Parallel versions of the mesh adaptive direct search algorithm

Sébastien Le Digabel[2,3], Antoine Lesage-Landry[1,3], Samuel Mendoza[1,3], Christophe Tribes[2,3]

[1]Department of Electrical Engineering, Polytechnique Montréal, 2500 Chemin de Polytechnique, Montréal, H3T 1J4, Québec, Canada.
[2]Department of Mathematics and Industrial Engineering, Polytechnique Montréal, 2500 Chemin de Polytechnique, Montréal, H3T 1J4, Québec, Canada.
[3]Group for Research in Decision Analysis (GERAD), 3000 Chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7, Québec, Canada.

Contributing authors: sebastien.le-digabel@polymtl.ca; antoine.lesage-landry@polymtl.ca; samuel.mendoza@polymtl.ca; christophe.tribes@polymtl.ca;

Abstract

This work surveys the different parallel variants of the mesh adaptive direct search (MADS) algorithm for constrained blackbox optimization. These problems can inherently imply high computational costs due to the possible large number of variables and multi-modality of the search space. In addition, the potential time-intensive nature and time heterogeneity of the blackboxes defining the problem prompts the need for efficient implementations. Parallelism emerges as an actionable solution to mitigate computation time, as modern computer systems rely on multi-core architecture. The reviewed methods employ diverse levels of parallelism and distinct parallel strategies to effectively tackle each aspect outlined above. The manuscript details the practical implementations, provides computational results, and offers insights into the advantages and limitations of each MADS parallel method.

Keywords: Blackbox optimization, derivative-free optimization, mesh adaptive direct search, parallel computing.

1 Introduction

This work considers optimization problems of the form

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}), \tag{$\mathbb{P}$}$$

where $\Omega = \{\mathbf{x} \in \mathcal{X} \mid c_j(\mathbf{x}) \leq 0, j \in \mathcal{J} = \{1, 2, \ldots, m\}\} \subseteq \mathbb{R}^n$, $m, n \in \mathbb{N}$, is the feasible set and $\mathcal{X}$ is a subset of $\mathbb{R}^n$, typically defined by bound constraints. The functions $f : \mathcal{X} \to \mathbb{R} \cup \{\infty\}$ and $c_j : \mathcal{X} \to \mathbb{R} \cup \{\infty\}$, $j \in \mathcal{J}$, are typically evaluated through a computer simulation with no available derivatives, and is considered as a blackbox. We consider that evaluating such a blackbox is time-consuming and that only a limited budget of evaluations is available. We further assume that the blackbox may be subject to heterogeneity in the time required to evaluate a point, as illustrated in Figure 1.

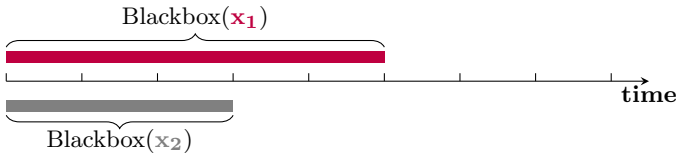

Figure 1: Illustration of a time-heterogeneous blackbox.

Problem ($\mathbb{P}$) is a blackbox optimization (BBO) problem, for which derivative-free optimization (DFO) algorithms are considered. These techniques are described in [1, 2], and some practical applications are illustrated in [3].

With DFO methods, a rule of thumb is to allow a budget of the order of $1{,}000n$ evaluations to obtain satisfactory results. This is not conceivable when blackboxes are time-consuming because this process would lead to impractical resolution times. The situation worsens when dealing with large-scale problems, typically with more than 50 variables, as the evaluations budget escalates significantly.

Parallel computing appears as a solution because most of computational resources exploits multi-core architectures available through various levels, e.g., workstations, servers, and supercomputers. In addition, with the end of Dennard scaling, parallelization has become much more important than before, as the alternative of increasing processor clock speed is no longer possible due to heat dissipation and power consumption issues.

An obvious way of using parallelism is to "open" the blackbox and to modify it to exploit the available parallel resources. However, this is not always possible due to many reasons: The source code of the blackbox may be unavailable, too complicated, too old, etc. Hence, in this work, we focus on the parallelization of DFO algorithms, and in particular direct search (DS) methods.

In an effort to take advantage from modern multi-core infrastructures, multiple DFO methods have been extended to leverage parallel computations through their resolution process. In particular, DFO methods can benefit from parallelism to reduce the time resolution, improve solution quality, and increase the size of BBO problems

that can be practically solved. Specifically, DS methods, a branch of DFO methods, provides a framework for the effective use of parallel computing.

Throughout this work, we consider the mesh adaptive direct search (MADS) [4] algorithm and its C++ implementation, NOMAD version 3 [5], and we review its different parallel versions. We measure the performance of these parallel algorithms by introducing metrics for benchmarking in a context of parallel computing, and we give recommendations to guide users on what kind of parallelism should be used depending on the different aspects of the problem at hand. This study is conducted with MADS, but remains applicable to other DS methods.

This work is structured as follows: Section 2 reviews the literature and provides an overview of parallel methods for BBO, Section 3 describes MADS, Section 4 presents its parallel versions, and Section 5 reports computational experiments with recommendations. Finally, concluding remarks are provided in Section 6.

2 Parallelism in BBO

A guiding principle for the efficient use of parallelization to solve ($\mathbb{P}$) is to consider that the time spent to evaluate the objective or constraint functions is much higher than the time required by an optimization algorithm to generate trial points. Hence, the parallelization that is the most likely to be beneficial, i.e., which yields a large reduction in overall execution time, consists of performing several concurrent evaluations of the blackbox and/or to perform an internal parallelization of the blackbox [6]. As mentioned above, we consider that the latter is not available in this work. Parallelization strategies, i.e., functional parallelism, domain decomposition, multi-search parallelism, and the hybridization of mutually compatible aforementioned strategies, are described in details in [7]. It also provides a literature review on integrative framework for parallel computational optimization in operations research. While these strategies are formulated in an operations research context, the scope of applicability can be straightforwardly extended to other optimization algorithm classes, such as DS methods, as described in [8].

Parallelism in BBO and specifically for DS methods, is a longstanding topic of interest. The first DS methods to employ parallelism with proofs of convergence are based on the generalized pattern search (GPS) [9] and the generating set search (GSS) [10] algorithms. The resulting method is the asynchronous parallel pattern search [11–15]. Implementation examples can be found in the APPSPACK [16] and HOPSPACK [17] solvers. Reference [18] combines instances of MADS with different line search methods which are run in parallel to accelerate the resolution process. A specific parallel search for MADS is later introduced in the technical report [19]. Other DS methods include the Nelder-Mead [20] whose first parallel version is described in [21], and revisited more recently in [22]. The DIRECT algorithm [23] is hybridized with GSS in [24] and its parallelized versions are analyzed in [25–28]. Other DFO methods include model-based algorithms utilizing a trust region and a model such as polynomial or radial basis functions. In this case, parallelization is not as inherent as for DS methods. Nonetheless, examples of parallel DFO can be found in [29–34]. Parallel surrogate-assisted methods are also described in [35–37]. Bayesian optimization methods based on the efficient

global optimization framework [38] are parallelized, for example, in [39–41]. There exists also multiple other parallel heuristics such as [42]. More generally, the survey [43] reviews different kinds of parallel heuristics.

Finally, parallel DS methods are employed in specific applications such as hyperparameters tuning [44], local optimum enumeration [45], multiobjective optimization [46], and small (or moderate) dimensional engineering design problems driven by computationally expensive simulations, where large-batch parallel sampling can outperform more elaborate techniques [47, 48].

3 MADS: Mesh Adaptive Direct Search

MADS is a direct search method introduced in [4] and refined in several articles, including its last major evolution in [49]. The version given in Algorithm 1 corresponds to the one provided in [1].

Blackbox evaluations occur at two different instances: The search and the poll. The points to be evaluated are gathered in two sets $\mathcal{S}^k \subset \mathbb{R}^n$ and $\mathcal{P}^k \subset \mathbb{R}^n$ and are evaluated opportunistically, i.e., evaluations are interrupted as soon as a new best point is found. When evaluations are computationally costly, as in BBO, this strategy is effective because it can reduce the number of blackbox evaluations which is usually subject to a budget constraint. In a sequential programming environment, opportunism occurs at an individual point level. Thus, a minimum of one evaluation can be computed in order to terminate the current iteration if a simple decrease condition is reached by an evaluation point. In contrast, in a parallel environment, evaluations are dispatched concurrently to several computing resources. The minimum number of evaluations that will be performed before the premature termination of an iteration is, therefore, equal to the number of computing resources being used in parallel. As a result, the evaluation budget is spent more quickly. This is why a computation time budget is a better-suited constraint in a parallel environment.

MADS relies on a special structure called the mesh: at iteration k, it is a discretization of the space of variables defined by

$$\mathcal{M}^k = \left\{ \mathbf{x}^k + \delta^k \mathbf{D}\mathbf{y} \;\middle|\; \mathbf{y} \in \mathbb{N}^p \right\} \subset \mathbb{R}^n, \tag{1}$$

where $\delta^k > 0$ is the mesh size parameter, $\mathbf{D} \in \mathbb{R}^{n \times p}$ is a positive spanning matrix, and $p \in \mathbb{N}$. MADS selects the candidate trial points on the mesh.

The notion of locality is parameterized by $\Delta^k > 0$, the frame size parameter, which defines the maximal distance between the incumbent solution $\mathbf{x}^k$ and the poll points $\mathbf{x} \in \mathcal{P}^k$ with

$$\|\mathbf{x} - \mathbf{x}^k\|_\infty \leq b\Delta^k,$$

where $b = \max_{d' \in \mathbb{D}} \|d'\|_\infty$ and $\mathbb{D}$ is the set of columns of matrix $\mathbf{D}$.

At every iteration k, δ^k and Δ^k are resized via $\tau \in (0, 1) \cap \mathbb{Q}$, the mesh size adjustment parameter, such that $0 < \delta^k \leq \Delta^k$ for all k. By doing so, a variety of directions that grow densely in the unit sphere is guaranteed to be generated.

All these conditions lead to the poll step being rigidly defined, but they ensure the convergence of the method. As detailed in [1], upon some assumptions about the problem, global convergence of MADS to a local optimum is guaranteed.

Algorithm 1: MADS with Extreme Barrier

Initialization
  Set $\Delta^0 > 0$ and $\mathbf{x}^0 \in \Omega$.
  Start the w worker processes for evaluation.
  Evaluate $\{f(\mathbf{x}^0), c_1(\mathbf{x}^0), c_2(\mathbf{x}^0), \ldots, c_m(\mathbf{x}^0)\}$ and initialize cache $\mathcal{C}$.
for $k = 0, 1, \ldots$ **do**
  Mesh (construction of $\mathcal{M}^k$ (1))
    Set $\mathbf{D} \in \mathbb{R}^{n \times p}$ a positive spanning matrix.
    Set $\delta^k = \min\{\Delta^k, (\Delta^k)^2\}$.
  Search (optional)
    Build the search set $\mathcal{S}^k \subset \mathcal{M}^k$.
    Evaluate $\{f(\mathbf{x}), c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_m(\mathbf{x})\}$ for all $\mathbf{x} \in \mathcal{S}^k$ (opportunistically)
    if $f_\Omega(\mathbf{x}) < f_\Omega(\mathbf{x}^k)$ for some $\mathbf{x} \in \mathcal{S}^k$ **then**
      set $\mathbf{x}^{k+1} \leftarrow \mathbf{x}$, $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$
      go to **Termination**.
    end
  Cache Search (optional and only for $k > 0$)
    if $f_\Omega(\mathbf{x}^*) < f_\Omega(\mathbf{x}^k)$ for some $\mathbf{x}^* \in \mathcal{C}$ **then**
      set $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^*$, $\Delta^{k+1} \leftarrow \tau\Delta^*$
      go to **Termination**.
    end
  Poll
    Build the poll set $\mathcal{P}^k \subset \mathcal{M}^k$.
    Perform evaluations and compute success
    if *Success* **then**
      go to **Termination**.
    end
    Set $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k$, $\Delta^{k+1} \leftarrow \tau\Delta^k$ (no success).
  Termination
    if *Stopping criteria* **then**
      End the algorithm.
      End the w workers.
    end
end

The optional search step is more flexible as it allows any strategy to be used for the construction of the sets $\mathcal{S}^k$. It can be generic or specific to a problem, and only needs to generate points on the current mesh. Examples of search strategies can be found in [50–52].

MADS is often run with a cache to provide a history of the evaluations that have been carried out. The main purpose of the cache is to ascertain whether the point

has been previously evaluated, thus preventing redundant evaluations. Optionally, the cache can be used during a special search step only when trial points are added to the cache asynchronously (see Section 4.1.2). If a point $\mathbf{x}^*$ in the cache is better than $\mathbf{x}^k$, it would be used as a new incumbent to start an iteration, and the corresponding frame size Δ^* is adopted. This step is disabled in a sequential environment.

In practice, the stopping criteria indicated in the algorithm can be multiple. Typically, they include a maximum number of blackbox evaluations, a time budget, and/or a minimal mesh or frame size.

MADS provides two distinct approaches to deal with inequality constraints. First, a straightforward extension with extreme barrier (EB), which simply defines $f_\Omega(\mathbf{x}) = f(\mathbf{x})$ if $\mathbf{x} \in \Omega$, and $f_\Omega(\mathbf{x}) = \infty$ otherwise. Algorithm 1 is based on the EB. The second approach is referred to as the progressive barrier (PB) and is defined in [53]. It considers a function $h(\mathbf{x})$ that measures the constraint violations at $\mathbf{x}$ and ranks the iterates in a graph (called a filter) where compromises in terms of f (objective) and h (constraints) are defined.

Finally, MADS is available in the open-source solver `NOMAD` [5] at `https://www.gerad.ca/en/software/nomad/`. This solver will be used for our numerical comparisons.

4 Parallel versions of MADS

This section describes the four parallel methods of MADS, namely pMADS-S for "Parallel MADS-Synchronous" (Section 4.1.1), pMADS-A for "Parallel MADS-Asynchronous" (Section 4.1.2), COOP-MADS for "Cooperative MADS" (Section 4.2), and PSD-MADS for "Parallel Space Decomposition MADS" (Section 4.3).

These methods are defined such that the convergence properties of MADS also hold. While PSD-MADS has been the subject of [54], the other three methods have never been described or benchmarked in the literature. These methods follow the master-worker paradigm typically implemented with the `Message Passing Interface` (`MPI`) library [55]. Following the parallel strategies described in Section 2, pMADS-A and pMADS-S are classified as functional parallelism, while PSD-MADS is classified as domain decomposition parallelism. Lastly, COOP-MADS is classified as multi-search parallelism.

4.1 pMADS

The first two parallel versions of MADS are both labelled pMADS and follow Algorithm 1. In these methods, the evaluation of trial points $\mathcal{P}^k$ is performed in parallel in a master-worker fashion (see Figure 3). In doing so, the master is responsible for sending evaluation points to the workers, who are then tasked with evaluating them (see Figure 2). The two versions of pMADS are the synchronous variant (pMADS-S) and the asynchronous variant (pMADS-A). They differ in the management of the evaluations by the w workers (shaded area in Algorithm 1). Starting and ending the workers are managed by the master process when the algorithm ends.

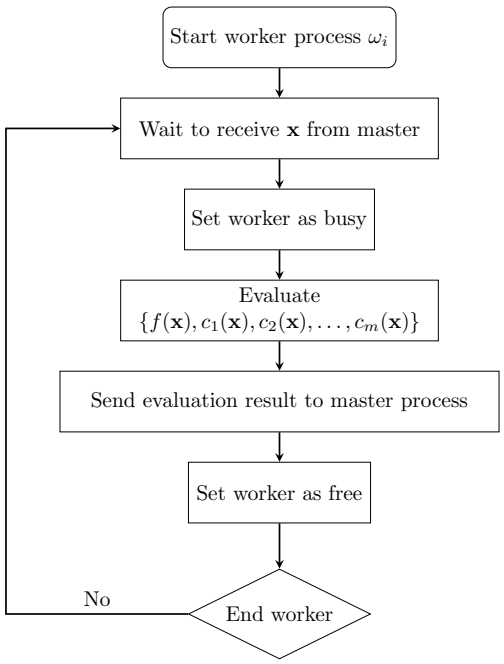

Figure 2: Flow chart of a single worker process for pMADS.

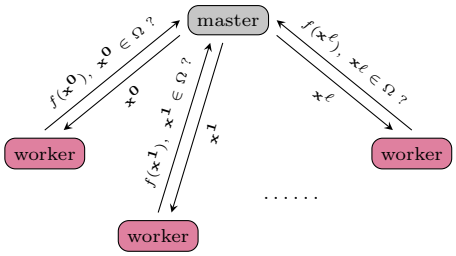

Figure 3: Master-worker communication organization for pMADS.

4.1.1 pMADS-S

pMADS-S has a deterministic behaviour, but at the cost of introducing a synchronization barrier which restricts the master to end an iteration only when all evaluations in progress are terminated. Consequently, this leads to idle computing resources at each iteration and hence to a loss in efficiency. The synchronization barrier eliminates the need for the cache search step.

An opportunistic block evaluation approach, using a block of size w, is achieved by executing the **Termination** step upon success, as shown in Figure 4.

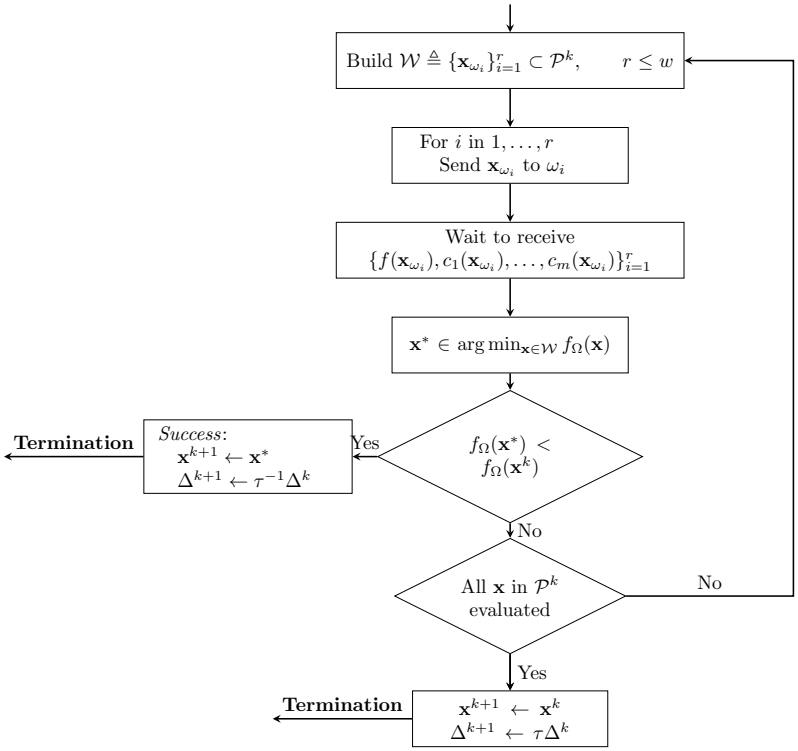

Figure 4: Master process flow chart of pMADS-S to manage parallel evaluations by the w parallel workers allocated to `NOMAD`, where ω_i represents their respective indices. Details the shaded area in Poll step of Algorithm 1.

4.1.2 pMADS-A

As mentioned in [5], pMADS-A is a simplified version of the asynchronous parallel pattern search algorithm (`APPSPACK`) [16]. The master is permitted to terminate an iteration and to generate new trial points as soon as a new success is obtained by a worker, even if evaluations are still in progress on other workers.

In the `NOMAD` implementation of pMADS-A, evaluations currently in progress are never cancelled (see Figure 5). But once all $\mathbf{x}$ in $\mathcal{P}^k$ have been sent to workers for evaluation and no success has been obtained among the received evaluations, the current incumbent solution and the frame size are updated. In the event that an evaluation that took place in an iteration prior to the current one finds the best solution, `NOMAD` backtracks and sets this solution as the incumbent one during a cache search step. Therefore, there is no synchronization barrier between `MPI` processes during the Poll step. This makes this mode effective for time-varying blackboxes because no `MPI` processes need to wait for the most time-consuming one to finish evaluating its solution. It is imperative to avoid idling computing resources, as this leads to poor overall performance and induces bottlenecks in a parallel programming context. This assertion is elaborated upon in Section 5 where numerical results are provided and discussed to

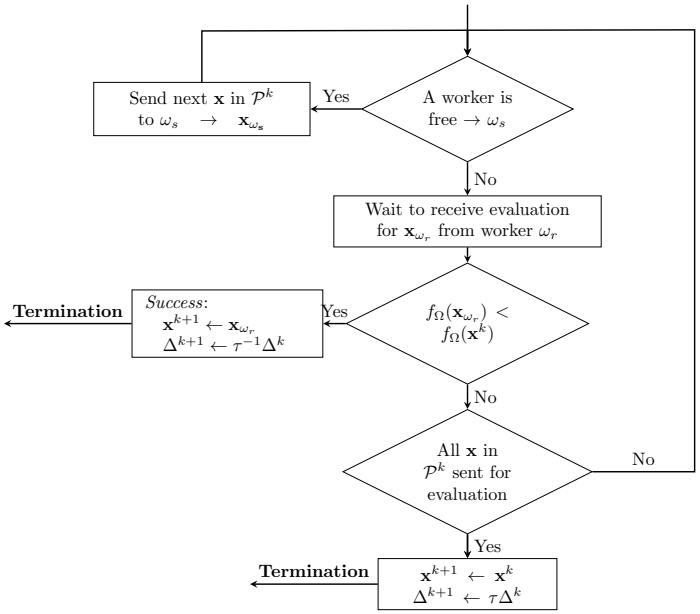

Figure 5: Flow chart of the master process for pMads-A. Details the shaded area in Poll step of Algorithm 1.

support this claim. The main downside of the asynchronous mode is pMADS-A's non-deterministic behaviour, which limits the reproducibility of the results. We remark that pMADS-A is the default mode used in `NOMAD` when parallelism is enabled. Also, one can note that there still exists a synchronization barrier between the Search and Poll steps that can affect the algorithm effectiveness. Notice that, to account for the possibility that evaluations may still be in progress during the **Termination** step, the stopping criteria of Algorithm 1 has to check if all evaluations are terminated.

4.2 COOP-MADS

To accelerate the exploration of $\mathcal{X}$ and to prevent a fast convergence of MADS to a local optimum, multiple paths can be generated by running c different instances of MADS in parallel. This can be particularly beneficial for multi-modal problems where the aim is to converge at a global optimum. Each MADS instance is initialized with a unique random seed, which leads to a varied selection of directions from OrthoMADS [56]. Hence, it yields distinct behavioural patterns in trial points generation. These MADS instances operate independently and without any synchronization mechanisms between them. To balance the evaluation effort, a dynamic distribution of the global evaluation budget is implemented between instances. Note that a static budget allocation which consists in distributing evaluations equally between instances often results in the resource under-utilization due to the varying convergence rates of individual instances.

In COOP-MADS, the only information exchange between MADS instances is via the shared cache to prevent redundant evaluations in an instance and between the instances. The optional cache search could also be enabled to periodically synchronize the incumbent solution between the instances. This option was not used in this work. Also, for simplicity, each MADS instance runs as a single process to perform the algorithm and the evaluations. The master-worker paradigm used in pMADS is not considered for COOP-MADS.

4.3 PSD-MADS

Solving high-dimensional BBO problems is a computationally demanding task. PSD-MADS [54] stands out as an algorithm tailored specifically to large-scale BBO problems. It specifically leverages a parallel space decomposition approach to increase MADS' scalability. The core concept of PSD-MADS is centred around the manager process generating optimization subproblems $\mathbb{P}_p$ by randomly selecting variable subspaces $\mathcal{N}_p$ from the original extensive variable space $\mathcal{N}$, i.e., $\mathcal{N}_p \subseteq \mathcal{N}$. These subproblems are then distributed among workers by the manager. Each worker applies MADS on their assigned subproblem and aims to improve the shared incumbent solution $\mathbf{x}^*$. Upon completing its task a worker receives a new subproblem $\mathbb{P}_p$ from the manager. A subproblem is defined by the incumbent solution given as an initial point, the initial and minimum mesh sizes Δ_0^p and $\Delta_{\min}^p$ and the fixed variables in $\mathcal{N}$ to define the variable subspace $\mathcal{N}_p$.

Note that the PSD-MADS operates asynchronously as no synchronization step is present. The algorithm introduces four roles, namely, the manager, the pollster, the regular worker, and the cache server. The pollster is a special worker which solves the original problem ($\mathbb{P}$) with a single-poll direction MADS algorithm instead of the conventional $2n$ or $n+1$ poll directions. The pollster is crucial to ensure the convergence of PSD-MADS. The manager assumes the responsibility of selecting the optimization subproblems $\mathbb{P}_p$ and performs adjustments to the main mesh and continuously updates the value of the incumbent solution. The cache server stores all evaluated points to avoid unnecessary, multiple, and expensive function evaluations. Finally, a regular worker is a MADS instance that solves a subproblem $\mathbb{P}_p$.

5 Computational study

In this section, we conduct numerical experiments to evaluate the performance of the described algorithms. We use different sets of test problems each of which are targeting specific blackbox characteristics, to make recommendations on when an algorithm should be used. All the experiments are run on Intel Core i7-12700 @ 2.10GHz processors using Version `3.9.1` of `NOMAD` [5] with `MPICH 4.3.0`. We disable the use of quadratic models because this feature is not available in PSD-MADS. Our tests use OrthoMADS $2n$ due to the lack of an OrthoMADS $n+1$ implementation in pMADS-A and PSD-MADS. The number of parallel resources used is set according to the dimension of the given problem and the number of parallel workers that can compute an evaluation given an algorithm. Hence, depending on the algorithm, the number of parallel workers changes. For pMADS, a single process out of the total number provided

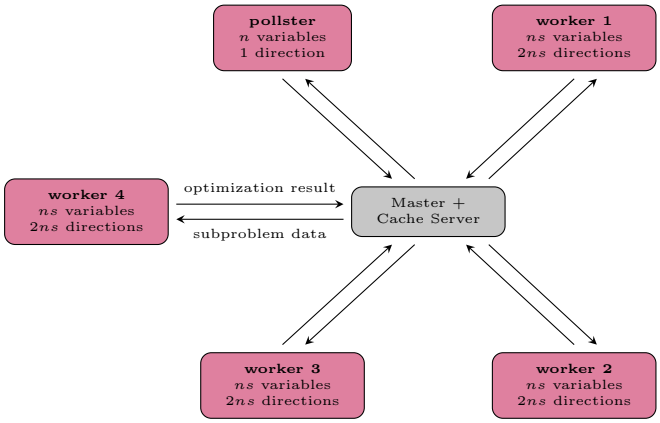

Figure 6: Master-worker communication organization for PSD-MADS, where for each worker p, the optimization result corresponds to the final mesh size Δ^p_{end} and the final solution $\mathbf{x}_p$, while the subproblem data corresponds to the starting solution $\mathbf{x}_0$, the set of variables $\mathcal{N}_p$, and the initial and minimum mesh sizes Δ^p_0 and Δ^p_{min}.

to `MPI` is used as the master, leading to the use of $n+1$ processes. For COOP-MADS, we consider $n+1$ MADS instance processes for the algorithm. For PSD-MADS, we use a master and a cache server that do not compute evaluations, leading then to a total of $n+2$ processes. Blackbox evaluations are performed in batch mode, with input and output files managed by `NOMAD`.

5.1 Scalability analysis

The performance of a parallel implementation is mainly characterized by its capacity to reduce a task execution time based on the computing units it uses, which refers to scalability. We use the following two metrics to measure scalability: speedup and efficiency. Let $S(\eta, p)$ denote the speedup for a task of size η using p computing units, defined as:

$$S(\eta, p) = \frac{T_1(\eta)}{T_p(\eta)},$$

where $T_1(\eta)$ and $T_p(\eta)$ are the execution times of the sequential and parallel implementations, respectively. Let the efficiency $E(\eta, p)$ be defined as:

$$E(\eta, p) = \frac{S(\eta, p)}{p}.$$

The efficiency relates the speedup to the number of parallel processing units used to achieve the task. In our case, the task to parallelize is not predetermined and it is the result of a dynamic aggregation of smaller tasks, namely, the blackbox functions evaluations, as the optimization algorithm proceeds. Hence, efficiency and speedup depend on communication overheads, blackbox evaluation times, and possible synchronization barriers in the algorithm. Maximum speedup and efficiency occur when

communication and synchronization overheads stay small for a relatively large number of computing units p.

A scalability analysis is performed with a fixed size η of 500 blackbox evaluations. A comparison between pMADS-A runs on a blackbox from [57]' collection set with different artificial evaluation times (tBBE) is presented in Figure 7. These results illustrate that parallel implementation significantly accelerates the computation time. Even at the lowest measured efficiency, a speedup of 43 is achieved using 64 processors. When subject to an evaluation time of 30 seconds, the speedup increases from 43 to 54. Hence, for more time-consuming blackbox functions, the parallel performance improves further, as the impact of communication overheads becomes negligible.

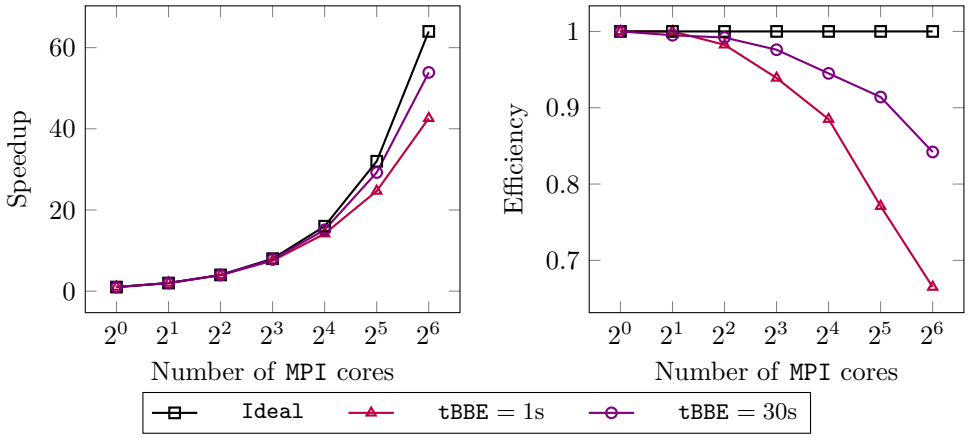

Figure 7: Parallel speedup and efficiency of pMADS-A measured over an optimization problem of 32 variables with a budget of 500 evaluations and a blackbox evaluation time (tBBE) of 1s and 30s. The reported number of MPI cores represents the worker processors.

5.2 Heterogeneous test

Next, we illustrate the impact of the synchronisation barrier over the computation time by comparing pMADS-S and pMADS-A. The solar simulator [58] implements in C++ a concentrated solar power thermal plant which relies on numerical methods such as Monte Carlo simulation, Newton's method, kernel smoothing, and other iterative methods. The heterogeneous nature of the blackbox comes from combining these methods. The blackbox solar offers ten instances, from which we use solar4.1 because it presents the largest dimension for a single-objective constrained optimization problem, that is, 29 variables and 16 constraints.

Figure 8 illustrates that as more processors are involved, the run time difference between pMADS-S and pMADS-A increases. This is due to the presence of the poll synchronization barrier in pMADS-S because, e.g., it only takes one evaluation with a significantly higher computation time than the others running in parallel to cause

substantial computation resource idling. Increasing the number of cores then leads to more evaluations being launched in parallel, thereby increasing the likelihood of encountering a more costly evaluation at each iteration. The potential slow down is particularly sensitive to the blackbox evaluation time distribution but is completely avoided in the case of pMADS-A which steadily decreases its run time.

Due to its clear limitation, for the rest of this work, pMADS-S will be omitted.

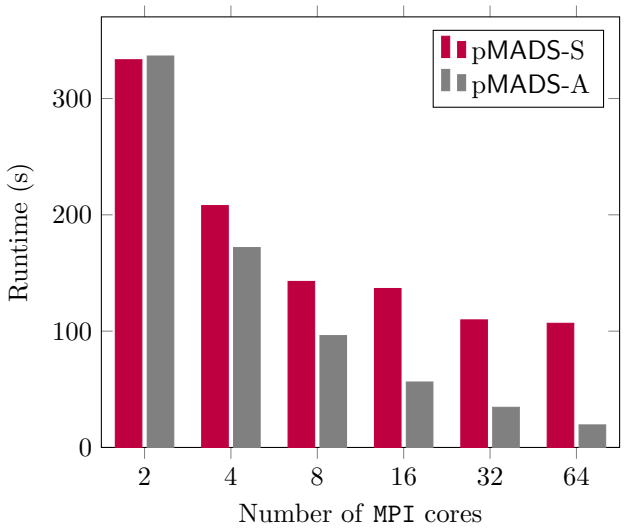

Figure 8: Running time in seconds of pMADS-A and pMADS-S for different numbers of `MPI` cores when solving solar4.1 with a 500 blackbox evaluation budget. The reported number of `MPI` cores represents both the master and worker processors.

5.3 Benchmarking with data profiles

Data profiles are used for benchmarking algorithms [59]. Comparisons are done in terms of computation time and number of evaluations scaled with problem dimensions n_p. The data profile of an algorithm is a series of values $d_a(k)$ that gives the proportion of τ-solved problems, with k the groups of n_p+1 function evaluations or the time [60]. The accuracy of an evaluation point is obtained at a given time or evaluation number for a given problem using the best and the initial objective values.

In this work, we have considered the cross-instances best objective value f^* obtained by all algorithms on all run instances of a given problem. The initial points of problems are given and fixed. Different run instances are obtained by providing fixed seeds to run algorithms.

5.4 Moré-Wild tests

In this section, the parallel versions of MADS, are compared on the Moré-Wild [59] problems. The Moré-Wild collection is a well-know test set for benchmarking unconstrained DFO algorithms in low dimensions. We select the SMOOTH subset which provides 53 unconstrained optimization problems where the objective function is twice continuously differentiable. For each parallel version of MADS and each problem, runs are conducted with 10 different random seeds that affect the trial points generation. This gives a total of 530 run instances per algorithm tested.

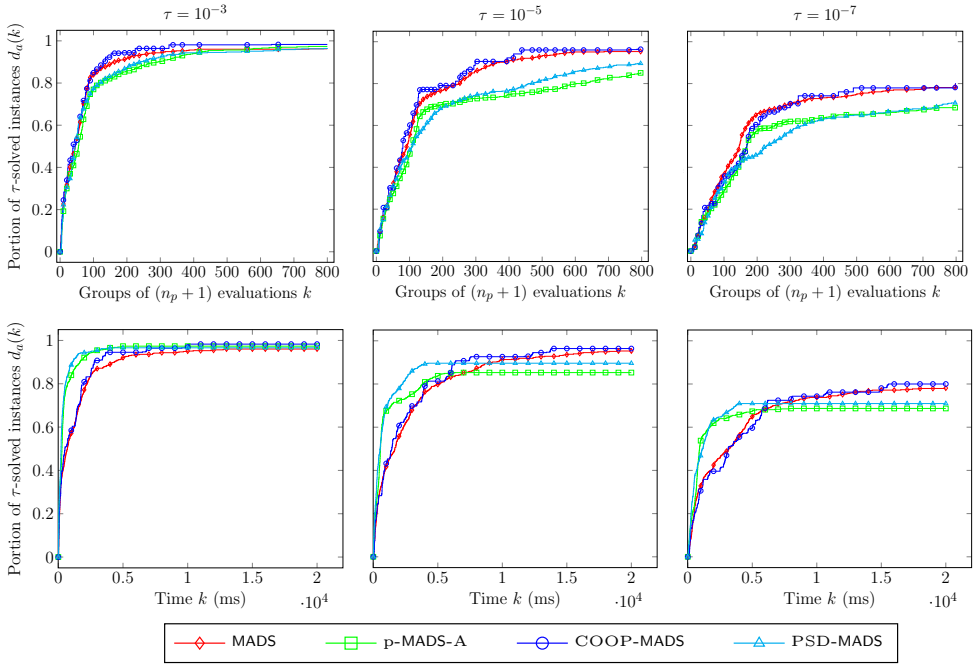

Figure 9: Data profiles on the Moré-Wild SMOOTH problem set.

In Figure 9, for $\tau = 10^{-5}$ and below, MADS and COOP-MADS solve more problems than the pMADS and PSD-MADS for the given evaluation budgets. We note that evaluation profiles for pMADS and PSD-MADS are still increasing towards the end. This probably indicates that more problem could have been solved by giving a higher evaluation budget. When looking at portion of solved instances with respect to time, pMADS and PSD-MADS achieve their best results faster, with flatter profiles.

5.5 Constrained test

Lastly, we consider a set of continuous constrained problems and use parallel versions of MADS with progressive barrier [53]. The progressive barrier offers a more sophisticated approach to handling inequality constraints compared to the extreme barrier described

Table 1: Description of the selected continuous constrained test set. The superscript § denotes the use of multiple starting points for the given problem.

#	Name	Ref.	n	m	Bnds
1	CHENWANG-F2§	[61]	8	6	yes
2	CHENWANG-F3§	[61]	10	8	yes
3	CRESCENT	[53]	10	2	yes
4	DISK	[53]	10	1	no
5	FLOUDAS	[62]	3	3	yes
6	G2	[63]	10	2	yes
7	G9	[63]	7	4	yes
8	HS83	[64]	5	6	yes
9	HS108	[64]	9	13	yes
10	HS114	[64]	9	6	yes
11	MAD6§	[61]	5	7	yes
12	OPTENG-RBF	[53]	3	4	yes
13	PENTAGON	[65]	6	15	no
14	SPRING§	[66]	3	4	yes
15	TAOWANG-F2§	[67]	7	4	yes
16	ZHAOWANG-F5	[61]	13	9	yes

in Algorithm 1. As the default strategy in NOMAD, the progressive barrier influences both the detection of success and the management of incumbents, allowing for the identification of two incumbent solutions; a feasible and an infeasible one. Despite its added complexity, it has demonstrated greater effectiveness than the extreme barrier, while maintaining comparable parallelization capabilities.

Table 1 lists and references the problem instances. Ten seeds are again considered to increase the problem collection. Data profiles for different tolerances τ are compared in Figure 10. PSD-MADS do not perform well compared with the other algorithms. The strategy of space decomposition seems not well adapted on small dimension problems with inequality constraints. pMADS-A performance is higher on smaller computation time and as good as COOP-MADS and MADS in terms of evaluations.

5.6 Discussion

The previous subsections numerically illustrates the effectiveness of parallel variants of the MADS algorithm. The results highlight the critical importance of employing an asynchronous parallel strategy when subject to heterogeneous evaluation times among blackbox functions. Lastly, the pMADS implementations demonstrates strong scalability performance with its efficient utilization of up to 64 computing cores. This in turn confirms its suitability for parallel computing contexts.

6 Closing remarks

In this work, we discuss and numerically compare the parallel extensions of the mesh adaptive direct search (MADS) algorithm for blackbox optimization (BBO). First, variants with a synchronization barrier, parallel MADS-Synchronous (pMADS-S) and without, parallel MADS-Asynchronous (pMADS-A), between evaluations are presented. While omitting the barrier improves computational efficiency, it also leads to the non-deterministic behaviour of pMADS-A. Next, the cooperative-MADS (COOP-MADS) algorithm, where several MADS instances are run in parallel with a shared cache, is outlined. Lastly, we survey the parallel space decomposition MADS (PSD-MADS), which specifically targets high-dimensional BBO problems by employing MADS on variable subspaces. The aforementioned extensions are designed such that

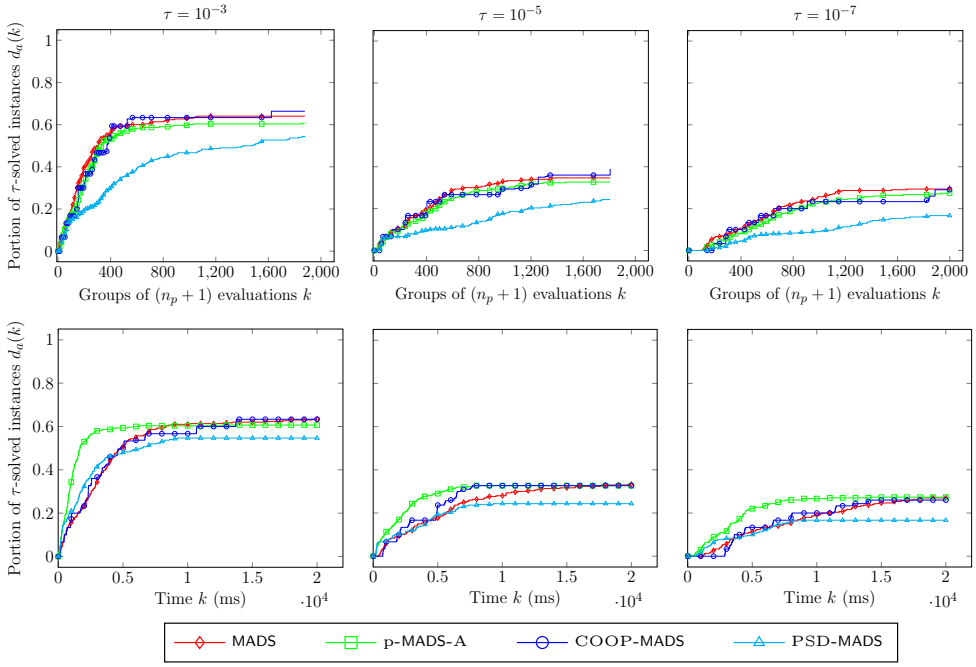

Figure 10: Data profiles on the inequality-constrained problem set of Table 1.

the local convergence analysis of MADS holds for them as well. Finally, the benefits of each variant, e.g., in terms of scalability, efficiency, running time and/or number of evaluations, is exemplified in numerical simulations. In both unconstrained and constrained problems, p-MADS-A stands out as a time-effective extension for BBO on multiple cores. In the future, our study is to be extended to the NOMAD 4 implementation of MADS. While, the resolution performance should be similar, the computation time and resource scalability may be affected by the its shared memory implementation.

Acknowledgments

This work is supported by the NSERC Alliance-Mitacs Accelerate grant ALLRP 571311-21 ("Optimization of future energy systems") in collaboration with Hydro-Québec.

Data availability statement

The open-source solver NOMAD [5] is available at https://www.gerad.ca/en/software/nomad/.

Use of AI statement

The authors used AI-assisted tools (ChatGPT, OpenAI) solely to improve the language and readability of the manuscript; all scientific content and conclusions are entirely the authors' own.

References

[1] Audet, C., Hare, W.: "Derivative-Free and Blackbox Optimization", 2nd edn. Springer Series in Operations Research and Financial Engineering. Springer, Cham, Switzerland (2026). https://doi.org/10.1007/978-3-032-00906-7

[2] Conn, A.R., Scheinberg, K., Vicente, L.N.: "Introduction to Derivative-Free Optimization". MOS-SIAM Series on Optimization. SIAM, Philadelphia (2009). https://doi.org/10.1137/1.9780898718768

[3] Alarie, S., Audet, C., Gheribi, A.E., Kokkolaras, M., Le Digabel, S.: Two decades of blackbox optimization applications. EURO Journal on Computational Optimization **9**, 100011 (2021) https://doi.org/10.1016/j.ejco.2021.100011

[4] Audet, C., Dennis, Jr., J.E.: Mesh Adaptive Direct Search Algorithms for Constrained Optimization. SIAM Journal on Optimization **17**(1), 188–217 (2006) https://doi.org/10.1137/040603371

[5] Le Digabel, S.: Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm. ACM Transactions on Mathematical Software **37**(4), 44–14415 (2011) https://doi.org/10.1145/1916461.1916468

[6] Olsson, P.-M.: Methods for Network Optimization and Parallel Derivative-Free Optimization. PhD thesis, Linköping University (2014). https://www.diva-portal.org/smash/get/diva2:695431/FULLTEXT02.pdf

[7] Schryen, G.: Parallel computational optimization in operations research: A new integrative framework, literature review and research directions. European Journal of Operational Research **287**(1), 1–18 (2020) https://doi.org/10.1016/j.ejor.2019.11.033

[8] Dennis, Jr., J.E., Wu, Z.: Parallel continuous optimization. Sourcebook of parallel computing, pp. 649–670. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003). http://portal.acm.org/citation.cfm?id=941480.941503

[9] Torczon, V.: On the Convergence of Pattern Search Algorithms. SIAM Journal on Optimization **7**(1), 1–25 (1997) https://doi.org/10.1137/S1052623493250780

[10] Kolda, T.G., Lewis, R.M., Torczon, V.: Optimization by direct search: New perspectives on some classical and modern methods. SIAM Review **45**(3), 385–482 (2003) https://doi.org/10.1137/S003614450242889

[11] Griffin, J.D., Kolda, T.G.: Nonlinearly-constrained optimization using heuristic penalty methods and asynchronous parallel generating set search. Applied Mathematics Research eXpress **25**(5), 36–62 (2010) https://doi.org/10.1093/amrx/abq003

[12] Griffin, J.D., Kolda, T.G., Lewis, R.M.: Asynchronous parallel generating set search for linearly-constrained optimization. SIAM Journal on Scientific Computing **30**(4), 1892–1924 (2008) https://doi.org/10.1137/060664161

[13] Hough, P.D., Kolda, T.G., Torczon, V.: Asynchronous Parallel Pattern Search for Nonlinear Optimization. SIAM Journal on Scientific Computing **23**(1), 134–156 (2001) https://doi.org/10.1137/S1064827599365823

[14] Kolda, T.G.: Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization. SIAM Journal on Optimization **16**(2), 563–586 (2005) https://doi.org/10.1137/040603589

[15] Kolda, T.G., Torczon, V.: On the Convergence of Asynchronous Parallel Pattern Search. SIAM Journal on Optimization **14**(4), 939–964 (2004) https://doi.org/10.1137/S1052623401398107

[16] Gray, G.A., Kolda, T.G.: Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. ACM Transactions on Mathematical Software **32**(3), 485–507 (2006) https://doi.org/10.1145/1163641.1163647

[17] Plantenga, T.D.: HOPSPACK 2.0 User Manual. Technical Report SAND2009-6265, Sandia National Laboratories, Livermore, CA (October 2009). http://www.sandia.gov/hopspack/HopspackUserManual_2_0_2.pdf

[18] Liuzzi, G., Truemper, K.: Parallelized hybrid optimization methods for nonsmooth problems using NOMAD and linesearch. Computational and Applied Mathematics **37**(3), 3172–3207 (2018) https://doi.org/10.1007/s40314-017-0505-2

[19] Talgorn, B., Alarie, S., Kokkolaras, M.: Parallel Surrogate-based Optimization Using Mesh Adaptive Direct Search. Technical Report G-2020-38, Les cahiers du GERAD (2020). https://www.gerad.ca/en/papers/G-2020-38

[20] Nelder, J.A., Mead, R.: A Simplex Method for Function Minimization. The Computer Journal **7**(4), 308–313 (1965) https://doi.org/10.1093/comjnl/7.4.308

[21] Dennis, Jr., J.E., Torczon, V.: Direct Search Methods on Parallel Machines. SIAM Journal on Optimization **1**(4), 448–474 (1991) https://doi.org/10.1137/0801027

[22] Ozaki, Y., Watanabe, S., Onishi, M.: Accelerating the Nelder-Mead method with predictive parallel evaluation. 6th ICML Workshop on Automated Machine Learning **185**, 186 (2019)

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

[24] Griffin, J.D., Kolda, T.G.: Asynchronous parallel hybrid optimization combining DIRECT and GSS. Optimization Methods and Software **25**(5), 797–817 (2010) https://doi.org/10.1080/10556780903039893

[25] He, J., Verstak, A., Sosonkina, M., Watson, L.T.: Performance Modeling and Analysis of a Massively Parallel DIRECT–Part 2. International Journal of High Performance Computing Applications **23**(1), 29–41 (2009) https://doi.org/10.1177/1094342008098463

[26] He, J., Verstak, A., Watson, L.T., Sosonkina, M.: Design and implementation of a massively parallel version of DIRECT. Computational Optimization and Applications **40**(2), 217–245 (2007) https://doi.org/10.1007/s10589-007-9092-2

[27] He, J., Verstak, A., Watson, L.T., Sosonkina, M.: Performance Modeling and Analysis of a Massively Parallel DIRECT–Part 1. International Journal of High Performance Computing Applications **23**(1), 14–28 (2009) https://doi.org/10.1177/1094342008098462

[28] He, J., Watson, L.T., Sosonkina, M.: Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT. ACM Transactions on Mathematical Software **36**(3), 1–24 (2009) https://doi.org/10.1145/1527286.1527291

[29] Berghen, F.V.: CONDOR: A Constrained, Non-Linear, Derivative-Free Parallel Optimizer for Continuous, High Computing Load, Noisy Objective Functions. PhD thesis, Université Libre de Bruxelles, Belgium (2004). http://www.applied-mathematics.net/optimization/thesis_optimization.pdf

[30] Berghen, F.V., Bersini, H.: CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. Journal of Computational and Applied Mathematics **181**, 157–175 (2005) https://doi.org/10.1016/j.cam.2004.11.029

[31] Hough, P.D., Meza, J.C.: A Class of Trust-Region Methods for Parallel Optimization. SIAM Journal on Optimization **13**(1), 264–282 (2002) https://doi.org/10.1137/S1052623498343799

[32] Regis, R.G., Shoemaker, C.A.: Parallel radial basis function methods for the global optimization of expensive functions. European Journal of Operational Research **182**(2), 514–535 (2007) https://doi.org/10.1016/j.ejor.2006.08.040

[33] Shoemaker, C.A., Regis, R.G.: "mapo: using a committee of algorithm-experts for parallel optimization of costly functions". In: "SPAA '03: Proceedings of

the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures", pp. 242–243 (2003). https://doi.org/10.1145/777412.777451 . ACM. https://doi.org/10.1145/777412.777451

[34] Xia, W., Shoemaker, C.A.: GOPS: efficient RBF surrogate global optimization algorithm with high dimensions and many parallel processors including application to multimodal water quality PDE model calibration. Optimization and Engineering **22**(4), 2741–2777 (2021) https://doi.org/10.1007/s11081-020-09556-1

[35] Briffoteaux, G.: Parallel surrogate-based algorithms for solving expensive optimization problems. PhD thesis, Université de Lille; Université de Mons (2023). https://hal.science/tel-03853862v2

[36] García-García, J.C., García-Ródenas, R., Codina, E.: A surrogate-based cooperative optimization framework for computationally expensive black-box problems. Optimization and Engineering **21**(3), 1053–1093 (2020) https://doi.org/10.1007/s11081-020-09526-7

[37] Haftka, R.T., Villanueva, D., Chaudhuri, A.: Parallel surrogate-assisted global optimization with expensive functions – a survey. Structural and Multidisciplinary Optimization **54**(1), 3–13 (2016) https://doi.org/10.1007/s00158-016-1432-3

[38] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient Global Optimization of Expensive Black Box Functions. Journal of Global Optimization **13**(4), 455–492 (1998) https://doi.org/10.1023/A:1008306431147

[39] Tran, A., Sun, J., Furlan, J.M., Pagalthivarthi, K.V., Visintainer, R.J., Wang, Y.: pBO-2GP-3B: A batch parallel known/unknown constrained Bayesian optimization with feasibility classification and its applications in computational fluid dynamics. Computer Methods in Applied Mechanics and Engineering **347**, 827–852 (2019) https://doi.org/10.1016/j.cma.2018.12.033

[40] Zhan, D., Qian, J., Cheng, Y.: Pseudo expected improvement criterion for parallel EGO algorithm. Journal of Global Optimization **68**(3), 641–662 (2017) https://doi.org/10.1007/s10898-016-0484-7

[41] Zhan, D., Xing, H.: Expected improvement for expensive optimization: a review. Journal of Global Optimization **78**(3), 507–544 (2020) https://doi.org/10.1007/s10898-020-00923-x

[42] Vázquez, S., Martín, M.J., Fraguela, B.B., Gómez, A., Rodríguez, A., Elvarsson, B.: Novel parallelization of simulated annealing and Hooke & Jeeves search algorithms for multicore systems with application to complex fisheries stock assessment models. Journal of Computational Science **17**, 599–608 (2016) https://doi.org/10.1016/j.jocs.2016.07.003

[43] Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances

and new trends. International Transactions in Operational Research **20**(1), 1–48 (2013) https://doi.org/10.1111/j.1475-3995.2012.00862.x

[44] Audet, C., Dang, C.-K., Orban, D.: Efficient use of parallelism in algorithmic parameter optimization applications. Optimization Letters **7**(3), 421–433 (2013) https://doi.org/10.1007/s11590-011-0428-6

[45] Larson, J., Wild, S.M.: Asynchronously Parallel Optimization Solver for Finding Multiple Minima. Mathematical Programming Computation **10**(3), 303–332 (2018) https://doi.org/10.1007/s12532-017-0131-4

[46] Tavares, S., Brás, C.P., Custódio, A.L., Duarte, V., Medeiros, P.: Parallel strategies for Direct Multisearch. Numerical Algorithms **92**(3), 1757–1788 (2023) https://doi.org/10.1007/s11075-022-01364-1

[47] Pang, M., Shoemaker, C.A.: Comparison of parallel optimization algorithms on computationally expensive groundwater remediation designs. Science of The Total Environment **857**, 159544 (2023) https://doi.org/10.1016/j.scitotenv.2022.159544

[48] Campos, G., da Silva Pereira, W., Vercellino, R., Mueller, J., Mann, M.: Parallel derivative-free optimization for simulation-based design of behind-the-meter energy systems. Computers and Chemical Engineering, 109422 (2025) https://doi.org/10.1016/j.compchemeng.2025.109422

[49] Audet, C., Le Digabel, S., Tribes, C.: The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. SIAM Journal on Optimization **29**(2), 1164–1189 (2019) https://doi.org/10.1137/18M1175872

[50] Audet, C., Béchard, V., Le Digabel, S.: Nonsmooth optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search. Journal of Global Optimization **41**(2), 299–318 (2008) https://doi.org/10.1007/s10898-007-9234-1

[51] Audet, C., Tribes, C.: Mesh-based Nelder-Mead algorithm for inequality constrained optimization. Computational Optimization and Applications **71**(2), 331–352 (2018) https://doi.org/10.1007/s10589-018-0016-0

[52] Conn, A.R., Le Digabel, S.: Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. Optimization Methods and Software **28**(1), 139–158 (2013) https://doi.org/10.1080/10556788.2011.623162

[53] Audet, C., Dennis, Jr., J.E.: A Progressive Barrier for Derivative-Free Nonlinear Programming. SIAM Journal on Optimization **20**(1), 445–472 (2009) https://doi.org/10.1137/070692662

[54] Audet, C., Dennis, Jr., J.E., Le Digabel, S.: Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm. SIAM Journal on Optimization **19**(3),

1150–1170 (2008) https://doi.org/10.1137/070707518

[55] Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J.: "MPI: The Complete Reference". The MIT Press, Cambridge, Massachusetts (1995)

[56] Abramson, M.A., Audet, C., Dennis, Jr., J.E., Le Digabel, S.: OrthoMADS: A Deterministic MADS Instance with Orthogonal Directions. SIAM Journal on Optimization **20**(2), 948–966 (2009) https://doi.org/10.1137/080716980

[57] Karmitsa, N.: Test problems for large-scale nonsmooth minimization. Technical Report B. 4/2007, Department of Mathematical Information Technology, University of Jyväskylä, Jyväskylä, Finland (2007). https://urn.fi/URN:ISBN:978-951-39-2784-4

[58] Andrés-Thió, N., Audet, C., Diago, M., Gheribi, A.E., Le Digabel, S., Lebeuf, X., Lemyre Garneau, M., Tribes, C.: Solar: a solar thermal power plant simulator for blackbox optimization benchmarking. Optimization and Engineering **26**(3), 1815–1861 (2025) https://doi.org/10.1007/s11081-024-09952-x

[59] Moré, J.J., Wild, S.M.: Benchmarking Derivative-Free Optimization Algorithms. SIAM Journal on Optimization **20**(1), 172–191 (2009) https://doi.org/10.1137/080724083

[60] Audet, C., Hare, W., Tribes, C.: A summary of benchmarking constrained, multi-objective and surrogate-assisted optimization methods. Optimization Letters (2026) https://doi.org/10.1007/s11590-026-02302-z

[61] Zhao, J., Wang, N.: A bio-inspired algorithm based on membrane computing and its application to gasoline blending scheduling. Computers and Chemical Engineering **35**(2), 272–283 (2011) https://doi.org/10.1016/j.compchemeng.2010.01.008

[62] Floudas, C.A.: "Nonlinear and Mixed-integer Optimization: Fundamentals and Applications". Oxford University Press, New York (1995). https://doi.org/10.1093/oso/9780195100563.001.0001

[63] Hedar, A.-R., Fukushima, M.: Derivative-free filter simulated annealing method for constrained continuous global optimization. Journal of Global Optimization **35**(4), 521–549 (2006) https://doi.org/10.1007/s10898-005-3693-z

[64] Hock, W., Schittkowski, K.: "Test Examples for Nonlinear Programming Codes". Lecture Notes in Economics and Mathematical Systems, vol. 187. Springer, Berlin, Germany (1981)

[65] Lukšan, L., Vlček, J.: Test Problems for Nonsmooth Unconstrained and Linearly Constrained Optimization. Technical Report V-798, ICS AS CR (2000)

[66] Rodríguez, J.F., Renaud, J.E., Watson, L.T.: Trust Region Augmented Lagrangian Methods for Sequential Response Surface Approximation and Optimization. Journal of Mechanical Design **120**(1), 58–66 (1998) https://doi.org/10.1115/1.2826677

[67] Tao, J., Wang, N.: DNA Double Helix Based Hybrid GA for the Gasoline Blending Recipe Optimization Problem. Chemical Engineering and Technology **31**(3), 440–451 (2008) https://doi.org/10.1002/ceat.200700322