Vilnius University
Institute of Data Science and Digital Technologies
L I T H U A N I A

INFORMATICS (09 P)

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

Linas Stripinis

October 2019

Technical Report MII-DS-09P-19-1 October 2016 - 30 September 2020

VU Institute of Mathematics and Informatics, Akademijos str. 4, Vilnius LT-08663, Lithuania
www.mii.lt

Abstract

Due to its simplicity and efficiency, the derivative-free global-search DIRECT algorithm has received significant attention from the optimization community. Various novel ideas and extensions of the algorithm have been proposed, including for general constrained global optimization problems. However, the performance of DIRECT-type algorithms highly depends on the characteristics of the optimization problem and computer implementation. Most of the publicly available DIRECT implementations are using static data memory management, which is more straightforward, but usually also less effective due to the difficulty predicting memory requirements in advance. Moreover, the iterative nature of DIRECT-type algorithms presents difficulties for efficient parallelization and only a few parallel DIRECT-type implementations are known. Therefore, here we introduce dynamic data structures for a better memory balancing and develop the first DIRECT-type parallel implementation (pDIRECT-GLce) for generally constrained global optimization. The effectiveness of two different pDIRECT-GLce parallel versions is evaluated solving box and generally constrained global optimizations problems of different complexity. The performance metrics include usual criteria such as the number of function evaluations and total execution time as well as criteria of parallelization such as load balancing and parallel efficiency.

Keywords: Global optimization, DIRECT-type algorithms, Derivative-free optimization, DIRECT-type constraint-handling, Nonconvex optimization, Dynamic Data structures, Parallel optimization, Load balancing

Contents

1 Accelerating DIRECT-GLce algorithm for constrained global optimization through dynamic data structures and parallelization

Simplicity and efficiency of the DIRECT algorithm attracted considerable research interest, and many modifications and extensions have been proposed [LC14, PCŽ18, PSKŽ14, PŽ13, PŽ14, SK06, SPŽ18]. An algorithm was designed as an effective global method that avoids being trapped at local minima regions by exploring less potentially optimal hyperrectangles to converge globally for Lipschitz-continuous optimization problems. As direct search method, DIRECT produces deterministic results without derivative information or the Lipschitz constant of the objective function. As a consequence, DIRECT-type methods have been successfully used in solving various multidisciplinary optimization problems [BWG+00, BBPW02, CGP+01, PWA+08, SPŽ19, ZTW05].

Nevertheless, the DIRECT algorithm has some well-known algorithmic weaknesses, especially evident on optimization problems with many local minima and when the solution with high accuracy is sought [Jon01, PSKŽ14, SPŽ18]. Various proposals have been introduced in the literature to overcome these sources of inefficiency (see [LLP10, LLP16, PSKŽ14, SK06, SPŽ18] and references therein). Another source of inefficiency is related to the space-partitioning strategy used in the DIRECT-type algorithms [Jon01]. For the slow converging and high-dimensional problems, the challenge is to maintain the fast-growing partitioning information efficiently. Most of the existing DIRECT-type implementations use a static data structure to store the current state of the search space partitioning. This can lead to failure of the code if the array is insufficient to hold the necessary information. To overcome this problem, usually, implementations will reallocate the array to be significantly larger than needed. But such a modification can cost a considerable amount of overhead in both execution time and space required. These issues have been discussed, and a few versions of dynamic data structures have been proposed to overcome them [HSS+93, HVSW09, HVWS08, HVWS09, HWR+02, HWS10].

A natural way to speed up DIRECT-type algorithms is to parallelize them. However, only very few parallel implementations of DIRECT-type algorithms are known, developed mainly by the same group of researchers [HVSW09, HVWS08, HVWS09, HWS10, PŽHC13, WB01]. The very first parallel version based on an "aggressive" DIRECT algorithm [BWG+00] was introduced in [WB01], parallelized using the master-slave paradigm. In the following works [HVSW09, HVWS08, HVWS09, HWS10] the authors introduced modifications of the previous parallel version. The first new idea is to use dynamic data structure [HWR+02] for a better organization of the local data. Later the authors improved the algorithm by transforming a single start into a multi-start strategy by initial subdividing of the feasible region D into smaller subdomains. Each subdomain is controlled by a different master, with more partitions generated across multiple subdomains. Such a parallel version can scale efficiently on even larger machines. While all these proposals are focusing on achieving better parallel efficiency, from the optimization

perspective, introduced novelties make algorithms significantly less effective.

Finally, to the best of our knowledge all the existing parallel DIRECT-type versions are only for box-constrained global optimization problems. Therefore, in this paper, we introduce the first parallel DIRECT-type algorithm (called hereafter as pDIRECT-GLce) for generally constrained global optimization problems. The pDIRECT-GLce algorithm is implemented within the MATLAB software environment, hence several well-known and broadly used implementations of the original DIRECT algorithm (see, e.g. [BH99, FK06, Gab01]), as well as many later DIRECT-type proposals (see, e.g. [LXC+17, LC14, LYZZ17, PŽ14]) were developed within MATLAB too.

2 Architecture of the DIRECT algorithm

The derivative-free global-search DIRECT (DIviding RECTangles) algorithm by Jones et al. [Jon01, JPS93] is an effective deterministic algorithm to solve global optimization problems subject to simple box-constraints

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

where $f : \mathbb{R}^n \to \mathbb{R}$ denotes the objective function and the feasible region is an n-dimensional hyper-rectangle $D = [\mathbf{a}, \mathbf{b}] = \{\mathbf{x} \in \mathbb{R}^n : a_j \leq x_j \leq b_j, j = 1, \ldots, n\}$. An objective function $f(\mathbf{x})$ supposes to be Lipschitz-continuous (with unknown Lipschitz constant) but can be non-linear, non-differentiable, non-convex, and multi-modal. The DIRECT algorithm seeks for the global optimum by partitioning *potentially optimal* (the most promising) *hyper-rectangles* (POH) and evaluating the objective function at the centers of these hyper-rectangles. 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 D^i. Let $\varepsilon > 0$ be a positive constant and $f_{\min}$ be the best currently known value of the objective function. A hyper-rectangle $D^j, j \in \mathbb{I}$ is said to be potentially optimal if there exists some rate-of-change (Lipschitz) constant $\tilde{L} > 0$ such that*

$$f(\mathbf{c}^j) - \tilde{L}\delta_j \leq f(\mathbf{c}^i) - \tilde{L}\delta_i, \quad \forall i \in \mathbb{I}_k, \tag{2}$$

$$f(\mathbf{c}^j) - \tilde{L}\delta_j \leq f_{\min} - \varepsilon|f_{\min}|, \tag{3}$$

where the measure of the hyper-rectangle is

$$\delta_i = \frac{1}{2}\|\mathbf{b}^i - \mathbf{a}^i\|_2. \tag{4}$$

In each iteration algorithms performs selection of such *hyper-rectangles*, which later will be fully sampled and subdivided. Brief description of the main steps of the original DIRECT

algorithm is given in Algorithm 1.

Initialization. Normalize the search space D to be the unit hyper-rectangle $\bar{D}$. Evaluate objective f at the center point $\mathbf{c}^1 \in \bar{D}$. Set $f_{\min} = f(\mathbf{c}^1)$, $\mathbf{x}_{\min} = \mathbf{c}^1$, and initialize algorithmic performance measures and *stopping criteria*.
while *stopping criteria are not satisfied* **do**
  Selection. *Identify* the sets S of potentially optimal hyper-rectangles (subregions of $\bar{D}$) using Definition 1.
  Sampling. For each hyper-rectangle $j \in S$ *sample* and *evaluate* objective functions at the centers of new hyper-rectangles along their longest dimensions. Update $f_{\min}, \mathbf{x}_{\min}$, and algorithmic performance measures.
  Subdivision. For each hyper-rectangle $j \in S$ subdivide (trisect), update partitioned search space information and *stopping criteria*.
end
Return $f_{\min}, \mathbf{x}_{\min}$, and algorithmic performance measures.

Algorithm 1: Main steps of the `DIRECT` algorithm

2.1 Overview of the `DIRECT-GLce` algorithm

All the existing parallel `DIRECT`-type versions are only for box-constrained global optimization problems. Therefore, in this article, we developed the first parallel `DIRECT`-type algorithm for generally constrained global optimization problems. Our recently proposed `DIRECT-GLce` algorithm [SPŽ19] showed very competitive results among all existing `DIRECT` extensions for such problems.

The original `DIRECT` algorithm addresses only optimization problems with bounds on the variables. In the last decade, a few different constraint handling strategies (including our recent `DIRECT-GLce` algorithm [SPŽ19]) within the `DIRECT` framework were proposed for the general optimization problem [BDLM12, CRF18, LXC+17, PLL+16, PLR10]:

$$\begin{aligned} &\min_{\mathbf{x}\in D} f(\mathbf{x}) \\ &\text{s.t. } \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \\ &\quad \mathbf{h}(\mathbf{x}) = \mathbf{0}, \end{aligned} \tag{5}$$

where $f : \mathbb{R}^n \to \mathbb{R}, \mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m, \mathbf{h} : \mathbb{R}^n \to \mathbb{R}^r$ are (possibly nonlinear) continuous functions and $D = [\mathbf{a}, \mathbf{b}] = \{\mathbf{x} \in \mathbb{R}^n : a_j \leq x_j \leq b_j, j = 1, \ldots, n\}$. The feasible region consisting of points that satisfy all the constraints is denoted by $D^{\text{feas}} = D \cap \Omega$, where $\Omega = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \text{ and } \mathbf{h}(\mathbf{x}) = \mathbf{0}\}$. Here, 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.

In [SPŽ19], a detailed comparison of five state-of-the-art `DIRECT`-type algorithms: `DIRECT-GLce` and `DIRECT-GL-L1` both from [SPŽ19], `DIRECT-L1` [Fin05], `eDIRECT-C` [LXC+17] and `Filter-DIRECT` [CRF18] was carried out. Among them, our recently introduced `DIRECT-GLce` algorithm was very competitive compared to other pro-

posals and often outperformed other algorithms solving test and engineering problems. One reason for this is that the DIRECT-GLce algorithm is based on a new two-step strategy for the selection of the extended set of potentially optimal hyper-rectangles (compared to the most of DIRECT-type methods) [SPŽ18]. In the first selection step, the set of POH is enlarged by adding more (compared to DIRECT) medium-sized hyper-rectangles, and in the second, closest to the current minimum point. This situation can be seen from Figs. 1 and 2, where it is clear, that DIRECT-GLce algorithm often selects a larger number of potentially optimal hyper-rectangles compared to DIRECT. Also, the DIRECT-GLce algorithm

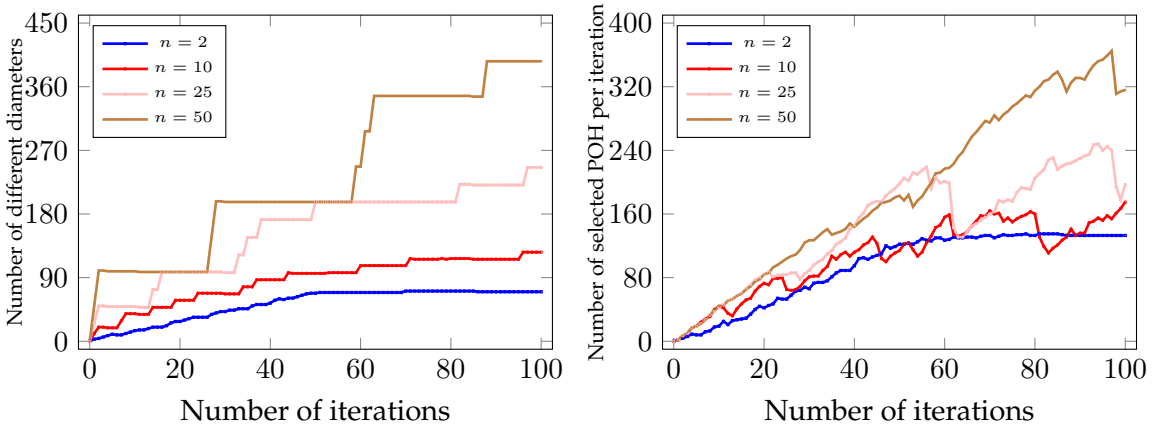

Figure 1: The growth of the number of different diameters (*left*) and the number of selected potentially optimal hyper-rectangles (POH, *right*), obtained by the DIRECT-GLce [SPŽ19] algorithm on the *Rosenbrock* test problem with different dimensionality ($n = 2, 10, 25$, and 50)

uses an auxiliary function approach, that combines information on the objective and constraint functions and does not require any penalty parameters. The DIRECT-GLce algorithm works in two phases, where during the first phase the algorithm handles infeasible initial points while in the second phase seeks to improve a located feasible solutions until a global solution is reached. In first phase algorithm employs an additional procedure, which samples the search space and minimizes not the objective function, but the sum of constraint violations, i.e.:

$$\min_{\mathbf{x}\in D} \varphi(\mathbf{x}), \tag{6}$$

where

$$\varphi(\mathbf{x}) = \sum_{i=1}^{m} \max\{g_i(\mathbf{x}), 0\} + \sum_{i=1}^{r} |h_i(\mathbf{x})|, \tag{7}$$

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

$$D^{\text{feas}}_{\varepsilon_\varphi} = \{\mathbf{x} : 0 \leq \varphi(\mathbf{x}) \leq \varepsilon_\varphi, \mathbf{x} \in D\}, \tag{8}$$

and ε_φ is a very small acceptable constraint violation. A separate phase for handling infeasible initial points is especially useful when the feasible region is small compared to

the design space. When at least one hyper-rectangle with feasible midpoint is located the efforts may be switched to finding better feasible solutions using new form of transformed problem (5):

$$\min_{\mathbf{x}\in D} f(\mathbf{x}) + \xi(\mathbf{x}, f_{\min}^{\text{feas}}),$$
$$\xi(\mathbf{x}, f_{\min}^{\text{feas}}) = \begin{cases} 0, & \mathbf{x} \in D_{\varepsilon_\varphi}^{\text{feas}} \\ 0, & \mathbf{x} \in D_{\varepsilon_{\text{cons}}}^{\text{inf}} \\ \varphi(\mathbf{x}) + \Delta, & \text{otherwise,} \end{cases} \tag{9}$$

where parameter $\Delta = |f(\mathbf{x}) - f_{\min}^{\text{feas}}|$ 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 midpoint. $D_{\varepsilon_{\text{cons}}}^{\text{inf}}$ is a small tolerance for constraint function sum, which automatically varies during the optimization process, but any detailed description of the $\varepsilon_{\text{cons}}$ controlling model will not be gives in this paper.

Original `DIRECT-GLce` version in each iteration performs selection of potentially optimal hyper-rectangles twice [SPŽ18], and finds an independent sets G and L there algorithm separately handles them. Version which used in this paper slightly differs compared to [SPŽ19]. In the current version of `DIRECT-GLce`, identification of these two sets is performed in succession and the unique union of these two sets ($P = G \cup L$) will be performed in Algorithm 1 Algorithm 1. This modification introduced seeking to reduce data dependency in the algorithm for better parallelization. Before the **Selection** step `DIRECT-GLce` decides in which phase algorithm works. If there exist at least one feasible midpoint ($\exists \mathbf{x} \in D_{\varepsilon_\varphi}^{\text{feas}}$), algorithm activates phase two and uses (9) for identification of potential optimal hyper-rectangles, otherwise algorithm works under phase one, and uses (6) in **Selection** step. The rest of **Sampling** and **Subdivision** steps remains the same as in original `DIRECT`. For a more detailed description of `DIRECT-GLce`, see [SPŽ19].

2.2 Design challenges of parallel `DIRECT`-type algorithms

The iterational nature of the existing `DIRECT`-type algorithms limits possibilities for effective parallelism [HVSW09, HVWS08, HVWS09, HWS10, PŽHC13, WB01]. The biggest challenge is a strong data dependency between different iterations and a quite small number of selected potentially optimal hyper-rectangles (even solving higher dimensionality problems) to process further, which does not allow to use many computational cores efficiently. In Fig. 2, the growth of the number of different diameters and the number of selected potentially optimal hyper-rectangles (POH), obtained by the `DIRECT` 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 POH is quite small.

Developing a parallel `DIRECT` version, the first design challenge comes in the very first **Initialization** step (see Algorithm 1, Algorithm 1), where the algorithm starts from a

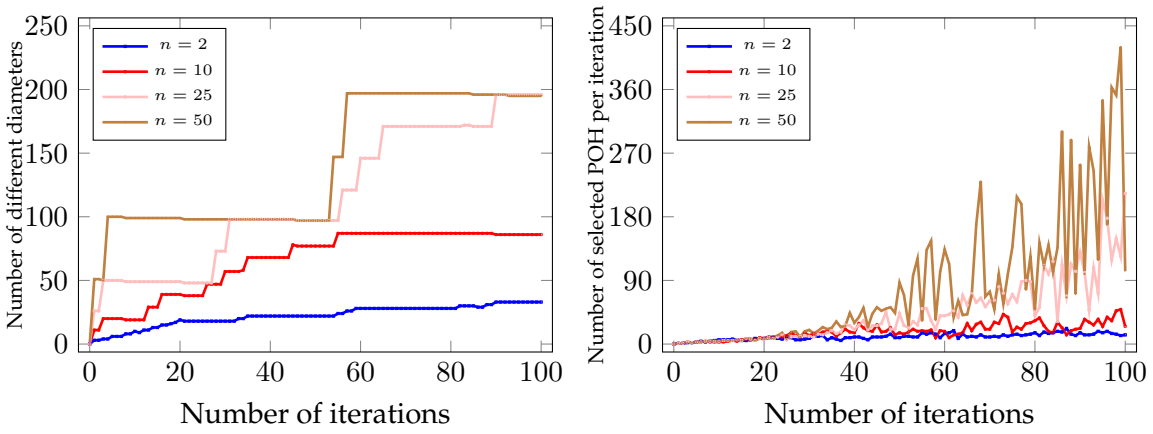

Figure 2: The growth of different diameters (*left*) and the number of selected potentially optimal hyper-rectangles (POH, *right*), obtained by the `DIRECT` algorithm on the *Rosenbrock* test problem with different dimensionality ($n = 2, 10, 25$, and 50)

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 load balancing issue is more critical, as the number of different partitions grows slower (see Fig. 2), compared to the problems of higher dimensionality.

Another important design challenge comes in the **Sampling** step (see Algorithm 1, Algorithm 1), which cannot be started until the **Selection** step is finished (see Algorithm 1, Algorithm 1) and vice versa. Only the **Sampling** and **Subdivision** (see Algorithm 1, Algorithm 1) can be parallelized efficiently. The **Selection** step is very hard to parallelize efficiently, especially preserving 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** procedure is increasing when the algorithm processes longer (see Fig. 6), therefore reduces 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. 1). When the number of selected POH is small, this results that insufficient work will be sent to workers, and processors will not be used efficiently, some of them possibly being idle.

2.2.1 Proposed ideas to handle the parallel challenges of `DIRECT`

The authors of previous works proposed to create multiple starting points for each worker in the **Initialization** step. The performance analysis in [HVSW09, HVWS09] revealed that such an idea improves the load balancing and the overall parallel efficiency. However, using different size of workers on the same problem it requires an uneven size of initial starting points, but that destroys determinism of the algorithm.

Also most of previous parallel `DIRECT`-type versions [HVSW09, HVWS08, HVWS09,

HWS10, WB01] focused on improving parallel efficiency by creating more computations in every iteration. Therefore, they used a different scheme to select potential optimal hyper-rectangles. The authors in [BWG+00] 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. The aggressive version relaxed the criteria of selection of po-

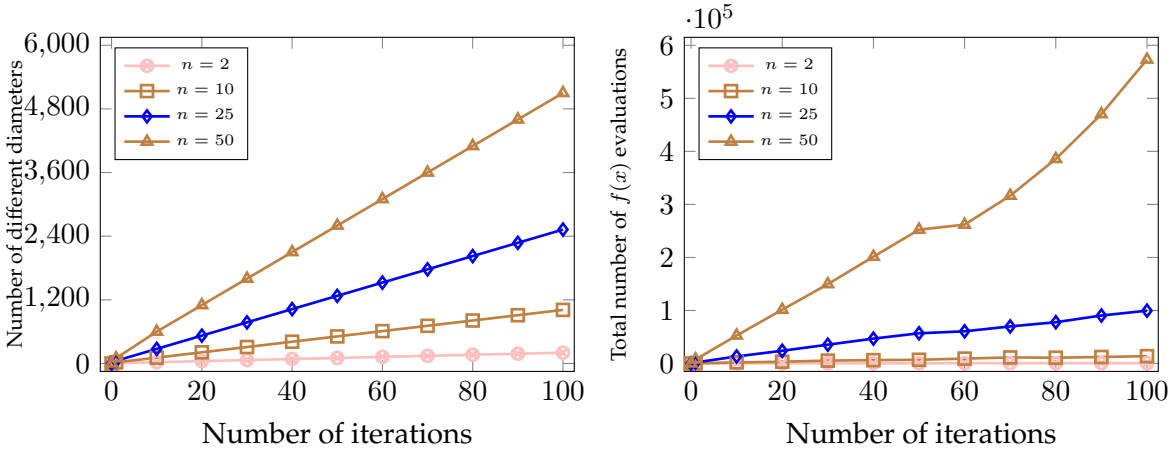

Figure 3: The growth of the number of different diameters (*left*) and the total number of objective function evaluations per iteration (*right*), obtained by the "aggressive" DIRECT algorithm [BWG+00] on the *Rosenbrock* test function with different dimensionality ($n = 2, 10, 25,$ and 50)

tentially optimal hyper-rectangles, thus assuring a much higher number of selected POH per iteration (which coincides with the number of different diameters, see Fig. 3), 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 aggressive DIRECT version grows much faster compared to the other DIRECT-type versions (see Figs. 1 to 3). 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 therefore, 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 will diminish the optimization effectiveness of the DIRECT algorithm [FK06, Gab01].

2.3 Influence of data structures on the performance of DIRECT-type algorithms

The performance of DIRECT-type algorithms highly depends on computer implementation. Most of the publicly available DIRECT implementations are using static data memory management, which is more straightforward but usually also less effective due to the difficulty predicting memory requirements in advance. The typical data structures are used to store a collection of objective (and constraint) function values, index numbers, center point coordinates, side lengths of hyper-rectangles, and so on. One of the most broadly used publicly available DIRECT algorithm implementation [Fin04] is using static data structures (SDS) whose size is predetermined, depending on stopping conditions. All information received after space partition is stored in the contiguous blocks of memory, as shown in the left side of Fig. 4. Using such data structures the algorithm

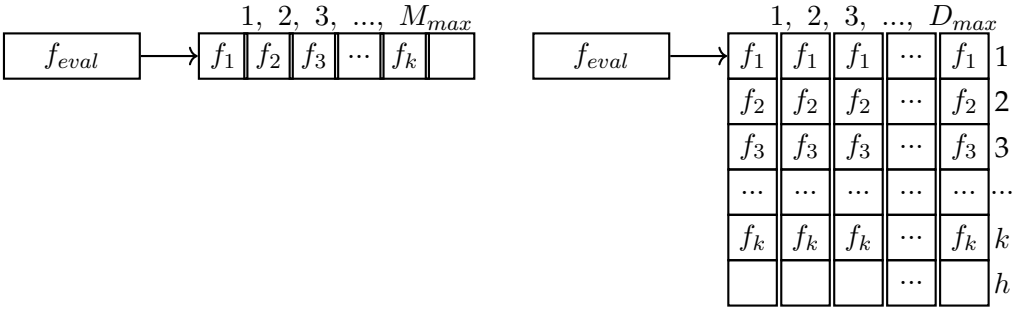

Figure 4: Information storage using static data structure (*left*) and dynamic data structure (*right*) in DIRECT-GLce algorithm

can access quickly and easily the elements for further **Selection**, **Sampling** and **Subdivision** steps. The content of the data structure can be modified without changing the memory space allocated to it while there is enough allocated memory. Otherwise, the algorithm spends extra time for re-allocating a larger contiguous block of memory and then copying/moving existing information into this new block. The original DIRECT implementation allocates a large array to store information received after **Sampling** and **Division** steps. Allocated row vector size corresponds to one of the stopping conditions, which is the maximal number of function evaluations. The main disadvantage of the static data structures used in DIRECT is that the algorithm from the first iteration reserves a large amount of memory, which slows the algorithm down and comes with an overhead, which is proportional to the size of the allocated array. In practice, choosing the maximum size of function evaluations is a guess, and the perfect selection of it is the actual number of function evaluations in which the algorithm can solve the problem. Another disadvantage of static data structures used in DIRECT is that in each iteration, the algorithm performs unnecessary recalculations. One of the most critical tasks in DIRECT is the **Selection** step, where the algorithm decides which hyper-rectangles are potentially optimal (the most promising) for further investigation, corresponding to the lower con-

vex hull of the cloud of red points (see Fig. 5). 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 this is visible for problems of higher dimensionality. To sum up, implementations using static data structures often fail in practice since it is difficult to predict memory requirements in advance. This motivates to use better structures to handle the variable amount of storage and information required by the space partition technique.

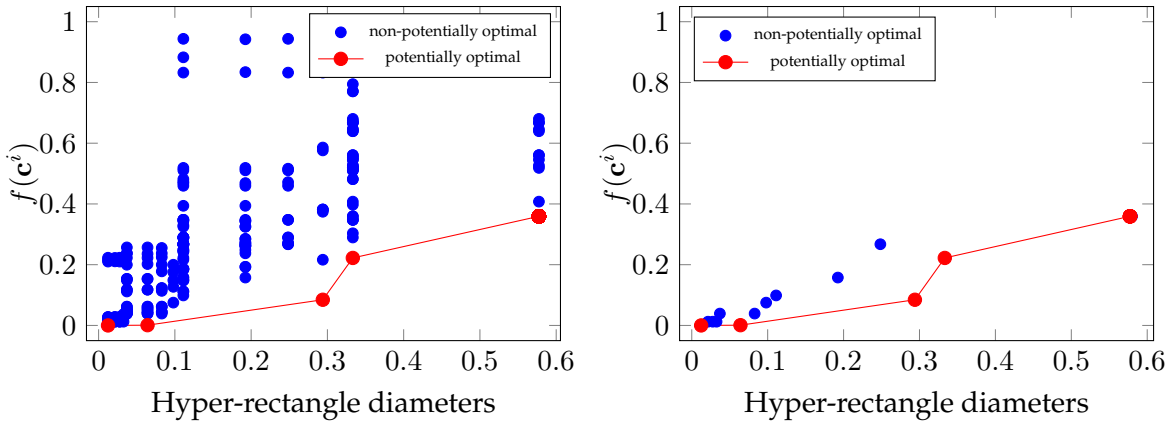

Figure 5: Selection of POH using SDS (*left*) and DDS (*right*) implementations in `DIRECT-GLce`

In [HWR+02], the authors proposed the idea to use dynamic data structures (DDS). Information received after space partition is sorted out by hyper-rectangle diameters and stored in columns, as shown in the right side of Fig. 4. All rectangles of the same diameter are stored in the column at any order. In [HWR+02], 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. With dynamic data structures, the selection of potentially optimal hyper-rectangles (in the **Selection** step) is simplified. The selection can be performed only in the set consisting of the best function values from each column, as shown in the right side of Fig. 5, and it can save a lot of time compared to the previous selection strategy. Another considerable advantage comes from the fact that **Selection** step is hardly parallelized, therefore the selection of POH using dynamic data structures not only simplifies but also significantly reduces the time needed for this step.

One of the obvious drawbacks of the dynamic data structure is a problem allocating the columns, which size is unpredictable. There are two operations that change columns size: first fully processed potential optimal hyper-rectangle needs to be removed from the previous columns and newly added to a new/existing column. During the execution of the algorithm, there can be a large number of distinct hyper-rectangle diameters, which is unpredictable. Depending on the dimension of the problem, the initial array is allocated of 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.

2.3.1 Comparison of `DIRECT-GLce` performance with static and dynamic data structures

The `DIRECT-GLce` algorithm was implemented using both static and dynamic data structures. The algorithm with static data structures (`S-DIRECT-GLce`) is implemented in the `MATLAB` programming language and is based on Finkel's implementation [Fin04]. The second version (`D-DIRECT-GLce`) is implemented in `MATLAB`, but with dynamic data structures. Both implementations are stopped when the number of function evaluations exceeds 10^6. The test problem G19 (see Table 3 in Appendix Nr. 1. and in [SP18]) is used in the experimental comparison. In Fig. 6 comparison of time cost of different versions is illustrated. As it was expected, the implementation based on dynamic data structures

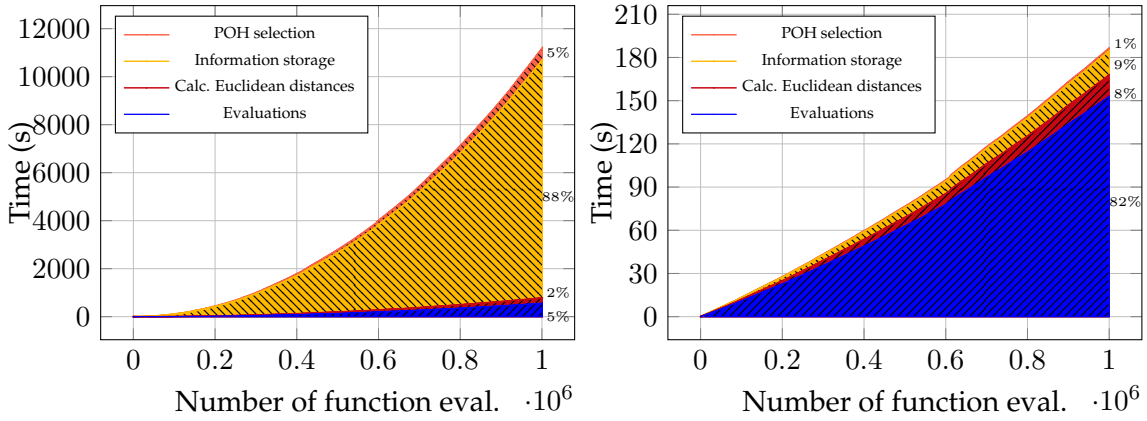

Figure 6: Geometric interpretation of running time(s) of two different `DIRECT-GLce` algorithm implementations: with static data structures (*left*) [Fin04] and with dynamic data structures (*right*) solving the G19 test problem

(`D-DIRECT-GLce`) significantly outperforms implementation (`S-DIRECT-GLce`) with static data structures. After termination, `D-DIRECT-GLce` version requires 98.6% less total execution time compared to `S-DIRECT-GLce` version, and the difference grows with the increase of the number of function evaluations. For the current G19 problem using `D-DIRECT-GLce` evaluation of objective (and auxiliary) functions takes about 150 seconds (which correspond to 82% of the total execution time) and about 578 seconds (which correspond to only 5% of the total time) using `S-DIRECT-GLce`. Selection of potentially optimal hyperrectangles costs only 1% of total time in `D-DIRECT-GLce`, which is about 1.8 seconds, while it takes around 277 times more – around 577 seconds using `S-DIRECT-GLce`.

3 Parallel Scheme and Implementation

The original DIRECT algorithm begins with a single hyper-rectangle and its center point, so there is only one candidate for the **Sampling** and **Subdivision** procedures at the first iteration. Furthermore, in initial iterations, the number of potentially optimal hyper-rectangles is small since the hyper-rectangles of different diameters are not generated quickly enough. Thus, a load imbalance between parallel workers is unavoidable at the early stage in a parallel DIRECT-type algorithm. Another difficulty is data dependency, after every step, the results from all workers must be collected. Therefore, the **Selection** and **Sampling** steps must be carried out sequentially, the **Selection** step cannot be parallelized, so it must be done by one worker. To overcome the difficulties, in this section we propose two parallel implementations of the DIRECT-GLce algorithm called pDIRECT-GLce and Aggressive pDIRECT-Ce, similar ideas may be implemented for other DIRECT-type algorithms.

3.1 *MPI* Parallel Version of pDIRECT-GLce

The parallel version of pDIRECT-GLce was implemented using a message passing interface (*MPI*) to allocate the work across multiple processors in MATLAB software environment. Each processing element stores information on their main memory block and data exchange achieved through message passing over the interconnection network. The master-slave paradigm is used to implement dynamic load balancing in pDIRECT-GLce. The scheme is illustrated in Fig. 7. One processor is a master ($Worker_1$) which makes all calculations for hyper-rectangle selection and controls the distribution of tasks to the workers. As it was described previously and shown in **??**, the DIRECT-GL algorithm selects potentially optimal hyper-rectangles in two steps in each iteration. The obtained results from both selection steps are combined into one set leaving only one hyper-rectangle index and avoiding duplication. Since the use of dynamic data structure has reduced **Selection** step time cost to 1% of the total time, the **Selection** calculations and decisions are done only by the master processor. The master also performs load balancing by distributing the selected hyper-rectangles to the workers. When the workers $W_i, i = 1, \ldots, k$ receive tasks from the master, each of them performs **Sampling** and **Subdivision** steps sequentially, sends the results back to the master and becomes idle until further instructions will be received. If any of the termination conditions is satisfied, all workers receive notification that the master has become inactive, and the workers will terminate themselves without further messaging.

In order to preserve the determinism, each iteration must be done in a row, moreover many calculations per iteration must be done in sequential, and some of them only by one processor and it will cause parallel overhead by keeping other computational resources to stand idle. Sequential and parallel parts of pDIRECT-GLce is separated in blocks, see Fig. 7. The sequential section is performed only by the master, which organizes every

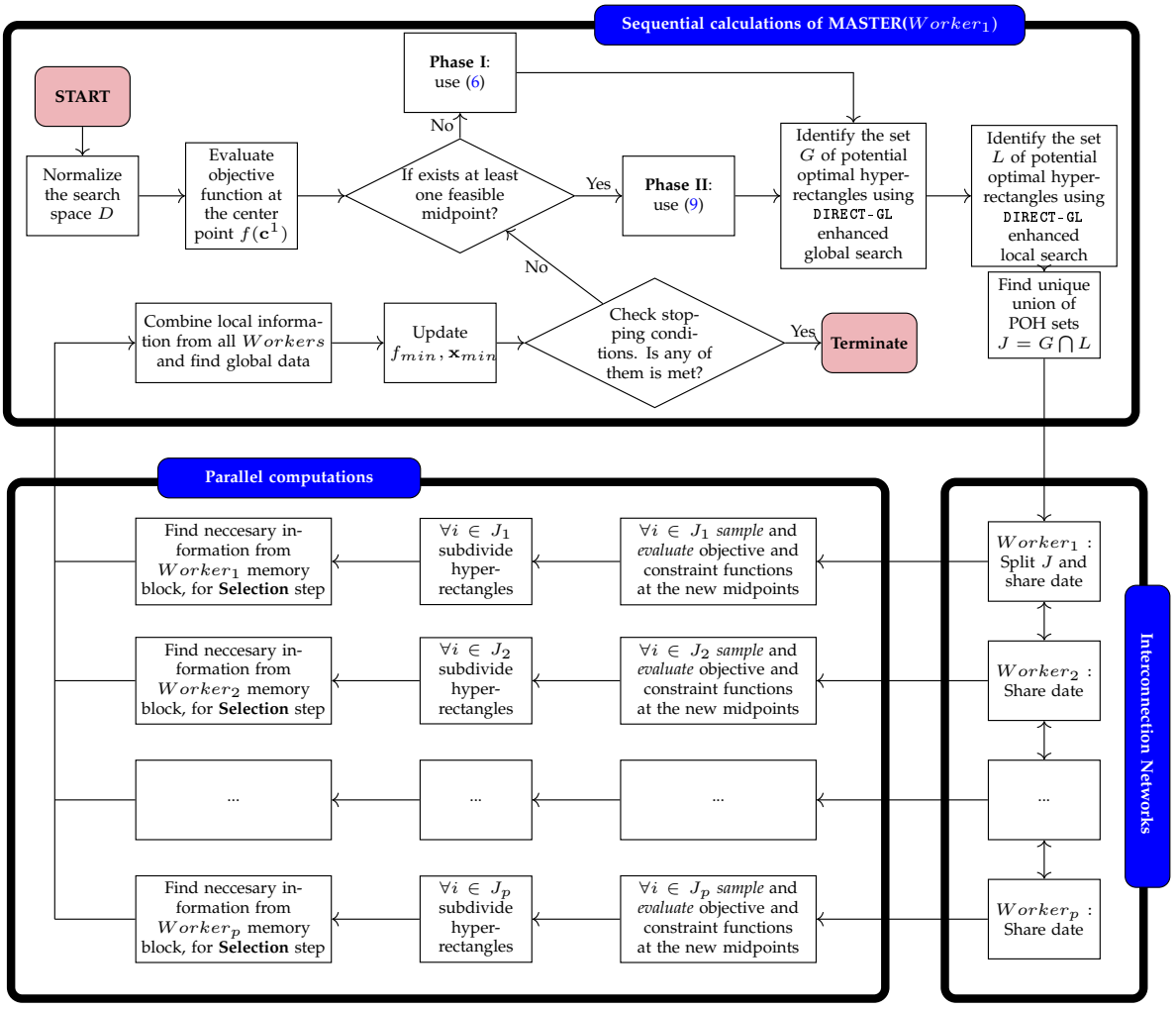

Figure 7: Flowchart diagram of optimization platform for pDIRECT-GLce algorithm.

iteration by making decisions and initial calculations such as normalization of the domain D, first objective function evaluation. The master processor runs the identification of potentially optimal hyper-rectangles and creates a set of tasks for the workers. The new tasks are distributed to all workers, and every worker evaluates the tasks in their own set. The problem is that differences in evaluation times can cause some processors to finish their tasks early and become idle, while other processors continue to work on their tasks. For a better load balancing master gives instructions to processors having 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. After **Sampling** and **Subdivision** steps, shared data with all new data received after trisection are stored inside of the new processor's memory blocks. Each worker finds its own local set of the least function values of each hyper-rectangle diameter column. The master gathers all of the local sets from the other processors and finds the global set of the least objective function values, updates information, and checks the stopping conditions.

The pseudocode in Line 2, shows more detailed control mechanism between the master M and the workers $W_i, (i = 1, \ldots, k)$. `pDIRECT-GLce` is a single start algorithm, and only the master M performs the **Initialization** step, only optimization problems, and the domain D is shared with the workers W_i. The stopping condition is also controlled only by M, which is based on the required tolerance ε_{pe}, the maximal number of function evaluations $\text{M}_{\max}$ and the maximal number of iterations $\text{L}_{\max}$. Whenever the master 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 and the point ($f_{\min}$, $\mathbf{x}_{\min}$) which was found.

Before every iteration, M collects local sets of the best function values $f(x_j), (j = 1, \ldots, p)$ of each hyper-rectangle diameter from every W_i including himself. M combines results from all local sets and finds the global set of $f(x_j)$ and performs the **Selection** step. The master splits the POH equally among all $k + 1$ workers and himself, with the priority that POH stored inside any W_i or M remains there and will be subdivided first. All workers $W_e, (e = 1, \ldots, v < k+1)$, who have an excess of POH, share them with those who have the deficit of POH $W_d, (d = 1, \ldots, u < k + 1)$. After all W_i collect necessary instructions and data, perform the **Sampling** and **Subdivision** steps. All information received by the last two steps is stored on the worker. Finally, all W_i find their own local set of the best function values $f(x_j)$ and send it to M.

3.2 `Aggressive pDIRECT-Ce` algorithm

An aggressive selection [BWG+00] generates more potentially optimal hyper-rectangles. Therefore, incorporation of such a selection in a parallel algorithm may help to balance the workload. Therefore, the `Aggressive pDIRECT-Ce` algorithm uses the same parallel scheme presented in Fig. 7, but the aggressive selection. The `Aggressive pDIRECT-Ce` algorithm selects POH using only values of objective and auxiliary function evaluations for

each size of the hyper-rectangle, i.e., the best hyper-rectangle for each size is selected. So that the algorithm could generate the same amount of work as in the version proposed in [HWS10], the additional selection step using Euclidean distances from the best point found was removed. Also, instead of starting from a single hyper-rectangle as in the original sequential algorithm, the algorithm performs initial subdivision without evaluating objective function and checking potential optimality until a number of hyper-rectangles become enough to distribute to the workers.

4 Numerical investigation

In this section we present the results of performance of the parallel `DIRECT`-type algorithms on test problems which are listed in Appendix Nr. 1. Tables 2 and 3, using a computer with 8th Generation Intel R Core^{TM} i7-8750H @ 2.20GHz Processor, which has 6 physical - 12 logical cores and 16 GB of main memory. Performance analysis was carried out using physical cores and enabled hyper-threading. The main features of these problems: problem number $(\#)$, name, source, dimensionality (n), number of constraint functions in the test problem, type of constraint functions (L, NL), variable bound (D), the number of local minima (if known), and the known minimum (f^*). Note that for the test functions presented in Table 2, the dimensionality can be chosen on demand.

To evaluate the efficiency of parallelization we used the speed-up ratio, which shows how much faster the algorithm runs in parallel. If T_1 is the time of best sequential algorithm and the parallel algorithm on p processors takes T_p time, then the speed-up ratio is given by the formula:

$$S(p) = \frac{T_1}{T_p}. \tag{10}$$

Another metric to measure the performance of a parallel algorithm is the efficiency ratio. The efficiency is defined as the ratio of speed-up to the number of processors. Efficiency

input : ε_{pe}, ε_{φ}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$;
output: $f_{\min}$, $\mathbf{x}_{\min}$, pe, k, m;
if M **then**
  M receives parameters (Problem, domain D, and stopping conditions ε_{pe}, $\mathsf{FE}_{\max}$, $\mathsf{K}_{\max}$);
  Perform **Initialization** step;
  labSend (W_i) ; // sends problem and domain D to $W_i, \forall i$
else
  labReceive (M) ; // receives problem and domain D from M
  labSend (M) ; // sends a handshaking message to M
end

```
while pe > ε_pe and m < FE_max and l < K_max ;          // pe defined in (12)
do
    if M then
        labReceive (W_i) ;          // receives information neccesary for the
         Selection step from W_i, ∀i
        if any of stoping condition is met then
            labSend (W_i) ;                                  // terminate workers
            output: f_min, x_min, pe, k, m;
            exit;
        else
            Perform the Selection step and split work between the workers;
            labSend (W_i) ;          // sends the instructions to W_i, ∀i, which
             hyper-rectangles should be subdivided
            if the worker has a deficit of POH stored inside it then
                labReceive (W_d) ;              // wait till other W_e share POH
            else if the worker has an excess of POH then
                labSend (W_e) ;           // share information of POH which is
                 excessive
            end
            Perform the Sampling and Division steps and store information inside
             the M memory block;
        end
    else
        labReceive (M) ;                        // receive instructions from M
        if not a termination message then
            if the worker has a deficit of POH stored inside it then
                labReceive (W_d) ;              // wait till other W_e share POH
            else if the worker has an excess of POH then
                labSend (W_e) ;           // share information of POH which is
                 excessive
            end
            Perform Sampling and Division steps and store information inside the
             W_i memory block;
            labSend (M) ;    // send neccesary data to M for the Selection
             step
        else
            exit;
        end
    end
end
```

Algorithm 2: Pseudo code of the pDIRECT-GLce algorithm

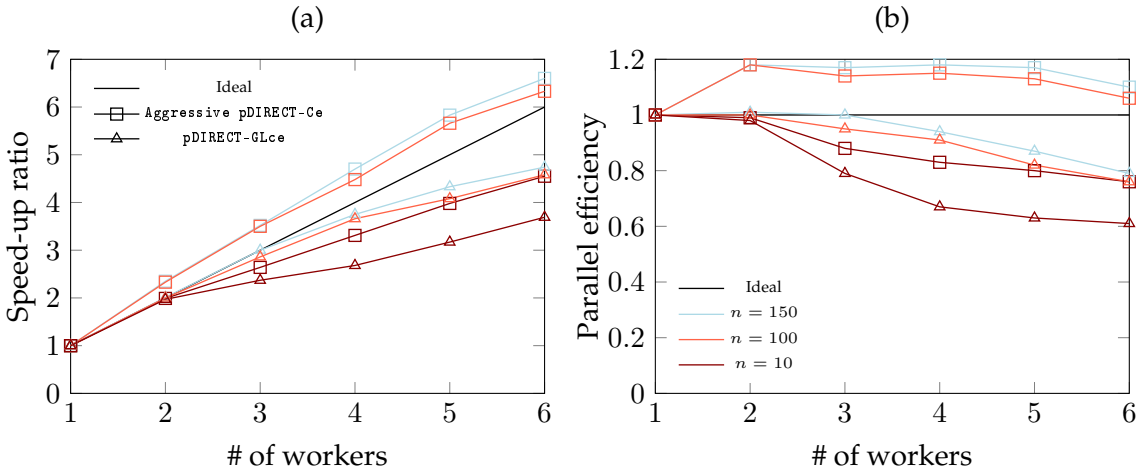

Figure 8: Speedup ratio (*left*) and parallel efficiency (*right*) on the *Michalewicz* test problem after $I_{max} = 30$ with different dimensions and $T_{delay} = 0$

ratio measures how much available processing power is being used and it is defined as

$$F(p) = \frac{S}{p}. \tag{11}$$

Because of the iterative nature of the DIRECT-type algorithms, the parallelization has not been widely investigated previously, and a comparison with the existing version is complicated. A massively parallel DIRECT algorithm, which was proposed in [HVSW09, HVWS08, HVWS09, HWR+02, HWS10], looks promising for expensive global optimization problems with large dimensionality ($n = 150$). But due to the usage of an aggressive selection scheme and additional step, which generates an extra function evaluation tasks for just keeping workers busy as much as possible, the existing strategy has some limitations. Massively parallel DIRECT is dividing more sub-optimal regions, which increases the number of function evaluations, which reduces the optimization effectiveness of an algorithm Fig. 12. Also, to overcome the high cost of later iteration solving more significant dimension problems Fig. 11 for the proposed algorithm may be needed a massively parallel supercomputer.

4.1 Box-constrained global optimization

First, we have focused on a performance test on box constrained global optimization test problems. We used the same five test problems, which are presented in Table 2 as were used in [HWS10]. The authors in [HWS10], introduced function evaluation cost T_ε to suit the different test purposes. In our experiments, the dimension was set to $n = 10, 25, 100, 150$, and several different values of artificial time delay (cost) $T_{delay} = 0$, 0.0001, 0.001, 0.01 in the evaluation of function values are used to make the test comparable ($T_\varepsilon = 0.1$ was used [HWS10]). Same as in [HWS10], the algorithms were stopped after $I_{max} = 30$ iterations.

The main factor that determines the efficiency of the parallel `pDIRECT-GLce` is the **Selection** step. The size of the potentially optimal hyper-rectangle set is the main criterion on how efficiently the work will be loaded to processors. Figs. 1 and 3 can be used to interpret the difference between the aggressive and the original `DIRECT-GL` selection schemes, where the number of selected POH latter the later scheme is approximately $10 - 34$ times smaller in every iteration, and the difference becomes even more significant when the dimension is growing. Moreover, the number of POH selected by `DIRECT-GL` per iteration has significant variance, and even in the late iteration, such a scheme does not ensure massive amounts of evaluations. The problem with such an unpredictable workload is that in some iterations, there is just not enough work happening, and the algorithm has a high parallel overhead. Even more, data needs to be transferred to processors after the **Selection** step, and the results grouped again afterward, and such communications cause additional overhead. Therefore, sometimes solving a problem in parallel can be slower than solving it sequentially, especially for lower dimensional test problems, where the algorithm cannot generate enough work to processors, and evaluations of function values are cheap.

Despite the difference in the cost of objective function evaluation, the number of hyper-rectangle diameters increases equally (solving the same dimensionality problems) using an "aggressive" selection scheme, therefore the algorithm uses the same number of evaluations per iteration. The only criteria that determine the efficiency of the `Aggressive pDIRECT-Ce` algorithm is the problem dimension and evaluation cost of the objective function. All problems, which are presented in Table 2 are cheap and the average cost of evaluation is approximately 10^{-6}. Fig. 8 plots speedup ratio and parallel efficiency of `Aggressive pDIRECT-Ce` and `pDIRECT-GLce` algorithms on the *Michalewicz* test problem with different dimensions $n = 10, 100, 150$ and original objective function evaluation cost $T_{delay} = 0$. Both algorithms showed similar results on small dimensions $n = 10$, and the achieved efficiency is ideal using two and three cores, more computational resources causing higher overhead, and efficiency is dropping for both algorithms. Larger dimension $n = 100, 150$ improves performance of the algorithms, and `Aggressive pDIRECT-Ce` showed ideal results with all cores, even though evaluations are cheap.

One of the critical parameters of the parallel performance is the cost of the objective function, in [HVSW09, HVWS08, HVWS09, HWR+02, HWS10] the authors used a fixed time cost of $T_{delay} = 10^{-1}$. For the simulation of expensive objective functions, we also used a small pause T_{delay} in evaluations, we choose three different values $10^{-4}, 10^{-3}, 10^{-2}$ for the algorithms on the *Michalewicz* test problem with $n = 10$. The results are presented in Fig. 9, selecting a higher value for T_{delay} increases parallel efficiency for both algorithms. `Aggressive pDIRECT-Ce` performs ideal on small dimensional test problems if the evaluation cost is larger then 10^{-3}. Performance analysis of `pDIRECT-GLce` is very similar if objective functions are more expensive.

Despite dimension and evaluation cost, `pDIRECT-GLce` has another dependence on

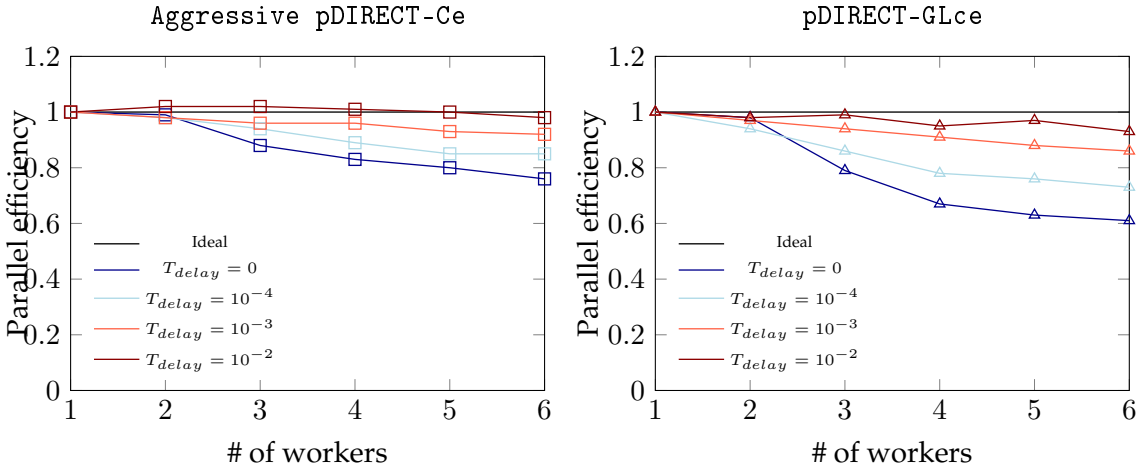

Figure 9: Parallel efficiency of the algorithms on the *Michalewicz* test problem after $I_{max} = 30$ with $n = 10$ and different values of T_{delay}

the objective function. The selection of potential optimal hyper-rectangles in the algorithm cannot be predicted. It depends on many criteria like the number of local and global minima, the type of function, etc. In Fig. 10, we showed performance results of `pDIRECT-GLce` solving test problems, which are presented in Table 2 with the same dimension $n = 150$. The results are very similar using two and three cores, but results begin to change using more computational cores. The algorithms differ in that `pDIRECT-GLce` is a single start and produces much slower growth of different size hyper-rectangles, which obliviously gives fewer evaluation tasks per iteration. Both differences give the slower start of an algorithm, and parallel efficiency is low at the beginning of optimization. In Fig. 11 an example of `pDIRECT-GLce` performance on the *Michalewicz* test problem is shown after $I_{max} = 100$ with $n = 25$ and $T_{delay} = 0$. In the initial 10 iterations, there is not enough work to all available cores, and parallel overheads dominate in the early stage of `pDIRECT-GLce`. But results are changing, when the number of iteration is increasing, the algorithm produces more different box diameters and as a result, more function evaluation tasks. Moreover, a large number of computational cores we use it results with a slower algorithm start. The parallel efficiency has a significant variance as the number of iterations grows, which can be explained as the result of different sizes of POH set in our selection scheme, and even in the late iterations, such a scheme can leave many cores to stand idle by producing only few POH in the iteration. Nevertheless, in long iterative progress, the total effectiveness ratio of the algorithm grows closer to the ideal parallelization.

The aggressive selection scheme seems more attractive for parallelization and can deliver better performance results comparing to any other selection scheme, including our proposed algorithm. But obviously, in terms of optimization effectiveness `pDIRECT-GLce` looks much more promising and can find a solution with less time and function evaluations, see Fig. 12. To evaluate optimization effectiveness, we took the same *Michalewicz*

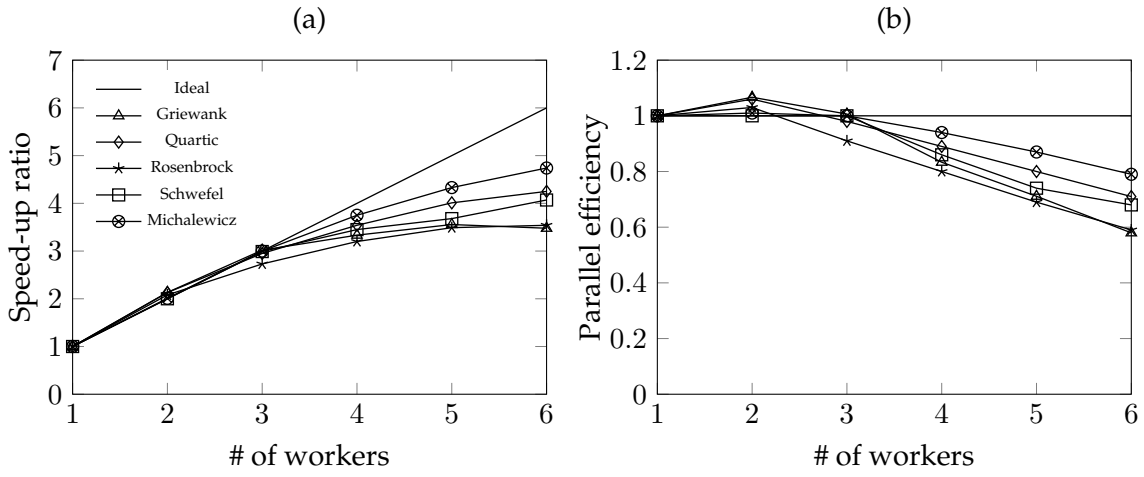

Figure 10: Speedup ratio (*left*) and parallel efficiency (*right*) onTable 2 test problems after $I_{max} = 30$ with $n = 150$ and $T_{delay} = 0$

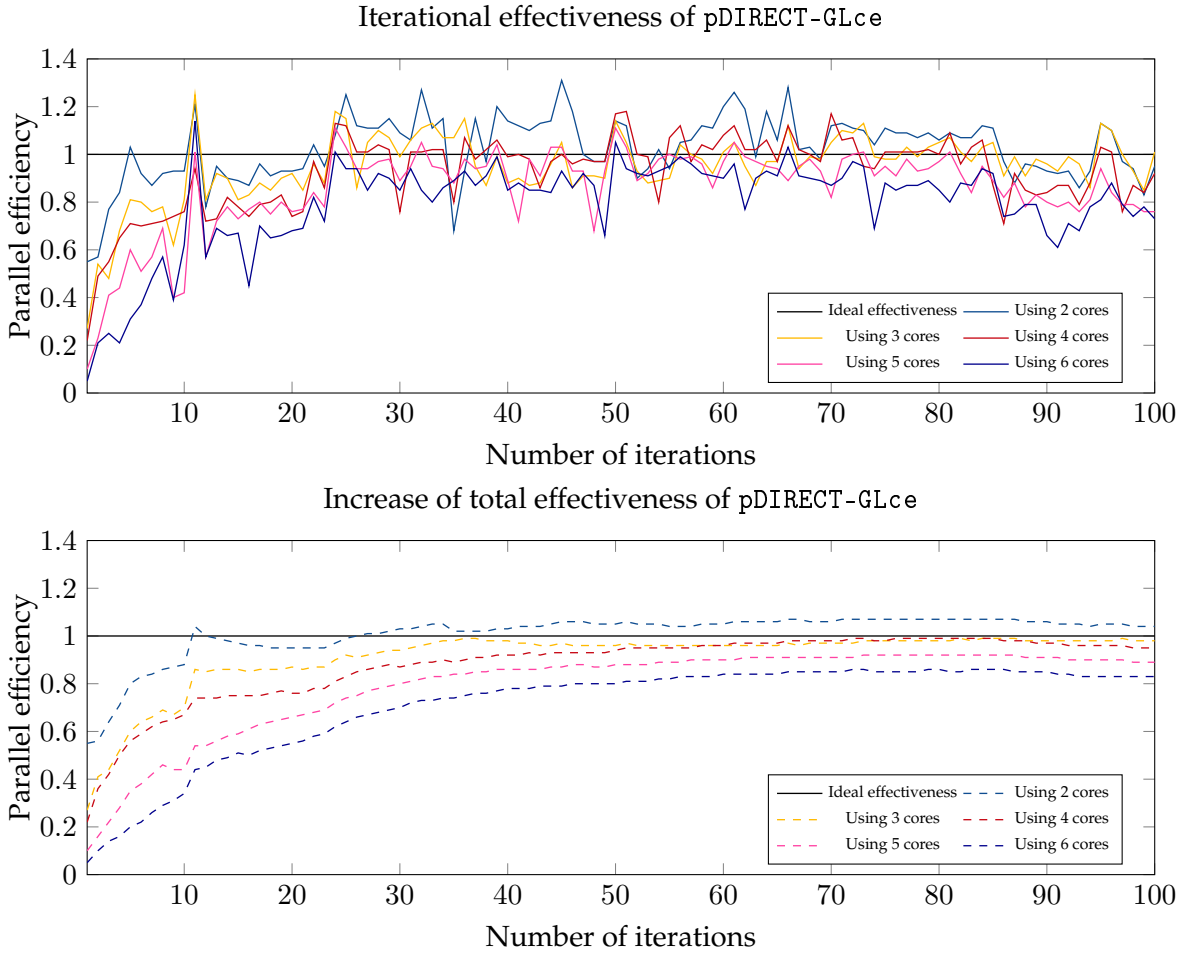

Figure 11: Variation of effectiveness ratio on the *Michalewicz* test problem on `pDIRECT-GLce` algorithm after $I_{max} = 100$ with $n = 25$ and $T_{delay} = 0$

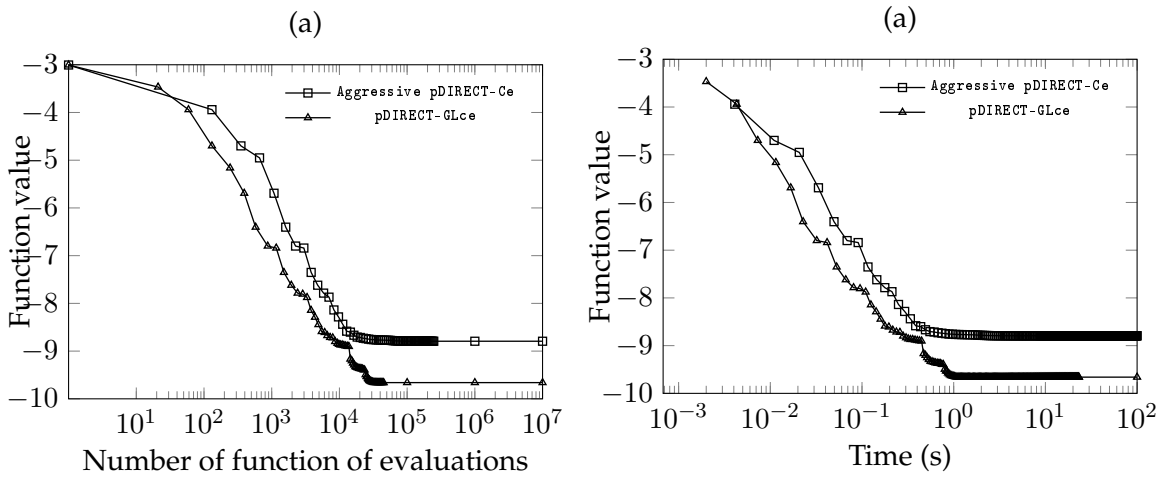

Figure 12: Comparison of function minimization on the 10-dimensional *Michalewicz* test problem. The number of function evaluations on the *left* and time in seconds on the *right*.

test problem with $n = 10$. The algorithms were stopped when the maximum number of function evaluations exceeds 10^7. On our chosen *Michalewicz* test problem, `Aggressive pDIRECT-Ce` was not able to locate the minima point even after 10^7 function evaluations, while `pDIRECT-GLce` was able to find it after $34,691$ evaluations with $\varepsilon_{\text{pe}} = 10^{-2}$. Moreover, the `pDIRECT-GLce` algorithm needs a shorter time to locate the minima.

4.2 General constrained global optimization

Unlike the massively parallel `DIRECT` algorithm, our proposed `DIRECT-GLce` can handle constraints. The constraint-handling technique raises the cost of our algorithm with extra calculations and constrains function evaluations. Additional required computations may increase the parallel performance of `DIRECT`-type algorithms.

Following, we focus on the problems with general constraints, which presented in Table 3. Many authors widely use all test problems in various experiments. Unfortunately, all test functions are of low dimensionality ($n = 2, \ldots, 24$) and cost of evaluations is cheap. Table 1 shows the cost of evaluations for all test problems, while the objective function cost is similar to Table 2 test problems which is approximately 10^{-6}, constraints functions are more expensive from 10 to 100 times. The performance results of the sequential `DIRECT-GLce` and `Aggressive DIRECT-Ce` algorithms are also presented in the same table. Since the global minimum f^* of all test problems are known, the algorithms stopped when the point $\bar{\mathbf{x}}$ generated such that the percent error

$$pe = 100\% \times \begin{cases} \frac{f(\bar{\mathbf{x}}) - f^*}{|f^*|}, & f^* \neq 0, \\ f(\bar{\mathbf{x}}), & f^* = 0, \end{cases} \tag{12}$$

is smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-2}$ or maximum number of function evaluations exceeds 10^5. The `DIRECT-GLce` algorithm to solve low dimensional test prob-

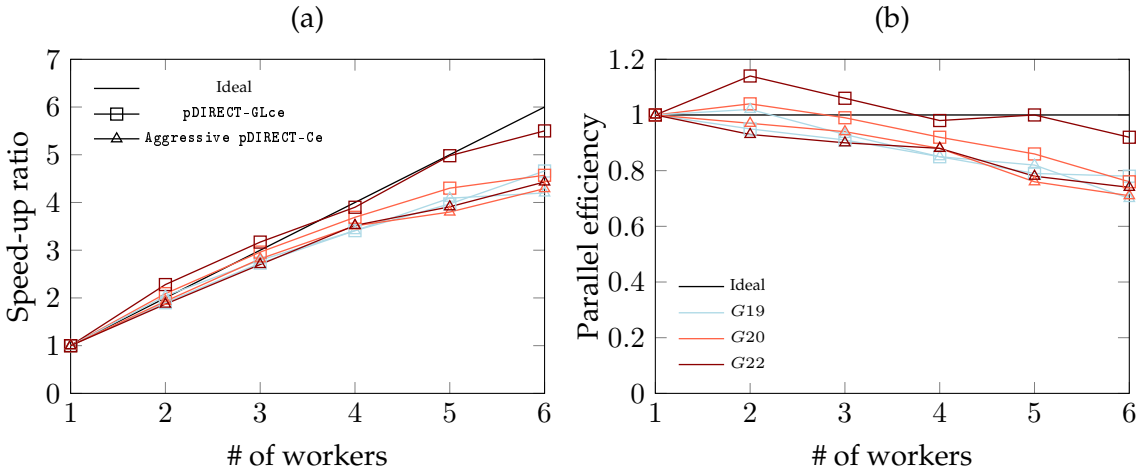

Figure 13: Speedup ratio (*left*) and parallel efficiency (*right*) on *G19, G20, G22* test problems after $I_{max} = 30$ with $T_{delay} = 0$

lems ($n \leq 10$) requires less than a second and any efficiency in parallelization on such problems cannot be expected. Evaluations per iteration in `Aggressive DIRECT-Ce` grows much faster comparing with our algorithm, but still for low dimensional test problems the algorithm spent only a few seconds. Evaluations per iteration is much larger in `Aggressive DIRECT-Ce` and better parallel performance can be expected. Looking from optimization effectiveness perspective, `DIRECT-GLce` can find better solution with less evaluations.

is smaller than the tolerance value $\varepsilon_{\text{pe}} = 10^{-2}$ or maximum number of function evaluations exceeds 10^5. The `DIRECT-GLce` algorithm to solve low dimensional test problems ($n \leq 10$) requires less than a second, and any efficiency in parallelization on such problems cannot be expected. Evaluations per iteration in `Aggressive DIRECT-Ce` grows much faster comparing with our algorithm, but still, for low dimensional test problems, the algorithm spent only a few seconds. Evaluations per iteration are much more extensive in `Aggressive DIRECT-Ce`, and better parallel performance can be expected. Looking from an optimization effectiveness perspective, `DIRECT-GLce` can find a better solution with fewer evaluations.

Table 1: Results of a sequential `DIRECT-GLce` and `Aggressive DIRECT-Ce` algorithms on Table 3 test problems and evaluation cost of objective f and constraints g functions.

			`DIRECT-GLce`				`Aggressive DIRECT-Ce`			
(#)	f_{cost}	g_{cost}	$ITER$	f_{eval}	f_{min}	T_1	$ITER$	f_{eval}	f_{min}	T_1
G01	3.61×10^{-6}	1.08×10^{-4}	94	100,059	$\mathbf{-12.04}$	17.13	27	107,299	-7.74	17.95
G02	9.61×10^{-6}	2.88×10^{-5}	70	101,319	$\mathbf{-0.24}$	8.45	17	110,805	-0.18	8.61
G03	1.58×10^{-6}	3.08×10^{-5}	220	100,317	$\mathbf{-0.99}$	9.10	34	104,769	-0.57	7.91
G04	1.28×10^{-6}	7.58×10^{-5}	98	20,111	$\mathbf{-30663.57}$	3.00	42	45,047	$\mathbf{-30663.57}$	6.20
G05	2.64×10^{-6}	7.08×10^{-5}	179	68,287	$\mathbf{5126.51}$	9.32	75	102,137	5241.52	12.61
G06	1.76×10^{-6}	2.74×10^{-5}	70	6,063	$\mathbf{-6961.17}$	0.87	48	10,497	$\mathbf{-6961.17}$	1.32
G07	1.33×10^{-6}	1.04×10^{-4}	131	100,285	$\mathbf{24.62}$	16.01	34	104,893	44.62	15.95
G08	1.71×10^{-6}	2.72×10^{-5}	24	959	$\mathbf{-0.09}$	0.26	21	2,009	$\mathbf{-0.09}$	0.37
G09	1.79×10^{-6}	5.56×10^{-5}	155	77,065	$\mathbf{680.69}$	8.58	49	103,359	680.83	10.77
G10	1.34×10^{-6}	7.70×10^{-5}	230	100,655	$\mathbf{7312.96}$	14.05	42	103,957	12461.11	13.15
G11	1.33×10^{-6}	1.45×10^{-5}	47	1,851	$\mathbf{0.75}$	0.36	33	5,445	$\mathbf{0.75}$	0.66
G12	1.33×10^{-6}	1.45×10^{-5}	6	173	$\mathbf{-1.00}$	0.06	6	249	$\mathbf{-1.00}$	0.07
G13	1.83×10^{-6}	4.39×10^{-5}	254	100,051	$\mathbf{0.62}$	11.52	62	102,279	0.99	10.50
G14	6.35×10^{-6}	3.97×10^{-5}	80	101,033	$\mathbf{-42.35}$	9.54	37	103,461	-42.03^a	6.34
G15	1.32×10^{-6}	2.74×10^{-5}	61	9,415	$\mathbf{961.71}$	1.11	98	100,843	964.29	9.06
G16	1.81×10^{-6}	5.05×10^{-4}	345	100,453	$\mathbf{-1.90}$	59.73	63	101,829	-1.74	59.74
G17	1.34×10^{-6}	3.95×10^{-5}	42	100,523	$\mathbf{8654.05}$	1.41	52	100,585	8746.17	9.24
G18	1.33×10^{-6}	1.66×10^{-5}	115	100,615	$\mathbf{-0.84}$	22.69	36	105,839	-0.83	22.79
G19	4.26×10^{-6}	8.34×10^{-5}	81	100,899	$\mathbf{113.01}$	13.50	22	103,711	222.67	13.27
G20	1.57×10^{-6}	2.25×10^{-4}	63	100,025	1.57^a	31.74	14	102,509	10.67^a	34.30
G21	1.29×10^{-6}	7.55×10^{-5}	79	101,773	500.00^a	13.04	45	104,417	500.00^a	13.40
G22	1.27×10^{-6}	2.65×10^{-4}	65	102,759	$10,000.00^a$	33.40	16	104,743	$16,666.66^a$	33.09
G23	2.19×10^{-6}	9.29×10^{-5}	72	102,355	-655.54^a	12.67	36	105,193	-354.51^a	14.24
G24	1.31×10^{-6}	2.82×10^{-5}	48	2,655	$\mathbf{-5.50}$	0.48	39	7,849	$\mathbf{-5.50}$	1.01

a – the final solution lies outside the feasible region

5 Conclusions

In Section 1, we have introduced the first parallel DIRECT-type algorithm for generally constrained global optimization problems. Due to iterative nature of DIRECT-type algorithms there exist only a few parallel DIRECT-type implementations, all them devoted for box-constrained optimization problems. Also we use the ideas of previous parallel implementation of the DIRECT algorithm, and proposing alternative modification of pDIRECT-GLce called Aggressive pDIRECT-Ce. The experimental results of two introduced versions: (single-start based pDIRECT-GLce and multi-start based Aggressive pDIRECT-Ce) revealed, that the parallel efficiency of Aggressive pDIRECT-Ce is better compared to pDIRECT-GLce when function evaluations are cheap. Since the "aggressive" selection scheme in Aggressive pDIRECT-Ce selects a larger number of hyper-rectangles per iteration, this assures more computationally intensive algorithmic iterations and better opportunities for parallelism. Moreover, a multi-start nature of the Aggressive pDIRECT-Ce algorithm ensures that all computational cores are busy from the beginning of the optimization process.

However, for the more expensive optimization problems, parallel efficiency of both algorithms is similar. Moreover, due to "aggressive" selection strategy, a large number non-potentially optimal hyper-rectangles is selected, therefore Aggressive pDIRECT-Ce wastes function evaluations on suboptimal regions and optimization effectiveness (based on the number of function evaluations) is significantly worse compared to pDIRECT-GLce. Finally, both introduced parallel versions pDIRECT-GLce and Aggressive pDIRECT-Ce preserve the determinism, where previous proposals have failed.

pDIRECT-GLce is a single-start algorithm, which gives only few evaluation tasks at the beginning, and parallel overhead is dominating in early stage of optimization. But in a long run, the parallel efficiency of the pDIRECT-GLce algorithm is getting closer to the ideal.

Motivated by the promising parallel performance, in the nearest future, we plan to develop an extension of the algorithm, by considering better-suited programming languages and frameworks for parallelization. Advanced data structures to better organize the local data and reduce communication overhead, and a hybrid CPU - GPU scheme will be considered.

Data access statement

Data underlying this article can be accessed on Zenodo at https://dx.doi.org/10.5281/zenodo.1218981, and used under the Creative Commons Attribution license.

References

[BBPW02] 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.

[BDLM12] 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.

[BFM10] 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.

[BG04] Lorenz T. Biegler and Ignacio E. Grossmann. Retrospective on optimization. *Computers & Chemical Engineering*, 28(8):1169–1192, 2004.

[BH99] Mattias Björkman and Kenneth Holmström. Global optimization using the DIRECT algorithm in Matlab. *Advanced Modeling and Optimization*, 1(2):17–37, 1999.

[BWG+00] 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.

[CEC08] 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.

[CGP+01] 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.

[CRF18] 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.

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[DS75] Laurence Charles Ward Dixon and Giorgio P Szegö. *Towards global optimisation: proceedings of a workshop at the University of Cagliari, Italy, October 1974*, volume 1. North Holland, 1975.

[Fin04] D. E. Finkel. MATLAB source code for DIRECT. http://www4.ncsu.edu/~ctk/Finkel_Direct/, 2004. Online; accessed: 2017-03-22.

[Fin05] D. E. Finkel. *Global Optimization with the* DIRECT *Algorithm*. PhD thesis, North Carolina State University, 2005.

[FK06] D. E. Finkel and C. T. Kelley. Additive scaling and the DIRECT algorithm. *Journal of Global Optimization*, 36(4):597–608, 2006.

[FK09] A. I. J. Forrester and A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1):50–79, 2009.

[FL02] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239—-269, 2002.

[Fle87] R. Fletcher. *Practical Methods of Optimization*. John and Sons Chichester, 2nd edition, 1987.

[Flo99] Christodoulos A Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Springer US, 1999.

[Gab01] J. M. Gablonsky. *Modifications of the DIRECT Algorithm*. PhD thesis, North Carolina State University, 2001.

[GK01] J. M. Gablonsky and C. T. Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21(1):27–37, 2001.

[Hed05] 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.

[HPT95] R. Horst, P. M. Pardalos, and N. V. Thoai. *Introduction to Global Optimization*. Nonconvex Optimization and Its Application. Kluwer Academic Publishers, 1995.

[HSS+93] J. He, M. Sosonkina, C. A. Shaffer, J. J. Tyson, L. T. Watson, J. W. Zwolak, and W. Hall. A hierarchical parallel scheme for global parameter estimation in systems biology. *Direct*, 1993.

[HVSW09] 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.

[HVWS08] 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.

[HVWS09] 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.

[HWR+02] 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.

[HWS10] 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.

[Jon01] 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.

[JPS93] 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.

[KPS03] D. E. Kvasov, C. Pizzuti, and Ya. D. Sergeyev. Local tuning and partition strategies for diagonal GO methods. *Numerische Mathematik*, 94(1):93–106, 2003.

[KWRG11] 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.

[LC14] Qunfeng Liu and Wanyou Cheng. A modified DIRECT algorithm with bilevel partition. *Journal of Global Optimization*, 60(3):483–499, 2014.

[LLP10] 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.

[LLP16] 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.

[LXC+17] 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.

[LYZZ17] 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.

[LZY15] 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.

[MPR+17] 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.

[MW09] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[NLH17] J. Na, Y. Lim, and C. Han. A modified DIRECT algorithm for hidden constraints in an LNG process optimization. *Energy*, page 488–500, 2017.

[PCŽ18] 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.

[Pin96] 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.

[PLL+16] 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.

[PLR10] 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.

[PSKŽ14] 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.

[PWA+08] 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.

[PŽ13] Remigijus Paulavičius and Julius Žilinskas. Simplicial Lipschitz optimization without the Lipschitz constant. *Journal of Global Optimization*, 59(1):23–40, 2013.

[PŽ14] Remigijus Paulavičius and Julius Žilinskas. *Simplicial Global Optimization*. SpringerBriefs in Optimization. Springer New York, New York, NY, 2014.

[PŽ16] Remigijus Paulavičius and Julius Žilinskas. Advantages of simplicial partitioning for Lipschitz optimization problems with linear constraints. *Optimization Letters*, 10(2):237–246, 2016.

[PŽHC13] 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.

[Reg11] 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.

[Reg14] 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.

[RL03] 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.

[Ser98] Yaroslav D Sergeyev. On convergence of "divide the best" global optimization algorithms. *Optimization*, 44(3):303–325, 1998.

[SHL+05] 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.

[SK06] 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.

[SK08] Ya. D. Sergeyev and D. E. Kvasov. *Diagonal Global Optimization Methods*. FizMatLit, Moscow, 2008. In Russian.

[SK17] Yaroslav D Sergeyev and Dmitri E Kvasov. *Deterministic Global Optimization: An Introduction to the Diagonal Approach*. SpringerBriefs in Optimization. Springer, 2017.

[SP18] Linas Stripinis and Remigijus Paulavičius. DIRECTLib – a library of global optimization problems for DIRECT-type methods, v1.1, 2018.

[SPŽ18] 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.

[SPŽ19] 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.

[SW10a] S. Shan and G. G. Wang. Metamodeling for high dimensional simulation-based design problems. *Journal of Mechanical Design*, 132(5):051009, 2010.

[SW10b] 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.

[VV09] 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.

[WB01] L. T. Watson and C. A. Baker. A fully-distributed parallel global search algorithm. engineering computations. *Engineering Computations*, 18(1/2):155–169, 2001.

[ZTW05] 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.

Appendixes

Appendix Nr. 1.
Test functions

Table 2: Key characteristics of the box constrained global optimization test problems

(#)	Label	Source	Variable bounds (D)	No. of local	Optimum (f^*)
1	Griewank	[Hed05]	$[-20, 30]^2$	multimodal	0.0
2	Quartic	[Hed05]	$[-2, 3]^n$	unimodal	$-29.816n$
3	Rosenbrock	[Hed05]	$[-2.048, 2.048]^n$	unimodal	0.0
4	Schwefel	[Hed05]	$[-500, 500]^n$	unimodal	0.0
5	Michalewicz	[Hed05]	$[0, \pi]^n$	n!	xx

Table 3: Key characteristics of the general constrained global optimization test problems

(#)	Label	Source	n	Number of C.	C. type	Variable bounds (D)	Optimum (f^*)
1	G01	[SHL+05]	13	9	L	$[0,10]^9 \times [0,10^2]^3 \times [0,10]$	-15.0000
2	G02	[SHL+05]	20	2	NL	$[0,10]^n$	-0.8036
3	G03	[SHL+05]	10	1	NL	$[0,10]^n$	-1.0005
4	G04	[SHL+05]	5	6	NL	$[78,102] \times [33,45] \times [27,45]^3$	30665.5386
5	G05	[SHL+05]	4	5	NL	$[10,1.2 \times 10^3]^2 \times [-0.55,0.55]^2$	5126.4967
6	G06	[SHL+05]	2	2	NL	$[13,10^2] \times [0,10^2]$	-6961.8138
7	G07	[SHL+05]	10	8	NL	$[-10,10]^n$	24.3062
8	G08	[SHL+05]	2	2	NL	$[0,10]^n$	-0.0958
9	G09	[SHL+05]	7	4	NL	$[-10,10]^n$	680.6300
10	G10	[SHL+05]	8	6	NL	$[10^2,10^4] \times [10^3,10^4]^2 \times [10,10^3]^5$	7049.2480
11	G11	[SHL+05]	2	1	NL	$[-1,1]^n$	0.7499
12	G12	[SHL+05]	3	1	NL	$[0.2,10]^n$	-1.0000
13	G13	[SHL+05]	5	3	NL	$[-2.3,2.3]^2 \times [-3.2,3.2]^3$	0.0539
14	G14	[SHL+05]	10	3	L	$[0,10]^n$	-47.7648
15	G15	[SHL+05]	3	2	NL	$[0,10]^n$	961.7150
16	G16	[SHL+05]	5	38	NL	$[704.4148,906.3855] \times [68.6,288.88] \times [0,134.75] \times [193,287.0966] \times [25,84.1988]$	-1.9051
17	G17	[SHL+05]	6	4	NL	$[0,4 \times 10^2] \times [0,10^3] \times [340,420]^2 \times [-10^3,10^3] \times [0,0.5236]$	8853.5396
18	G18	[SHL+05]	9	13	NL	$[0,10]^8 \times [0,20]$	-0.8660
19	G19	[SHL+05]	15	5	NL	$[0,10]^n$	32.6555
20	G20	[SHL+05]	24	18	NL	$[0,10]^n$	–
21	G21	[SHL+05]	7	6	NL	$[0,10^3] \times [0,40]^2 \times [10^2,3 \times 10^2] \times [6.3,6.7] \times [5.9,6.4] \times [4.5,6.25]$	193.7245
22	G22	[SHL+05]	22	20	NL	$[0,2 \times 10^4] \times [0,10^6]^3 \times [0,4 \times 10^7] \times [10^2,299.99] \times [10^2,399.99] \times [100.01,3 \times 10^2] \times [10^2,4 \times 10^2] \times [10^2,6 \times 10^2] \times [0,5 \times 10^2]^3 \times [10^{-2},3 \times 10^2] \times [10^{-2},4 \times 10^2] \times [-4.7,6.25]^5$	236.4309
23	G23	[SHL+05]	9	6	NL	$[0,3 \times 10^2]^2 \times [0,10^2] \times [0,2 \times 10^2] \times [0,10^2] \times [0,3 \times 10^2] \times [0,10^2] \times [0,2 \times 10^2] \times [10^{-2},0.03]$	-400.0551
20	G24	[SHL+05]	2	2	NL	$[0,3] \times [0,4]$	-5.5080

– no feasible solution is found so far