1.5x_6 + 1.9}{x_4} - 1 \leq 0, \\
& g_{11}(\mathbf{x}) = \frac{1.1x_7 + 1.9}{x_5} - 1 \leq 0
\end{aligned}
$$

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

Pressure vessel design problem

There are four design variables (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 formulated

as follows:

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

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

Welded beam design problem

The problem is to design a welded beam for minimum cost, subject to some constraints [62, 63]. The objective is to find the minimum fabrication cost. Considerating four design variables and constraints of shear stress τ, bending stress in the beam σ, buckling load on the bar P_c, and end deflection on the beam δ. The optimization model is summarized in the next equation:

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

with:

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

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 $\mathbf{x}^* = (0.20572551, 3.47062057, 9.03666456, 0.20573141)$, where $f(\mathbf{x}^*) = 1.72488430$. One of the constraint function is active (g_3).

Linas Stripinis

IMPROVEMENT, DEVELOPMENT AND IMPLEMENTATION OF DERIVATIVE-FREE GLOBAL OPTIMIZATION ALGORITHMS

Doctoral Dissertacion
Natural Sciences
Informatics (N 009)
Editor Zuzana Šiušaitė

Vilniaus universiteto leidykla
Saulėtekio al. 9, LT-10222 Vilnius
El. p. info@leidykla.vu.lt,
www.leidykla.vu.lt
Tiražas 20 egz.