Open access • Journal Article • DOI:10.1137/070707518

Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm

Charles Audet, John E. Dennis, Sébastien Le Digabel

Institutions: Rice University

Published on: 01 Oct 2008 - Siam Journal on Optimization (Society for Industrial and Applied Mathematics)

Topics: Parallel algorithm and Optimization problem

Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm *

Charles Audet † J.E. Dennis Jr. ‡ Sébastien Le Digabel §

7th November 2007

Abstract

This paper describes a Parallel Space Decomposition (PSD) technique for the Mesh Adaptive Direct Search (MADS) algorithm. MADS extends Generalized Pattern Search for constrained nonsmooth optimization problems. The objective here is to solve larger problems more efficiently. The new method (PSD-MADS) is an asynchronous parallel algorithm in which the processes solve problems over subsets of variables. The convergence analysis based on the Clarke calculus is essentially the same as for the MADS algorithm. A practical implementation is described and some numerical results on problems with up to 500 variables illustrate advantages and limitations of PSD-MADS.

Keywords: Parallel Space Decomposition, Mesh Adaptive Direct Search, Asynchronous parallel algorithm, Nonsmooth optimization, Convergence analysis.

*Work of the first author was supported by FQRNT and NSERC. Work of the first and second authors was supported by AFOSR, the Boeing Company, and ExxonMobil Upstream Research Company. Work of the first and third authors was supported by the Consortium for Research and Innovation in Aerospace in Québec.

†GERAD and Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal (Québec), H3C 3A7 Canada, www.gerad.ca/Charles.Audet, Charles.Audet@gerad.ca

‡Computational and Applied Mathematics Department, Rice University, 8419 42nd Ave SW, Seattle, WA 98136 http://www.caam.rice.edu/∼dennis, dennis@rice.edu

§GERAD and Département de Mathématiques et de Génie Industriel, Ecole Polytechnique de Montréal, Sebastien.Le.Digabel@gerad.ca

1 Introduction

The paper considers optimization problems of the form

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

where the objective function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$. The feasible region Ω is assumed to satisfy a nonsmooth constraint qualification, and we only assume the presence of an oracle to tell whether or not a given $x \in \mathbb{R}^n$ is feasible. We are concerned primarily with cases where $f(x)$ or the oracle are given by black-box computer simulations, which are assumed to evaluate in finite time. This is common in engineering design. Indeed, the reason we allow $f(x)$ to take on the value ∞ is that for many such problems, no value of $f(x)$ is returned even for $x \in \Omega$ because of the internal workings of the simulation used to drive the design. See [1, 3, 10, 13, 21, 27, 32, 42].

There are other useful derivative-free direct search methods designed for problems similar to $\mathcal{P}$. These include the Nelder-Mead simplex [43], the DIRECT algorithm [20, 24, 30], the frame based methods [16, 44], the Generalized Pattern Search (GPS) [7, 14, 49], the Asynchronous Parallel Pattern Search (APPS) approach [25, 29, 33, 35, 36], and the Mesh Adaptive Direct Search (MADS) [2, 8]. Related is the implicit filter method [31], though it does use a coarse difference gradient approximation. The reader may consult [31, 34, 39] for a survey of some of these direct search methods.

Using these methods to solve expensive problems with more than a few dozen variables may be impractical since they may need a large number of costly black-box evaluations. One possible approach is to use parallelization. Dennis and Wu [18] reviewed different parallel methods for continuous optimization and concluded that a combination of GPS and the Parallel Variable Distribution (PVD) of Ferris and Mangasarian [19] should be considered.

PVD is an evolution of the block-Jacobi technique of [11] which optimizes in parallel a series of reduced subproblems on subspaces of the original variables of $\mathcal{P}$. The present paper is based on the remark of Dennis and Wu. Dennis and Torczon [17] described a first parallel version of GPS, which evaluates the black-box evaluations in parallel and synchronizes at each iteration to compare solutions and update the current iterates. The Asynchronous Parallel Pattern Search, APPS [25, 33], removes this synchronization barrier. In APPS, each process explores the space of variables using its own set of directions and does not wait for the other processes to terminate. APPS is expected to be more efficient than the synchronous version of [17], especially if the black-boxes have heterogeneous behavior that depends on the point where they are evaluated. A convergence analysis is presented in [36] for the smooth case.

Our work applies a decomposition of the variables of $\mathcal{P}$ based on the block-Jacobi technique of [11] that inspired the PVD method of [19]. This allows a natural parallel application of MADS to smaller subproblems, in an asynchronous way. The new al-

gorithm, called PSD-MADS, can be interpreted as a particular instance of MADS, thus inheriting the main results of the MADS convergence analysis.

The paper is divided as follows: Section 2 gives an overview of the Parallel Space Decomposition and MADS methods. Section 3 presents the new asynchronous parallel algorithm, PSD-MADS, and Section 4 shows that main convergence results of MADS are maintained by showing that the entire PSD-MADS algorithm may be interpreted as a specific MADS instance. An implementation of PSD-MADS is described in Section 5, with some numerical results on problems with a number of variables ranging from 20 to 500. Finally, Section 6 gives some conclusions and proposes possible extensions of PSD-MADS.

2 Relevant literature

This section presents an overview of parallel space decomposition methods, and the Mesh Adaptive Direct Search algorithm. MADS, its convergence analysis and its LT-MADS implementation are described in detail, since Sections 4 and 5 depends on them.

2.1 Parallel space decomposition methods

Parallel space decomposition methods decompose $\mathcal{P}$ into a finite number of smaller dimension subproblems, which can be solved in parallel with one process assigned to each subproblem.

Define $N = \{1, 2, ..., n\}$ where n is the number of variables of the optimization problem $\mathcal{P}$, and $Q = \{1, 2, ..., q\}$ where q is the number of available processes. Each process $p \in Q$ works on a nonempty subset $N_p \subseteq N$ of the variables. The other variables are fixed, based on the incumbent solution $x^* \in \Omega$, the current best known solution. More precisely, process $p \in Q$ works on the optimization subproblem

$$\min_{x \in \Omega_p(x^*)} f(x) \qquad (\mathcal{P}_p(x^*))$$

with $\Omega_p(x^*) = \left\{x \in \Omega \ : x_i = x_i^* \ \forall i \in \overline{N}_p\right\}$ and $\overline{N}_p = N \setminus N_p$. The subproblem $\mathcal{P}_p(x^*)$ contains $n_p = |N_p|$ free variables, indexed by N_p. In Section 5 we propose a strategy to build the subsets N_p.

The block-Jacobi method in [11] is an iterative two-steps algorithm and may be described in a very general way as follows. At each iteration, the first step, the *parallelization*, consists in solving the subproblems in parallel, and the second step, the *synchronization*, gathers the subproblems solutions and construct the next iterate. Similar methods are described in [26, 41, 50].

A variant of the method was introduced by Ferris and Mangasarian [19], as the Parallel Variable Distribution (PVD) for a differentiable objective function f with continuous

partial derivatives. In order to solve the subproblems more efficiently, the PVD method allows *a priori* fixed variables to change in a limited fashion, along directions typically based on ∇f. These variables are denoted as "forget-me-not" terms.

The convergence analysis in [19] requires that subproblems be solved to optimality. In the unconstrained case, if ∇f exists and is Lipschitz, then the accumulation points of the generated sequences are stationary points. In addition, if f is assumed to be convex, the convergence rate is shown to be linear. When Ω is nonempty, closed, convex, block-separable, and the functions defining it are also continuously differentiable, convergence results are still available. When there are general constraints, Ferris and Mangasarian recommend transforming the problem into unconstrained problems via penalty functions or exploiting possible block-separable constraints.

These are parallel synchronous algorithms because the synchronization step waits for all the processes to end. The conclusion of [19] is that an asynchronous version of the algorithm would increase efficiency. This is done in [40] for unconstrained problems, where the synchronization step is dropped at the expense of the convergence analysis.

Extensions of the PVD method are given in [45, 46, 47] with similar convergence results to those in [19] under less restrictive conditions. For example, subproblems do not need to be solved to complete optimality, as for example when one Newton-like iteration is used. A convergence analysis for the constrained case is given with either block separability or convexity assumptions on the structure of Ω. These are not reasonable assumptions for our target class of engineering design problems.

In the above references, no practical and generic strategy is given concerning the choice of the subproblems variables (sets N_p). However, the sets do need to form a partition of N, and they are fixed throughout the entire process. In the parallel space decomposition [22] the subspaces can be chosen differently at each iteration.

In Fukushima [23], the PVD method is extended to a more general framework for unconstrained problems. The sets of subproblems variables are not fixed through the iterations, are not required to form a partition of N, but they must span N. In particular, an overlapping of the subproblems variables is allowed. Some experiments with such methods are given in [51].

More recently, the MoVars algorithm [12] combines the GPS method with the synchronous PVD framework (including the "forget-me-not" terms from [19]) on fixed subsets N_p, but there is no convergence analysis.

In most of the references of this section, f is assumed to be at least differentiable, and constraints, if they are considered, are block-separable or convex. These assumptions are not reasonable for the problems of interest to us, and thus, our convergence analysis does not rely on the analysis of [19] or its extensions. Rather, by incorporating MADS with its weaker hypotheses, we will inherit the MADS convergence analysis. It will also give us greater flexibility concerning the choice of the subsets N_p, the way we handle constraints, the amount of work we must devote to the subproblems, and the lack of necessity for a synchronization step.

2.2 Mesh Adaptive Direct Search (MADS)

We now summarize the MADS algorithm [8] for problem $\mathcal{P}$, which extends the Generalized Pattern Search (GPS) algorithm for linearly constrained optimization [14, 49].

The constraints defining Ω are handled by the extreme barrier approach, as in [8, 37, 38]. This means that trial points outside Ω are simply rejected by setting their objective function value to $+\infty$. Of course, this requires that the user provide a feasible initial point $x_0 \in \Omega$. We make the standard assumption that all the trial points generated by the algorithm lie in a compact set.

MADS is an iterative algorithm where the black-box functions are evaluated at some trial points that are either accepted as new iterates because they are feasible and decrease the objective, or they are rejected.

All trial points generated by these algorithms are constructed to lie on a mesh

$$M(\Delta) = \bigcup_{x \in V} \left\{ x + \Delta D z : z \in \mathbb{N}^{n_D} \right\} \subset \mathbb{R}^n \tag{1}$$

where the set V, called the *cache*, is a data structure memorizing all previously evaluated points so that no double evaluations occur, $\Delta \in \mathbb{R}^+$ represents a mesh size parameter, and D is a $n \times n_D$ matrix representing a fixed finite set of n_D directions in $\mathbb{R}^n$. More precisely, D is called the set of mesh directions and is chosen so that $D = GZ$, where G is a non-singular $n \times n$ matrix, and Z a $n \times n_D$ integer matrix. The definition given by (1) differs slightly from the one in [8]. There the mesh was indexed by the iteration number instead of being parameterized by Δ. The reason for this difference is that our parallel algorithm will be working simultaneously on different size meshes originally generated at different iterations. Note also that in order to simplify the notation, the mesh size parameter Δ used here is the equivalent of Δ^m in [8].

Each iteration is divided into three steps, the search, the poll, and an update step determining the success of the iteration and producing the next iterate. The search and poll are treated specially in that the poll need not be carried out at an iteration if the search finds a better point. At each iteration, the algorithm attempts to generate an improved incumbent solution on the current mesh $M(\Delta_k)$, where Δ_k is the mesh size parameter at iteration k. The search step is very flexible and allows for trial points anywhere on the mesh. The way of generating these points is free of any rules, as long as they remain on the current mesh $M(\Delta_k)$ and that the search terminates in finite time. Some search strategies can be tailored for a specific application, while others are generic, such as the use of Latin Hypercube sampling [48], or Variable Neighborhood Search [4]. In summary, if one wants to define a new MADS algorithm with its specific search, all that needs to be done to ensure convergence is to show that the search requires finite time and generates a finite number of trial points lying on the mesh.

The poll step explores the mesh $M(\Delta_k)$ near the current iterate x_k and its rules ensure theoretical convergence of the algorithm. The way of choosing the directions

used to generate the poll points is the difference between GPS and MADS. In GPS, the normalized set of potential poll directions must be chosen from a finite set that is fixed across all iterations. In MADS, the directions may be chosen to be asymptotically dense in the unit sphere, which allows better coverage. We use the terminology of [16, 44] and say that at iteration k, the set of trial poll points is called the frame P_k. The set of directions used to construct P_k is denoted D_k, and it is not a subset of D.

In the last step of the kth iteration, the mesh size parameter is updated according to $\Delta_{k+1} \leftarrow \tau^{\omega_k}\Delta_k$, where $\tau > 1$ is a fixed rational number and ω_k an integer that depends on the success of the iteration. When no improvement is made, the iteration is said to fail, and ω_k is taken to be an integer in the interval $[\omega^-; -1]$ with $\omega^- \leq -1$, forcing the next trial poll points to be closer to the current iterate. When a new best iterate is found, the iteration is said to succeed, and Δ_k is possibly increased with ω_k in $[0; \omega^+]$, with the integer $\omega^+ \geq 0$. In the LTMADS implementation of [8], τ is fixed to 4, $\omega^- = -1$, and $\omega^+ = 1$.

A high level description of the algorithm is summarized in Figure 1. We encourage the reader to consult [8] for a complete description.

MADS

[0] Initializations

$x_0 \in \Omega$, $\Delta_0 > 0$, $k \leftarrow 0$

[1] Poll and search steps

objective: find a $y \in M(\Delta_k) \cap \Omega$ *such that* $f(y) < f(x_k)$

Search step (optional)

evaluate the functions on a finite number of points of $M(\Delta_k)$

Poll step (optional if the search step succeeded)

compute n_{dir} MADS directions $d_i \in \mathbb{R}^n$

evaluate the functions on the MADS frame $P_k = \{x_k + \Delta_k d_i : i = 1, ..., n_{dir}\} \subseteq M(\Delta_k)$

[2] Updates

$x_{k+1} \leftarrow y$ (iteration success) or x_k (iteration failure)

$\Delta_{k+1} \leftarrow \tau^{\omega_k}\Delta_k$ (reduced if iteration fails)

$k \leftarrow k + 1$

goto [1]

Figure 1: High level description of the MADS Algorithm. The directions d_i are positive integer combinations of the columns of D. The search or poll steps can be stopped before all evaluations are terminated (opportunistic strategy).

2.3 MADS convergence analysis

We will summarize the main convergence results for MADS given in [8]. These results assume that constraints are treated by the extreme barrier approach, and they constitute a hierarchical series of results relying on the Clarke calculus [15] for nonsmooth functions.

The main theorem is that under a local Lipschitz assumption on f, and under the assumption that the set of all normalized poll directions is dense in the unit sphere, the algorithm produces a Clarke stationary point. More precisely, MADS generates a point $\hat{x} \in \Omega$ at which the Clarke generalized directional derivatives of f in all the directions in the Clarke tangent cone at $\hat{x}$ are non negative. The only assumptions needed are that f is Lipschitz near $\hat{x}$ and the constraint qualification that the hypertangent cone of Ω at $\hat{x}$ is nonempty. A corollary to this result in the unconstrained case is that if f is strictly differentiable near $\hat{x}$, then $\nabla f(\hat{x}) = 0$.

The convergence result that requires the least assumptions on f and Ω, the zero'th order result, is that MADS generates a limit point $\hat{x}$, which is the limit of mesh local minimizers on meshes that get infinitely fine. The notion of local optimality is with respect to the current poll set, defined using a positive spanning set of directions. More formally, MADS generates a convergent subsequence of iterates $\{x_k\}_{k \in K} \subset \Omega$ such that $x_k \to \hat{x}$, and $f(x_k) \leq f(x_k + \Delta_k d_k)$ for all directions d_k in a positive spanning set D_K, and $\|\Delta_k d_k\| \to 0$.

The price to pay for our new capability to handle a large number of variables is that this last convergence result will be lost. We will consider a MADS algorithm whose poll set contains a single element instead of being built using a positive spanning set of directions. We will refer to this as a *single-poll* MADS algorithm, and it still retains the property of generating asymptotically dense polling directions.

The next section discusses the LTMADS implementation of the MADS algorithm. LTMADS uses positive bases to construct the poll sets. It is stated that the union of theses normalized directions forms a dense set because if one looks closely at the proof in [8], one sees that it is the subset of single-poll normalized MADS directions that grows dense in the unit sphere. Thus, with the assumption of local Lipschitz continuity the main convergence result guaranteeing a Clarke stationary point holds.

2.4 The LTMADS implementation of MADS

The search and poll steps need to satisfy certain conditions for the convergence results to hold. In particular, one of these conditions is that the total set of normalized poll directions used by the algorithm is dense in the unit sphere. In [8] there is a practical way of accomplishing this with through the LTMADS implementation of MADS.

LTMADS fixes τ to 4 and the set of mesh directions $D = [-I_n \; I_n]$ where I_n represents the $n \times n$ identity matrix. The mesh is based on the nonnegative integer value $\ell = -\log_4(\Delta_k)$, $\Delta_k = 4^{-\ell}$, and directions are constructed randomly using a lower tri-

angular matrix. One of these directions is a special case and fixed just once for each value of ℓ. This direction, called $b(\ell)$, has one coordinate set to $\pm 2^\ell$ so that poll points are within $\sqrt{\Delta_k}$ of the poll center x_k in the ℓ_∞ norm.

The result stated in [6, 8] is that with probability one, the series of normalized directions $b(\ell)$ grows dense in the unit sphere. In LTMADS, the direction $b(\ell)$ is augmented at each iteration with other directions to form a positive spanning set of polling directions. We can, as explained in the preceding section, construct a single-poll MADS algorithm with dense polling directions using only the $b(\ell)$ directions, but the zero'th order convergence result of MADS is lost. Also, because we are not polling at each iteration in a positive spanning set of directions, the mesh size might drop too quickly with this single-poll version of MADS, and so the search step is of extra importance. This is the key to the PSD-MADS algorithm described in the next section: one process executes a single-poll MADS algorithm, while the work of the other processes may be interpreted as a search step.

3 Parallel Space Decomposition of MADS

This section describes the combination of MADS with a parallel space decomposition method. The resulting algorithm is called PSD-MADS. It is an asynchronous parallel algorithm where a master process decides on the subsets $N_p \subseteq N$ and assigns the resulting optimization subproblems $\mathcal{P}_p(x^*)$ to slaves. The slaves apply MADS to attempt to improve the incumbent solution x^*. No synchronization step is performed. When a slave completes its assigned task, the master assigns a new subproblem with a possible new N_p and x^*.

3.1 General description of PSD-MADS

While PSD-MADS is an asynchronous parallel algorithm, the notion of iteration is kept, and corresponds to two successive calls by the master to one special slave, called the *pollster* slave, described more precisely in Section 3.2. The pollster slave executes a single-poll MADS algorithm on the entire problem $\mathcal{P}$, while the other slaves, called the *regular* slaves, work on the subproblems $\mathcal{P}_p(x^*)$.

Each subproblem $\mathcal{P}_p(x^*)$ is a subproblem of $\mathcal{P}$ with a reduced number of variables indexed by the set N_p. When an optimization process terminates, the slave communicates its progress to the master. If it has found an improved solution, then that becomes the new incumbent solution. The slave immediately starts work on a new subproblem assigned by the master. There is no need to synchronize all the slaves.

With several MADS instances executing in parallel, it is necessary to define different mesh size parameters: first, Δ_j^p corresponds to the mesh $M(\Delta_j^p)$ used at iteration j of the MADS algorithm performed by a regular slave s_p. This mesh size parameter is denoted

differently for the pollster slave, with Δ_k^1 (notice the same iteration counter k used both for the pollster slave and PSD-MADS). Δ_k^1 is called the *pollster mesh size parameter* at iteration k of PSD-MADS. Finally, an additional mesh size parameter, Δ_k^M, is called the *master mesh size parameter*. The mesh $M(\Delta_k^M)$ is never used explicitly, but it is useful to compare the two other meshes. At iteration k of PSD-MADS, and at iteration j of the MADS algorithm performed on a subproblem $\mathcal{P}_p(x^*)$ by a regular slave s_p for $p \in Q_{reg}$, the PSD-MADS construction ensures that

$$\Delta_k^1 \leq \Delta_k^M \leq \Delta_j^p \quad . \tag{2}$$

Inequalities (2) are formally proved in the convergence analysis of Section 4, where PSD-MADS is interpreted as a valid single-poll MADS instance performed by the pollster slave. An additional hypothesis on the different meshes $M(\Delta_k^M)$, $M(\Delta_k^1)$, and $M(\Delta_j^p)$ is necessary:

Hypothesis 3.1 *If two mesh size parameters Δ and Δ' satisfy $\Delta = \tau^\omega \Delta'$ where $\omega \in \mathbb{N}$, then $M(\Delta) \subseteq M(\Delta')$.*

This assumption holds for the PSD-MADS implementation given in Section 5.

The q processes are partitioned into a master, $q-2$ slaves, and a cache server (process number $q-1$), which memorizes all points that have been evaluated. The $q-2$ slaves include the pollster slave (process number 1) and $q-3$ regular slaves. The notation s_p with $p \in Q \setminus \{q-1, q\}$ is used to identify the $q-2$ processes assigned as slaves, and $Q_{reg} = \{2, 3, ..., q-2\}$ is the set of indices of the $q-3$ regular slaves. The qth and last process is used as the master, which defines the lower dimensional subproblems $\mathcal{P}_p(x^*)$ and communicates them to the slaves.

An advantage of applying the parallel space decomposition method to MADS instead of another optimization method is that most of the conditions necessary for convergence in other parallel space decomposition methods mentioned in Section 2.1 can be relaxed:

- f and the functions defining Ω need not be differentiable, and there are no conditions on the constraints other than requiring a feasible initial point $x_0 \in \Omega$, and a nonempty hypertangent cone at a limit point. Constraints are easily handled by the extreme barrier approach;

- There is no synchronization step [19]. Each process takes the current best solution as its starting point without waiting for another process to terminate;

- The choice of sets N_p, $p \in Q_{reg} \cup \{1\}$, is completely flexible and dynamic. Moreover, sets N_p do not have to define a partition of the variables, and some variables can belong to more than one set. An example for a practical strategy deciding the sets N_p is given in Section 5.

This new algorithm is not a particular case of the method in [23], which generalizes many parallel variable decomposition methods, since general constraints are allowed, and f is not assumed to be smooth. PSD-MADS is also different from the recent MoVars algorithm [12], which does require N_p to partition the variables, because it provides a convergence analysis, dynamically changes the sets N_p, and it is an asynchronous parallel method. The next sections describe precisely the role of each process.

3.2 The pollster slave s_1, on $M(\Delta_k^1)$

The pollster slave s_1 has a special role; its set of variables is always fixed to $N_1 = N$, so that it works on the original problem $\mathcal{P}$. Due to its greater impact on the algorithm and to distinguish s_1 from the other slaves, we call it the pollster slave, or simply the pollster.

To reduce the expected high number of evaluations done by all the pollster instances, a single-poll MADS algorithm is used (the poll directions are reduced to a single element), with the conditions that the union of all the normalized directions used throughout the algorithm are dense in the unit sphere, and that the norms of those directions is in the proper relation with the mesh size parameter.

Moreover, the pollster is limited to only one MADS iteration, with no search step and one poll step. It follows that at most one function evaluation will be performed (zero function evaluation if the unique poll trial point is in the cache), and the pollster mesh size parameter Δ_k^1 will not be updated (this is done by the master).

The notation MADS(pollster) or MADS(s_1) refer to the single-poll MADS algorithm performed by the pollster. MADS(pollster) is defined so that its mesh size parameter Δ_k^1 cannot be larger than the master mesh size Δ_k^M at iteration k of PSD-MADS.

The pollster pseudocode is shown in Figure 2. The pollster mesh size is updated by the master. The best obtained solution is described by x_p, which is sent to the master. The convergence analysis in Section 4 is based on the pollster, and on the fact that consecutive runs of MADS(s_1) form a valid single-poll MADS instance on $\mathcal{P}$.

Pollster ($p = 1$)
- given the master data (pollster mesh size Δ_k^1, starting point x_0)
- solve problem $\mathcal{P}$: MADS(pollster)
- – terminate after a single evaluation
- send optimization data to master (pollster solution x_p)

Figure 2: Pseudocode for pollster slave. MADS(pollster) considers all n variables with the single-poll direction $b(\ell)$, and terminates after one iteration.

3.3 The regular slaves s_2 to s_{q-2}, on $M(\Delta_j^p)$

The regular slaves s_p, $p \in Q_{reg}$, work on subsets N_p of N, and use positive spanning sets of directions. The MADS algorithm working on problem $\mathcal{P}_p(x^*)$ and performed by slave s_p is designated by MADS(s_p).

Subproblem $\mathcal{P}_p(x^*)$ is defined as a $|N_p|$ variable problem since all the variables in $N \setminus N_p$ are fixed. Trial points generated by MADS(s_p) are then in $\mathbb{R}^n$, with some coordinates fixed. The values of these fixed coordinates are directly taken from the starting point for MADS(s_p), i.e., x^*, the incumbent solution. The user supplies a parameter, $bbe_{max} > 0$, that indicates the maximum allowed number of black-box calls for the application of MADS to the optimization of a subproblem.

The pseudocode for the regular slaves is shown in Figure 3. MADS(s_p) generates trial points on meshes of sizes Δ_j^p, where j is the iteration counter of the subproblem algorithm. The initial mesh size Δ_0^p for MADS(s_p) is set by the master. The value of the parameter Δ_{min}^p also is supplied by the master, and equals Δ_k^M, where k is the PSD-MADS iteration at which MADS(s_p) started. Finally, we impose that no mesh size for MADS(s_p), $p \in Q_{reg}$, exceeds the PSD-MADS initial mesh size, Δ_0^{user}, provided by the user. MADS(s_p) terminates if bbe_{max} evaluations are made, or if a minimal mesh size Δ_{min}^p is reached. The final mesh size (Δ_{stop}), and the best solution found (x_p), are sent to the master.

The union of all regular slaves MADS(s_p) instances is interpreted as a search step for the total problem single-poll MADS algorithm. This is important to the convergence analysis in Section 4.

Slave s_p ($p \in Q_{reg}$)
- given the master data (initial and minimum mesh sizes Δ_0^p, Δ_{min}^p ; starting solution x_0, set of variables N_p)
- solve subproblem $\mathcal{P}_p(x^*)$: MADS(s_p)
  – terminate when $\Delta_j^p < \Delta_{min}^p$ or after bbe_{max} function evaluations
- send optimization data to master (final mesh size Δ_{stop}, slave solution x_p)

Figure 3: Pseudocode for slaves processes. Does not include pollster slave, which is specifically described in Figure 2.

3.4 The cache server ($(q-1)^{\text{th}}$ process)

The cache server is a specialized process that simply memorizes all evaluated points. Each time a process generates a trial point, the cache server is interrogated in case this

point has already been evaluated, to avoid unnecessary expensive functions evaluations. The cache server allows the global availability of any improvement made by any slave. This is interpreted in Section 5 as a search step (the *cache search*) by the regular slaves on their subproblems.

3.5 The master (q^{th} process)

The master process coordinates the work of the $q-2$ slaves. It waits for slave results, updates data, and assigns work to slaves. It only evaluates the black-box functions at the starting point x_0.

The master process provides the master mesh size Δ_k^M at iteration k of PSD-MADS, which is the link between the mesh sizes Δ_k^1 and Δ_j^p on which the different MADS(s_p), $p \in Q_{reg}$ work. The initial master mesh size $\Delta_0^M = \Delta_0^{user}$ is set by the user.

The master process updates the pollster mesh size Δ_k^1, after a pollster instance terminates. If no improvement is made by any slave s_1 to s_{q-1} during iteration k, the iteration is a failure and the pollster mesh size is reduced. If the iteration succeeds, then the pollster mesh size is increased. In all cases, the pollster mesh size is smaller than the master mesh size (2). The value of the pollster mesh size is also kept less than or equal to Δ_0^{user}.

For all regular slaves s_2 to s_{q-1}, the minimal mesh size Δ_{min}^p is set to the current value of Δ_k^M. This, as explained in more detail in the convergence analysis, leads to the fact that at iteration k of PSD-MADS, no regular slave can generate trial points on meshes finer than $M(\Delta_k^M)$, and that all the slaves work in fact on the pollster mesh of size Δ_k^1.

The master process pseudocode is described in Figure 4, and the pollster mesh size update is detailed in Figure 5. The pseudocode for the master process implies that when the master mesh size is updated, it is always possible to find an integer $\alpha_k \in [0; w^+]$ such that $\tau^{\alpha_k}\Delta_k^1 \leq \min_{p\in Q_{reg}} \Delta_{min}^p$. The next proposition shows that $\alpha_k = 0$ is always a candidate.

Proposition 3.2 *At iteration k of the PSD-MADS algorithm, there exists a nonnegative integer α_k such that $\tau^{\alpha_k}\Delta_k^1 \leq \min_{p\in Q_{reg}} \Delta_{min}^p$.*

Proof. At iteration 0, $\Delta_0^1 = \Delta_0^M = \Delta_0^{user} = \min_{p\in Q_{reg}} \Delta_{min}^p$ so $\alpha_0 = 0$, and therefore it exists. Then $\Delta_1^M = \Delta_0^{user}$ and $\min_{p\in Q_{reg}} \Delta_{min}^p$ at iteration 1 is equal to Δ_0^{user}. Figure 5 ensures that Δ_1^1 is bounded above by Δ_0^{user}, and therefore $\alpha_1 = 0$ is a possible value.

Suppose, by way of induction, that for some $k \geq 2$, the proposition is true at iteration $k-1$. It follows that $\Delta_k^M = \tau^{\alpha_{k-1}}\Delta_{k-1}^1 \leq \min_{p\in Q_{reg}}$, and as it corresponds to new values for Δ_{min}^p, $p \in Q_{reg}$, the new smaller possible value of $\min_{p\in Q_{reg}} \Delta_{min}^p$ at iteration k remains Δ_k^M. The largest value that Δ_k^1 may take is also Δ_k^M, which shows $\alpha_k = 0$ validates the result. ∎

This proof allows all values of α_k to be zero, but in practice, non-zero values are

Master

[0] initializations

$x^* \leftarrow x_0 \in \Omega$, $\Delta_0^1 \leftarrow \Delta_0^M \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$

start MADS(pollster) with (Δ_0^{user}, x_0)

for all ($p \in Q_{reg}$)

construct the set of indices N_p and set $\Delta_{min}^p \leftarrow \Delta_0^M$

start MADS(s_p) with (Δ_0^{user}, Δ_{min}^p, x_0, N_p)

[1] iterations

given values from a slave s_p (Δ_{stop}, x_p)

if $\big(f(x_p) < f(x^*)\big)$ (success)

$x^* \leftarrow x_p$

if $(p = 1)$ (pollster, Δ_{stop} corresponds to Δ_k^1)

$\Delta_{k+1}^M \leftarrow \tau^{\alpha_k}\Delta_k^1 \leq \min_{p \in Q_{reg}} \Delta_{min}^p$ with $\alpha_k \in [0; \omega^+]$, $\omega^+ \in \mathbb{N}$

$\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$ (detailed in Figure 5)

$k \leftarrow k + 1$

start MADS(pollster) with (Δ_k^1, x^*)

else (regular slave)

construct N_p

$\Delta_{min}^p \leftarrow \Delta_k^M$

$\Delta_0^p \leftarrow \tau^{\gamma}\Delta_{stop}$ with $\gamma \in \mathbb{Z}$ and so that $\Delta_k^M \leq \Delta_0^p \leq \Delta_0^{user}$

start MADS(s_p) with (Δ_0^p, Δ_{min}^p, x^*, N_p)

goto [1]

Figure 4: Pseudocode for master process. Δ_k^M and Δ_k^1 are the master and pollster mesh sizes at iteration k, and Δ_{stop} the last mesh size of a slave s_p. If $p = 1$, $\Delta_{stop} = \Delta_k^1 \leq \Delta_k^M$, and else $\Delta_{stop} \geq \Delta_k^M$. The master evaluates the black-boxes just once for x_0.

pollster mesh size update $\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$

if (iteration success)

$\omega_k = \alpha_k \in [0; \omega^+]$, $\omega^+ \geq 0$ $\big(\Delta_{k+1}^1 \leftarrow \Delta_{k+1}^M\big)$

$\big($pollster mesh size increase, $\Delta_{k+1}^1 \geq \Delta_k^1\big)$

else

$\omega_k \in [\omega^-; -1]$, $\omega^- \leq -1$

$\big($pollster mesh size decrease, $\Delta_{k+1}^1 < \Delta_k^1\big)$

Figure 5: Update of the next pollster mesh size Δ_{k+1}^1. In any case, the pollster mesh size verifies $\Delta_k^1 \leq \Delta_k^M$.

likely. For example, if iteration 1 failed and $\Delta_1^1 = \Delta_0^{user}$, then the following mesh updates are possible: $\Delta_2^M \leftarrow \Delta_0^{user}$ ($\alpha_1 = 0$) and $\Delta_2^1 \leftarrow \Delta_0^{user}/4$. $\min_{p \in Q_{reg}} \Delta_{min}^p$ is still equal to Δ_0^{user} at iteration 2, and so α_2 can be either 0 or 1.

4 Convergence analysis of PSD-MADS

It is shown here that the entire algorithm may be interpreted as a single-poll MADS algorithm applied to the original problem $\mathcal{P}$ and that conditions are met so that the main convergence results from [8] hold. These conditions are that the regular slaves generate a finite number of trial points lying on the the pollster mesh, and that all these trial points can be interpreted as a search step with the pollster slave providing the poll step. This is detailed in Figure 6, and we refer to it as the *apparent pollster algorithm*. This algorithm is another way of interpreting the PSD-MADS algorithm described by the pseudocodes in Figures 2, 3, 4, and 5. Iteration k of the apparent pollster algorithm corresponds to the iteration k of PSD-MADS (used by the master process), and the notions of iteration success and failure remain the same.

Apparent pollster algorithm

[0] Initializations

$x_0 \in \Omega$, $\Delta_0^M \leftarrow \Delta_0^1 \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$

[1] Poll and search steps

Search step (by other slaves, opportunistic)

ask cache server for $x_s \in M(\Delta_k^M) \subseteq M(\Delta_k^1)$

Single-poll step

construct and evaluate $P_k = \{x_{poll}\} \subseteq M(\Delta_k^1)$

[2] Updates

determine type of success of iteration k

$\Delta_{k+1}^1 \leftarrow \tau^{\omega_k} \Delta_k^1$ (cannot be larger than Δ_{k+1}^M)

$x_{k+1} \leftarrow (x_s$ or x_{poll} or $x_k)$

$k \leftarrow k + 1$

goto [1]

Figure 6: Detailed pseudocode of the apparent pollster algorithm, the algorithm from the point of view of the pollster slave. At every moment, a finite number of $M(\Delta_k^1)$ points are evaluated in parallel by other slaves. These evaluations are considered within the opportunistic search step. Δ_k^M is updated by the master after the poll step.

The convergence analysis in this section proves that the apparent pollster algorithm is a single-poll MADS algorithm with the following components:

- A search step performed by regular slaves $s_2, s_3, ..., s_{q-2}$ on mesh coarseness larger than or equal to Δ_k^M;
- A poll step at iteration k (the same k used by the master process in Figure 4) performed by one call to the pollster slave s_1 on a mesh of size $\Delta_k^1 \leq \Delta_k^M$;
- A mesh update performed by the master process with $\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$ and the integer $\omega_k \in \begin{cases} [0;\omega^+] & \text{iteration success} \\ [\omega^-;-1] & \text{iteration failure.} \end{cases}$

The master mesh size parameter Δ_k^M at iteration k is the link described by inequalities (2) between the mesh size of MADS(pollster) and the different mesh sizes of MADS(s_p). It is updated by the master with the MADS(pollster) mesh (via $\Delta_{stop} = \Delta_k^1$), in such a way that, at every iteration k of the apparent pollster algorithm, Δ_k^1 satisfies $\Delta_k^1 \leq \Delta_k^M$. This Δ_k^M updated by the master in the apparent pollster algorithm occurs when the mesh size Δ_k^1 is updated, and while its value does not change during the poll step, it can possibly be updated during the search step since that is performed in parallel. This possible change of the Δ_k^M value within the search step of the apparent pollster algorithm is governed by the fact that Δ_k^M cannot be exceeded by any regular slave mesh size ($\Delta_k^M \leq \min_{p \in Q_{reg}} \Delta_{min}^p$).

To show that the apparent pollster algorithm is a valid single-poll MADS algorithm applied to the original problem $\mathcal{P}$, and that the convergence conditions of [8] hold, the search trial points, whose evaluations are performed at any time in parallel by the other slaves, must remain finite in number and on the current pollster mesh at iteration k, Δ_k^1. This will be shown via the following propositions.

Proposition 4.1 *The mesh size parameter at iteration j of the MADS algorithm performed by a slave s_p, $p \in Q_{reg}$, on a subproblem $\mathcal{P}_p(x^*)$, satisfies $\Delta_j^p = \tau^{-\eta_j}\Delta_0^{user}$ for some integer $\eta_j \geq 0$. This can be extended to the pollster slave at iteration k with $\Delta_k^1 = \tau^{-\eta_k}\Delta_0^{user}$.*

Proof. We first show that the proposition is true for the first optimization subproblem solved by a regular slave s_p, $p \in Q_{reg}$. The initial mesh size parameter used for this MADS instance is Δ_0^{user}, and with the standard MADS mesh update rules, at iteration j, $\Delta_j^p = \tau^{\omega_{j-1}}\Delta_{j-1}^p$ =... $= \tau^{\sum_{i=0}^{j-1}\omega_i}\Delta_0^{user}$. Then $\eta_j = -\sum_{i=0}^{j-1}\omega_i \geq 0$ because no mesh size can be larger than Δ_0^{user}.

Suppose now that the proposition is true for the r^{th} MADS instance performed by s_p. In particular, the last mesh size parameter of this instance can be written $\Delta_{stop} = \tau^{-\eta_{stop}}\Delta_0^{user}$ where η_{stop} is a nonnegative integer. From the algorithm described in Figure 4, the first mesh size parameter of the $(r+1)^{\text{th}}$ MADS instance performed by s_p is $\Delta_0^p = \tau^\gamma\Delta_{stop}$ with $\gamma \in \mathbb{Z}$. Then at iteration j of the $(r+1)^{\text{th}}$ instance, $\Delta_j^p = \tau^{\sum_{i=0}^{j-1}\omega_i}\Delta_0^p$

and $\eta_j = -\sum_{i=0}^{j-1} \omega_i - \gamma + \eta_{stop} \geq 0$ because $\Delta_j^p \leq \Delta_0^{user}$. The proposition can be extended to the pollster slave with the same induction proof on k. ∎

Proposition 4.2 *At iteration k of PSD-MADS, and at iteration j of the MADS algorithm performed by s_p ($p \in Q_{reg}$) on a subproblem $\mathcal{P}_p(x^*)$, there exists a nonnegative integer β_j such that $\Delta_j^p = \tau^{\beta_j} \Delta_k^M$.*

Proof. From the algorithm in Figure 4, the master mesh size parameter, at iteration k of PSD-MADS, can be written $\Delta_k^M = \tau^{\alpha_k - 1} \Delta_{k-1}^1$ with $\alpha_{k-1} \in \mathbb{N}$, and $\Delta_{k-1}^1 = \tau^{-\eta_{k-1}} \Delta_0^{user}$, with $\eta_{k-1} \in \mathbb{N}$, from Proposition 4.1. From the same proposition, the mesh size parameter at iteration j of MADS(s_p), $p \in Q_{reg}$, can be written $\Delta_j^p = \tau^{-\eta_j} \Delta_0^{user}$, $\eta_j \in \mathbb{N}$. Then $\Delta_j^p = \tau^{\beta_j} \Delta_k^M$ with $\beta_j = \eta_{k-1} - \eta_j - \alpha_{k-1}$. The minimal mesh size parameter Δ_{min}^p considered by MADS(s_p) corresponds to Δ_i^M where $i \leq k$ is an anterior iteration of PSD-MADS. The current value of Δ_k^M was chosen to be smaller than $\min_{p \in Q_{reg}} \Delta_{min}^p \leq \Delta_i^M$. Then, $\Delta_k^M \leq \Delta_i^M \leq \Delta_j^p$ and β_j is a nonnegative integer. ∎

An immediate corollary, with Hypothesis 3.1, is that at iterations k of PSD-MADS and j of MADS(s_p), $p \in Q_{reg}$, $M(\Delta_j^p) \subseteq M(\Delta_k^M)$.

Proposition 4.3 *At iteration k of PSD-MADS, every trial point generated by the MADS algorithm performed by s_p, $p \in Q_{reg}$, on any subproblem $\mathcal{P}_p(x^*)$, lies on the pollster mesh $M(\Delta_k^1)$.*

Proof. From the algorithm in Figure 4, the pollster and master mesh size parameters at iteration k of PSD-MADS are linked with $\Delta_k^M = \tau^{\alpha_k} \Delta_k^1$, $\alpha_k \in \mathbb{N}$. With Hypothesis 3.1 and Proposition 4.2, at iteration j of MADS(s_p), $M(\Delta_j^p) \subseteq M(\Delta_k^M) \subseteq M(\Delta_k^1)$, meaning that all trial points of MADS(s_p), already lying on $M(\Delta_j^p)$, lie on $M(\Delta_k^1)$. ∎

This series of propositions ensures that all the trial points of the search step of the apparent pollster at iteration k, performed in parallel by regular slaves, lie on the current pollster mesh Δ_k^1. In addition, their number remains finite as the time between two iterations, corresponding to a single-point poll, is finite (with the hypothesis that black-boxes evaluate, or are terminated to return ∞, in finite time). The PSD-MADS algorithm, viewed from the perspective of the pollster slave, thus executes a valid single-poll MADS search, and the main convergence results of [8] remain valid. Let $\hat{x}$ be the limit of a subsequence of PSD-MADS incumbents at unsuccessful iterations, then

- If f is Lipschitz near $\hat{x} \in \Omega$, then the Clarke derivative satisfies $f^\circ(\hat{x}; v) \geq 0$ for all $v \in T_\Omega^H(\hat{x})$, the hypertangent cone to Ω at $\hat{x}$;
- In the unconstrained case and if f is strictly differentiable at $\hat{x}$, $\nabla f(\hat{x}) = 0$.

As mentioned in Section 2.3, the fact that the single-poll version of MADS is used sacrifices the zero'th order result of [8], i.e., $\hat{x}$ cannot be said to to be the limit of local optima on meshes that get infinitely fine.

5 A practical implementation of PSD-MADS

This section proposes a practical implementation of the PSD-MADS algorithm described in Section 3 based on the LTMADS implementation proposed in [8] and summarized in Section 2.4. An illustrative example and some numerical tests complete the implementation description.

5.1 PSD-MADS implementation

Verification of Hypothesis 3.1

The above convergence analysis relies on Hypothesis 3.1. An easy way to satisfy this hypothesis is to simply choose τ to be an integer. Indeed, consider the mesh point $x \in M(\Delta)$, and mesh size $\Delta \in \mathbb{R}$. From the mesh definition (1), x can be written as $y + \Delta \sum_{i=1}^{n_D} z_i d_i$ where y belongs to V, the set of currently evaluated points, and the z_i are nonnegative integers. Now, if $\Delta' = \tau^\omega \Delta$ where $\omega \in \mathbb{N}$ and $1 \leq \tau \in \mathbb{N}$, then x can be rewritten as $x = y + \Delta' \sum_{i=1}^{n_D} \tau^\omega z_i d_i$. It follows that, $\tau^\omega z_i \in \mathbb{N}$, $i = 1, 2, ..., n_D$, and therefore $x \in M(\Delta')$. We have shown that $M(\Delta) \subseteq M(\Delta')$ and thus, Hypothesis 3.1 is satisfied. In the proposed PSD-MADS implementation, the same LTMADS fixed value of $\tau = 4$ is used.

Directions used by the pollster

The LTMADS direction $b(\ell)$ is used in the single-poll MADS algorithm executed by the pollster slave. The union of normalized directions $b(\ell)$, $\ell = 1, 2, ...$, is dense in the unit sphere with probability one, and MADS(pollster) with the $b(\ell)$ direction respects the conditions for a valid single-poll MADS algorithm.

Sets N_p of subproblems variables

We take the sets N_p, $p \in Q_{reg}$, to be randomly generated by the master using an uniform distribution before each subproblem parameters are sent to a regular slave process. In order to keep an easy parametrization of this PSD-MADS implementation, the number of variables for each subproblem is fixed throughout the entire algorithm, $|N_2| = |N_3| = ... = |N_q| = ns$, where ns is a parameter chosen by the user (recall that for the pollster, $N_1 = N$). Furthermore, when MADS(s_p), $p \in Q_{reg}$, succeeds in

improving the incumbent, the same set N_p is kept for the next run performed by the slave s_p.

Mesh update rules

The mesh directions of definition (1) are the standard LTMADS $2n$ directions, $D = [-I_n \ I_n]$. The following mesh size parameter updates are in accordance with the LT-MADS mesh update rules:

- **Regular slaves mesh size Δ_j^p (at iteration j of MADS(s_p), $p \in Q_{reg}$):** after an iteration fails, the mesh size is updated with $\Delta_{j+1}^p \leftarrow \Delta_j^p/4$ ($\omega_j = -1$ in Figure 1). If a poll step is successful, $\Delta_{j+1}^p \leftarrow 4\Delta_j^p$ ($\omega_j = 1$). In the next search step, if a successful point is found in the cache server, set $\Delta_{j+1}^p \leftarrow 4\Delta_{cache}$ where Δ_{cache} is the mesh size used to generate this point. Equation (3) summarizes these updates:

$$\Delta_{j+1}^p \leftarrow \begin{cases} \min\{\Delta_0^{user}, 4\Delta_j^p\} & \text{poll success} \\ \min\{\Delta_0^{user}, 4\Delta_{cache}\} & \text{cache search success} \\ \Delta_j^p/4 & \text{iteration failure.} \end{cases} \tag{3}$$

  If $\Delta_{j+1}^p < \Delta_{min}^p$, or if the number of new function evaluations exceeds bbe_{max}, MADS(s_p) terminates and communicates $\Delta_{stop} = \Delta_j^p$ to the master. The next optimization performed by this slave will start with an initial mesh size parameter Δ_0^p equal to $4^\gamma\Delta_{stop}$, with $\gamma = 1$ if at least one success was achieved since the beginning of the current optimization (even by another slave), or else $\gamma = -1$. However, this may lead to a value smaller than $\Delta_{min}^p = \Delta_k^M$, and in this case, set $\Delta_0^p \leftarrow \Delta_k^M$.

  The Δ_0^p choice for the next MADS(s_p) is summarized by:

$$\Delta_0^p \text{ (next MADS}(s_p)) \leftarrow \begin{cases} \min\{\Delta_0^{user}, 4\Delta_{stop}\} & \text{success} \\ \max\{\Delta_k^M, \Delta_{stop}/4\} & \text{else.} \end{cases} \tag{4}$$

- **Master mesh size Δ_k^M at iteration k of PSD-MADS:** the update of the master mesh size is performed by the master after a pollster instance terminates. Δ_{k+1}^M is bounded below by the mesh size parameter of the terminated pollster, Δ_k^1, and above by the minimum of all Δ_{min}^p values currently used by regular slaves. These Δ_{min}^p values correspond to previous master mesh sizes.

  It would be possible to choose the parameter α_k in Figure 4 at each update so that Δ_{k+1}^M is fixed to Δ_0^{user}, with α_k equal to the η_k from Proposition 4.1. However, such a strategy would not be efficient as regular slaves would always generate trial points on the same mesh $M(\Delta_0^{user})$. The master mesh size has then to be reduced somehow through the PSD-MADS evolution. However, it should not be reduced

too rapidly, or the algorithm would progress slowly, or even terminate prematurely in practice.

We propose the following strategy: from Figure 4, Δ_k^M is updated by $\Delta_{k+1}^M \leftarrow 4^{\alpha_k}\Delta_k^1$ with $\alpha_k \in \mathbb{N}$, and from Proposition 4.1, $\Delta_k^1 = 4^{-\eta_k}\Delta_0^{user}$ with some $\eta_k \in \mathbb{N}$. If iteration k succeeded, set $\alpha_k = \eta_k = \log_4(\Delta_0^{user}/\Delta_k^1)$ (maximal Δ_k^M increase), and else, $\alpha_k = \eta_k - \lfloor(\eta_k+1)/3\rfloor$ (attenuated Δ_k^M increase). In both cases, if Δ_{k+1}^M is greater than at least one of the regular slaves mesh size Δ_{min}^p, then Δ_{k+1}^M is set to the least Δ_{min}^p values. This can be summarized by the following:

$$\Delta_{k+1}^M \leftarrow \begin{cases} \min\left\{\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration success} \\ \min\left\{4^{-\lfloor(\eta_k+1)/3\rfloor}\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration failure.} \end{cases} \tag{5}$$

For example, if $\Delta_0^{user} = \Delta_{min}^p = 1$ for each $p \in Q_{reg}$ and if the pollster instance fails with a pollster mesh size of $\Delta_k^1 = 1/16$, then the master mesh size Δ_{k+1}^M is set to $1/4$ ($\eta_k = 2$, $\alpha_k = 1$; this is what happens at time $t = t_3$ in the example described in Section 5.2) .

- **Pollster mesh size Δ_k^1 at iteration k of PSD-MADS:** in the case of an iteration success, Δ_{k+1}^1 is set to Δ_{k+1}^M ($\omega_k = \alpha_k \in \mathbb{N}$), or else $\Delta_{k+1}^1 = \Delta_k^1/4$ ($\omega_k = -1$):

$$\Delta_{k+1}^1 \leftarrow \begin{cases} \Delta_{k+1}^M = \min\left\{\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration success} \\ \Delta_k^1/4 & \text{iteration failure.} \end{cases} \tag{6}$$

MADS parameters for MADS(s_p), $p \in Q_{reg}$

The regular slaves $p \in Q_{reg}$ solve MADS(s_p) using the standard MADS $2|N_p|$ directions. All polls are opportunistic, meaning that a subproblem optimization terminates as soon as a better point is found. The one point dynamic search strategy of [8] is also performed: it consists, after a successful poll step, in evaluating, within a single-point search, the black-box functions at a mesh point located further along the same successful direction.

In addition to the poll and the one-point dynamic search, MADS(s_p) performs a specialized search step, which simply consists in querying the cache server for the best available feasible point. This special search step generates no additional function evaluation and allows every regular slave to know the best points eventually obtained by other slaves. Note that this search step has no obligation to give a point lying on the current mesh of MADS(s_p), but this does not influence the convergence analysis as it is based on the pollster s_1, and as the point given by this search must come from another slave, thus lying on $M(\Delta_k^M)$.

Practical termination criteria

The regular slaves $p \in Q_{reg}$ terminate MADS(s_p) as soon as the mesh size parameter Δ_j^p drops below $\Delta_{min}^p = \Delta_k^M$ (where k is the PSD-MADS iteration at which MADS(s_p) was started), or after a finite number of bbe_{max} black-box function evaluations are made. The PSD-MADS algorithm is stopped after an overall limit of bbe_{max}^{global} black-box evaluations is reached.

The PSD-MADS implementation described above is illustrated in the next section, where an example of a few steps is presented (Figure 7).

5.2 A detailed PSD-MADS illustrative example

We consider a problem with $n = 4$ variables and $q = 5$ processes (one pollster, two regular slaves, one master, and one cache). The two regular slaves have a limit of $bbe_{max} = 2$ evaluations, and their sets of variables are of cardinality $ns = 2$. The initial (and maximal) mesh size value is $\Delta_0^{user} = 1$.

The progress of the four processes over time from iteration $k = 4$ to the end of $k = 6$ is illustrated in Figure 7. The cache server is not represented in the figure. The grayed rectangles illustrate the black-box evaluations and their widths represent their different running times to return a value. Arrows represent communications between processes.

The time t_1 corresponds to the beginning of iteration $k = 5$. The incumbent solution at the start of iteration $k = 5$ is $x^* = [10\ 10\ 10\ 10]^T$ with $f(x^*) = 10$ and the current master mesh size is $\Delta_5^M = 1$. This information appears in the figure in the master's section.

The notation y^i is used for the ith poll trial point of one instance of MADS(s_p), $p \in \{1, 2, 3\}$ solved by a slave. The pollster evaluates the black-boxes at time t_1 at the poll point $y^1 = [10.0625\ 10\ 10\ 10]^T$ with pollster mesh size $\Delta_5^1 = 1/16$.

At t_2, slave s_3 terminates and communicates to the master: the incumbent is not modified. The stopping criteria Δ_{min}^p value of slave s_3 is set to the current master mesh size value Δ_5^M. The coordinates of the regular slave trial points marked by stars indicate that these coordinates are fixed to the ones of the poll center. For example, the two trial poll points of slave s_3 at time t_2 and t_3 are $[10^*\ 11\ 10\ 10^*]^T$ and $[10^*\ 10\ 11\ 10^*]^T$ with poll center $x_0 = x^* = [10\ 10\ 10\ 10]^T$, $N_2 = \{2, 3\}$, and with a mesh size parameter $\Delta_0^3 = 1$.

At time t_3, the pollster returns information to the master, and iteration 5 is declared to have failed. The master mesh size is set to $\Delta_6^M = 1/4$, with $\eta_5 = 2$ and $\alpha_5 = 1$, according to (5) (attenuated master mesh size increase). Also, the pollster mesh size is reduced to $\Delta_6^1 = \Delta_5^1/4 = 1/64$ (6).

At t_4, the slave s_2 improves the current incumbent solution, and the mesh size is increased: $\Delta_1^2 \leftarrow 1 = \Delta_0^{user}$ (3). Since MADS(s_2) is opportunistic, it begins a new

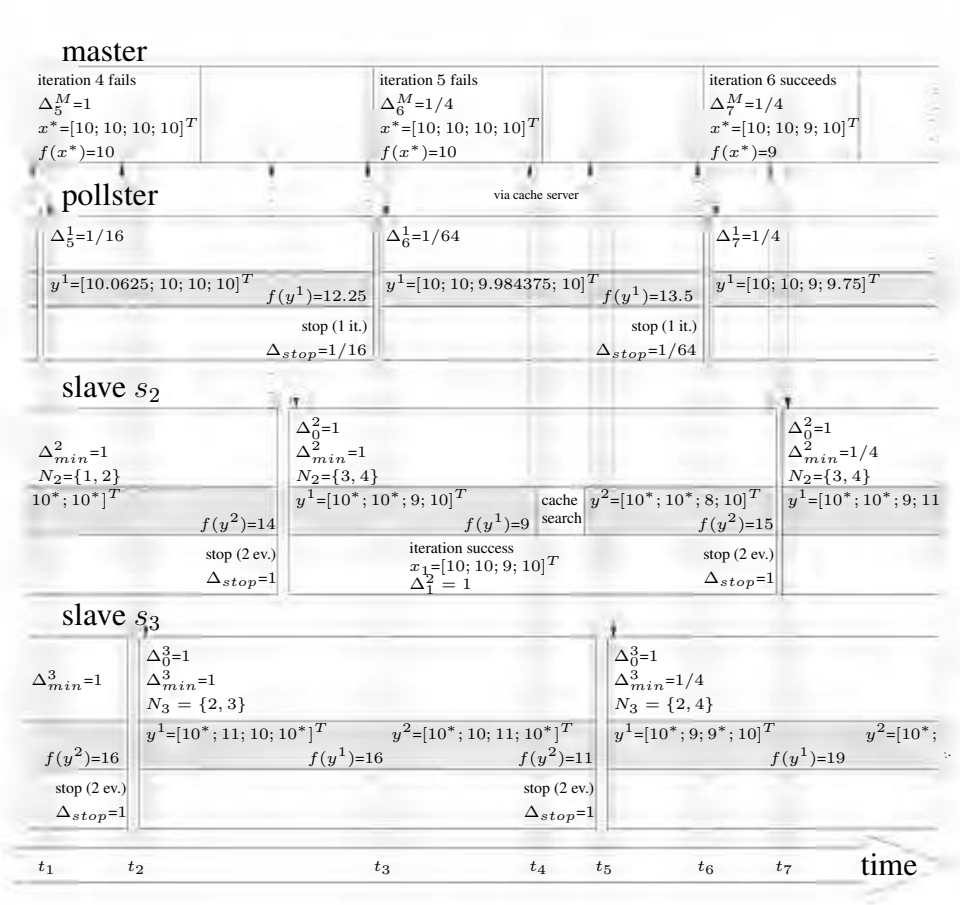

Figure 7: A PSD-MADS illustrative example.

iteration with a cache search.

At t_5, slave s_3 terminates and communicates to the master: the incumbent is not modified. MADS(s_3) starts with $N_3 = \{2, 4\}$, $\Delta_0^3 = 1$, and $\Delta_{min}^3 = 1/4$, because a new incumbent was produced by s_2 since s_3 last communicated with the master ("success" in equation (4)).

At time t_6, the pollster returns information to the master, and iteration 6 is declared to be successful. The maximal increase is performed for Δ_7^M, which is set to $\min_{Q_{reg}} \Delta_{min}^p = 1/4$ (5), and $\Delta_7^1 \leftarrow \Delta_7^M$.

Finally, at t_7, since MADS(s_2) was successful, the new instance of MADS(s_2) keeps the same free variables $N_2 = \{3, 4\}$.

5.3 Numerical experiments

The PSD-MADS implementation described in Section 5 is tested here, on two different problems. The implementation of MADS used to optimize subproblems is the research

version of the NOMAD C++ code [5]. The parallel master/slaves paradigm is achieved with MPI with $q = 6$ or 14 processes.

PSD-MADS is compared to three other parallel algorithms, on the same number q of processes: first, the pGPS method described in [17], which corresponds to the unmodified GPS method where evaluation are made in parallel. Then pMADS, which is the trivial adaptation of pGPS that uses MADS instead of GPS. pGPS and pMADS are both synchronous parallel algorithms. The third method is APPS version 5.0 [25, 33], the only available GPS asynchronous parallel algorithm.

The first problem (referred as Problem A) considered for the tests is the G2 example from [28]. It has been chosen for its difficulty and for its variable size: our tests involve $n = 20, 50, 250$ and 500 variables. Problem A is written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4 x_i - 2 \prod_{i=1}^{n} \cos^2 x_i}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

$$s.t. \begin{cases} -\prod_{i=1}^{n} x_i + 0.75 \leq 0 \\ \sum_{i=1}^{n} x_i - 7.5n \leq 0 \\ 0 \leq x_i \leq 10, \ i = 1, 2, ..., n. \end{cases}$$

The problem is treated as a black-box one, and an upper limit of $100n$ function evaluations is imposed. The feasible starting point for all methods is the center of the bound constrained domain $x_0 = [5\ 5\ ...\ 5]^T \in \Omega$. The best known value from [28], for $n = 20$, is $f(x) = -0.803619$. In [28], various genetic algorithms gave good solutions, after several hundred thousand of evaluations. Here, after a maximum of 2000 evaluations, PSD-MADS achieved $f(x) \simeq -0.76$.

The second test problem (Problem B) was designed for the MoVars algorithm [12]. It has $n = 60$ variables and one constraint with two different versions: $G \geq 250$, or $G \geq 500$ (see [12] for a more complete description). An infeasible starting point is provided in [12], but cannot be used in the present work since constraints are treated with the extreme barrier approach. The feasible starting points considered here for the two versions of Problem B have been obtained by minimizing the constraint violation $\max\{0, (250 - G)^2\}$ or $\max\{0, (500 - G)^2\}$, from the starting point of [12], with the pMADS algorithm. These optimizations required 3 evaluations for $G \geq 250$, with the resulting feasible point x_0 giving $f(x_0) = 3678.35$ and 74 evaluations for $G \geq 500$, and $f(x_0) = 3014$. These evaluations costs are considered in Figure 9. The feasible starting points, our source code for Problem B, and our best points are available on the website `www.gerad.ca/Charles.Audet`. Our results for Problem B are not compared with the MoVars algorithm results because numerical values are not given

in [12]. The best solutions found gave $f(x) = 13.565$ for $G \geq 250$, and $f(x) = 245.866$ for $G \geq 500$.

The various results of this section are measured considering two quantities: z represents the best value of the objective function of problem $\mathcal{P}$, and bbe, the total number of black-box evaluations. One evaluation is counted for the calls to both the objective f and constraints of Ω.

The most representative cost of a black-box algorithm is the number of black-box evaluations. For the same reason, no speedup curves are given and q is kept constant for each problem ($q = 14$ for Problem A and $q = 6$ for Problem B). The PSD-MADS method was not conceived in order to reduce the time to obtain a solution. Instead, we seek to obtain better solutions than a non-decomposing algorithm for problems with a large number of variables ($20 \leq n \leq 500$).

For all our tests, the termination criteria is the maximum total number of black-box evaluations, which is $bbe_{max}^{global} = 100n$ for Problem A and $bbe_{max}^{global} = 3000$ for Problem B (as in [12]).

The initial (and maximal) mesh size parameter is $\Delta_0^{user} = 2$ for Problem A. For Problem B, due to scaling reasons, the value of Δ_0^{user} differs for each variable and is set to be 0.2 times the range of the variables (i.e $\Delta_0^{user} = 0.3$ for the 15 first variables, 0.35 for the next 30 variables, and 0.44 for the last 15 variables). These values has been decided empirically to give good results with standard MADS and APPS runs. The linear nature of the second constraint of Problem A is exploited by APPS. Since PSD-MADS and pMADS involve randomness in the polling directions, 30 runs are made for each test Parallel execution of pGPS and APPS can affect their determinism. However, this effect was ignored and one run was performed for each test.

To measure the quality of the solutions found, the best (z_{best}), worst (z_{worst}), and average (z_{avg}) values of z after the $100n$ evaluations, are reported. Another measure is S_{avg}, representing the area between a curve z v.s bbe and the line $z = -0.8$ for Problem A (no run gave $z < -0.8$), and $z = 0$ for Problem B. Best runs are obtained with small values for all these quantities.

PSD-MADS was tested on Problem A with $n = 20$ and 50 by varying bbe_{max}, the maximum number of black-box evaluations for each regular subproblem, and ns the number of variables in each subproblem. The number of processes has been set to $q = 14$, in order to fully exploit 12 processors. Good results were obtained by setting $bbe_{max} = 10$, and having the regular slaves working on small dimensional subspaces $ns = 2$. These values are kept for $n > 50$. For Problem B, bbe_{max} is kept to 10. The best results have been obtained by distributing the 60 variables amongst 3 regular slaves with $q = 6$ and $ns = 20$.

Table 1 and Figures 8 and 9 summarize the numerical results. For all instances of Problem A, APPS outperforms pGPS, but it does not do as well as PSD-MADS. In the three larger instances of Problem A, the worst f value produced by PSD-MADS is always better than all the other methods f values. For Problem B, pGPS outperforms

APPS, and better results are obtained with pMADS and PSD-MADS, with a small advantage to PSD-MADS. In all the curves in Figures 8 and 9, one can notice that pMADS is always the fastest to descend, but PSD-MADS overtakes it and produces better solutions.

Algorithm	Problem	z_{best}	z_{worst}	z_{avg}	S_{avg}	Problem	z_{best}	z_{worst}	z_{avg}	S_{avg}
pGPS		-0.450	-0.450	-0.450	1,010		-0.279	-0.279	-0.279	3,400
APPS	A	-0.517	**-0.517**	-0.517	806	A	-0.461	-0.461	-0.461	2,443
pMADS	$n = 20$	**-0.775**	-0.434	-0.592	670	$n = 50$	-0.498	-0.430	-0.457	1,939
PSD-MADS		-0.761	-0.430	**-0.666**	**595**		**-0.727**	**-0.528**	**-0.663**	**1,553**
pGPS		-0.089	-0.089	-0.089	18,322		-0.073	-0.073	-0.073	37,395
APPS	A	-0.193	-0.193	-0.193	16,980	A	-0.129	-0.129	-0.129	35,816
pMADS	$n = 250$	-0.449	-0.438	-0.444	9,703	$n = 500$	-0.447	-0.439	-0.443	19,380
PSD-MADS		**-0.698**	**-0.464**	**-0.603**	**8,568**		**-0.688**	**-0.461**	**-0.576**	**17,660**
pGPS		764.741	764.741	764.741	2,731,920		869.559	869.559	869.559	3,552,910
APPS	B	813.237	813.237	813.237	3,906,060	B	1,162.580	1,162.580	1,162.580	4,579,370
pMADS	$G \geq 250$	32.700	317.167	112.522	1,071,870	$G \geq 500$	417.049	948.768	662.841	2,892,140
PSD-MADS		**13.565**	**307.305**	**70.121**	**965,553**		**245.866**	**731.023**	**463.969**	**2,603,480**

Table 1: Numerical Result for Problems A and B: z_{best}, z_{worst} and z_{avg} give information on the 30 runs performed for each PSD-MADS test series, and S_{avg} gives a measure of the area below the curves in Figures 8 and 9. Best values appear in bold.

6 Discussion and possible extensions

This paper introduced PSD-MADS, a new parallel space decomposition technique with less restrictive conditions on the functions to be optimized than usual PSD methods. A convergence analysis is given based on the Clarke calculus and the MADS convergence analysis. A practical implementation is described, with a small number of parameters (bbe_{max} and ns), and very encouraging results have been obtained on a difficult problem from the literature, with up to 500 variables.

We presented a first basic implementation of PSD-MADS. An obvious extension is a strategy to decide on the sets of variables in the subproblems, which is done randomly for these tests. Of course, it is not clear how to do this in general or we would have done it here. However, for some application, the user may have special knowledge that would help in this task. For example, the user might put similarly scaled variables in the same subproblem.

It would also be interesting to incorporate the PVD idea of the "forget-me-not" terms, and allow some basic changes in the subproblems for fixed variables. A third possibility would be to perform some additional search steps in the slave subspaces. Another possible extension would be to reintroduce the synchronization step of the original block-Jacobi method, but without the parallel barrier. This "recomposition" step could be performed in parallel by one of the regular slaves, from a pool of successful points, in

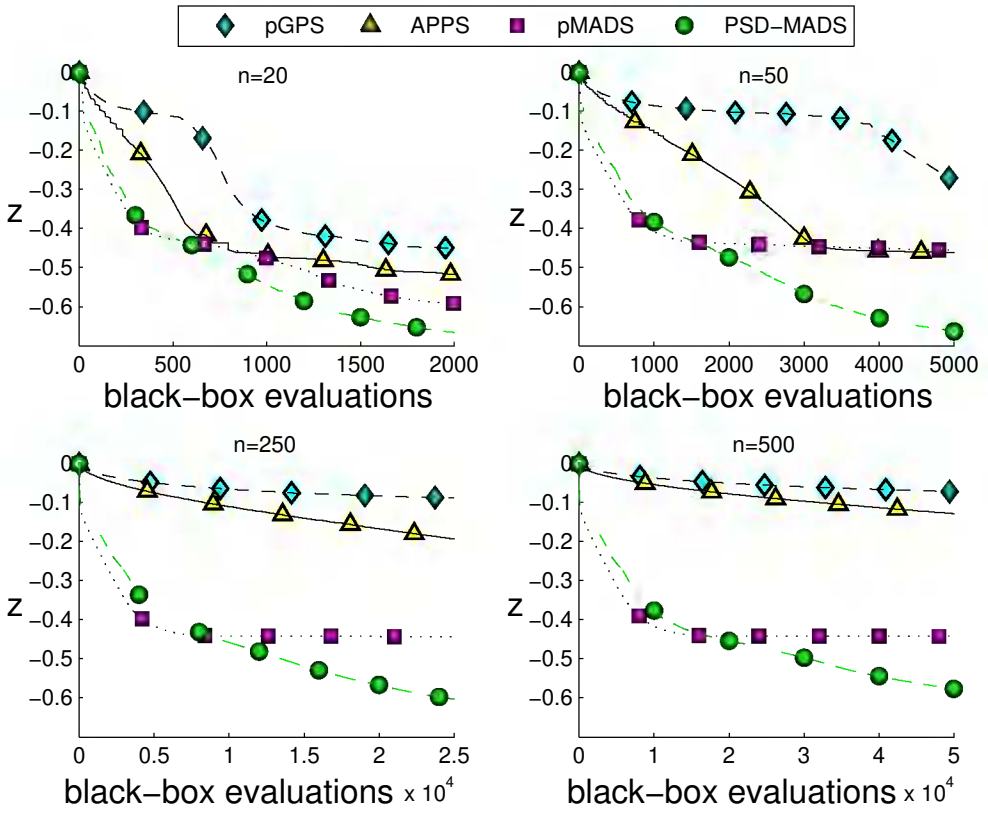

Figure 8: Problem A: graphs of the objective function value v.s the number of evaluations for all test results. PSD-MADS and pMADS plots correspond to average values of the 30 runs performed for each test.

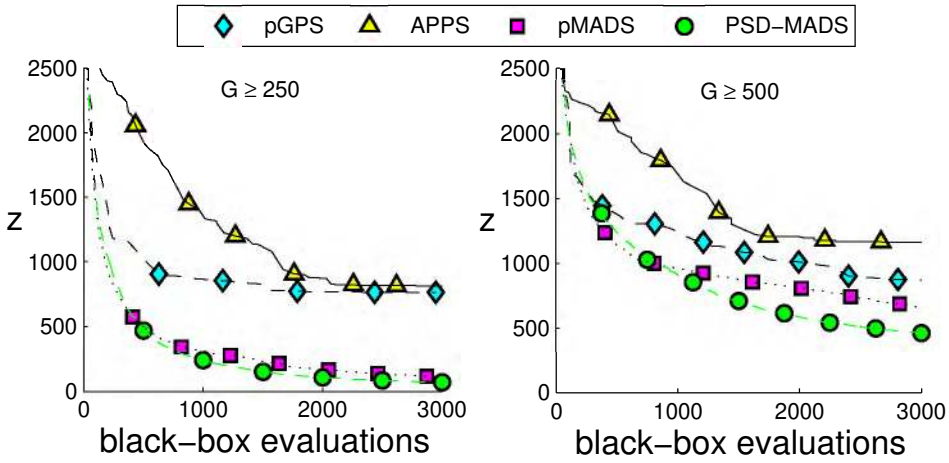

Figure 9: Problem B: graphs of the objective function value v.s the number of evaluations for all test results. PSD-MADS and pMADS plots correspond to average values of the 30 runs performed for each test.

order to create a problem similar to the one in [19]. Finally, constraints of Ω could be treated with the progressive barrier [9], instead of the extreme barrier approach. This would allow for infeasible iterates, including the starting point.

References

[1] M. A. Abramson. Mixed Variable Optimization of a Load-Bearing Thermal Insulation System Using a Filter Pattern Search Algorithm. *Optimization and Engineering*, 5(2):157–177, 2004.

[2] M. A. Abramson and C. Audet. Second-order convergence of mesh-adaptive direct search. *SIAM Journal on Optimization*, 17(2):606–619, 2006.

[3] P. Alberto, F. Nogueira, U. Rocha, and L. N. Vicente. Pattern search methods for user-provided points: application to molecular geometry problems. *SIAM Journal on Optimization*, 14(4):1216–1236, 2004.

[4] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth Optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search. To appear in *Journal of Global Optimization*, DOI: 10.1007/s10898-007-9234-1, 2007.

[5] C. Audet, G. Couture, and J. E. Dennis, Jr. NOMAD project (LTMADS package). Software available at `www.gerad.ca/nomad`.

[6] C. Audet, A. L. Custodio, and J. E. Dennis, Jr. Erratum : Mesh adaptive direct search algorithms for constrained optimization. To appear in *SIAM Journal on Optimization*, 2006.

[7] C. Audet and J. E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[8] C. Audet and J. E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[9] C. Audet and J. E. Dennis, Jr. A MADS Algorithm with a Progressive Barrier for Derivative-Free Nonlinear Programming. Technical Report G-2007-37, Les Cahiers du GERAD, May 2007.

[10] C. Audet and D. Orban. Finding Optimal Algorithmic Parameters Using the Mesh Adaptive Direct Search Algorithm. *SIAM Journal on Optimization*, 17(3):642–664, 2006.

[11] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[12] A. J. Booker, E. J. Cramer, P. D. Frank, J. M. Gablonsky, and J. E. Dennis, Jr. MoVars: Multidisciplinary Optimization Via Adaptive Response Surfaces. AIAA Paper 2007–1927, Presented at the 48th AIAA/ASME/ASCE/AHS/ ASC Structures, Structural Dynamics, and Materials Conference, Honolulu, 2007.

[13] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. W. Moore, and D. B. Serafini. Managing surrogate objectives to optimize a helicopter rotor design – further experiments. AIAA Paper 1998–4717, Presented at the 8th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, 1998.

[14] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, February 1999.

[15] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Wiley, New York, 1983. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

[16] I. D. Coope and C. J. Price. Frame-based methods for unconstrained optimization. *Journal of Optimization Theory and Applications*, 107(2):261–274, 2000.

[17] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, November 1991.

[18] J. E. Dennis, Jr. and Z. Wu. *Sourcebook of parallel computing*, pages 649–670. Parallel continuous optimization. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[19] M. C. Ferris and O. L. Mangasarian. Parallel variable distribution. *SIAM Journal on Optimization*, 4:815–832, 1994.

[20] D.E. Finkel and C.T. Kelley. Convergence analysis of the DIRECT algorithm. Technical Report CRSC-TR04-28, Center for Research in Scientific Computation, 2004.

[21] K. R. Fowler, C. T. Kelley, C. T. Miller, C. E. Kees, R. W. Darwin, J. P. Reese, M. W. Farthing, and M. S. C. Reed. Solution of a Well-Field Design Problem with Implicit Filtering. *Optimization and Engineering*, 5(2):207–234, 2004.

[22] A. Frommer and R. A. Renaut. A unified approach to parallel space decomposition methods. *Journal of Computational and Applied Mathematics*, 110:205–233, 1999.

[23] M. Fukushima. Parallel variable transformation in unconstrained optimization. *SIAM Journal on Optimization*, 8(3):658–672, 1998.

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

[25] G. A. Gray and T. G. Kolda. Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software*, 32(3):485–507, September 2006.

[26] S.-P. Han. Optimization by updated conjugate subspaces. *Numerical Analysis*, 140:82–97, 1986.

[27] R. E. Hayes, F. H. Bertrand, C. Audet, and S. T. Kolaczkowski. Catalytic combustion kinetics: Using a direct search algorithm to evaluate kinetic parameters from light-off curves. *The Canadian Journal of Chemical Engineering*, 81(6):1192–1199, 2003.

[28] A. Hedar and M. Fukushima. Derivative-free filter simulated annealing method for constrained continuous global optimization. *Journal of Global Optimization*, 35(4):521–549, 2006.

[29] P. D. Hough, T. G. Kolda, and V. Torczon. Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, June 2001.

[30] 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, October 1993.

[31] C. T. Kelley. *Iterative Methods for Optimization*. Number 18 in Frontiers in Applied Mathematics. SIAM, Philadelphia, 1999.

[32] M. Kokkolaras, C. Audet, and J. E. Dennis, Jr. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2(1):5–29, 2001.

[33] T. G. Kolda. Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization. *SIAM Journal on Optimization*, 16(2):563–586, 2005.

[34] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Rev.*, 45(3):385–482, 2003.

[35] T. G. Kolda and V. Torczon. Understanding asynchronous parallel pattern search. In G. DiPillo and A. Murli, editors, *High Performance Algorithms and Software for Nonlinear Optimization*, pages 316–335. Kluwer Academic Publishers B.V., 2003.

[36] T. G. Kolda and V. Torczon. On the convergence of asynchronous parallel pattern search. *SIAM Journal on Optimization*, 14(4):939–964, 2004.

[37] R. M. Lewis and V. Torczon. Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, 9(4):1082–1099, 1999.

[38] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.

[39] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124(1–2):191–207, December 2000.

[40] C.-S. Liu and C.-H. Tseng. Parallel synchronous and asynchronous space-decomposition algorithms for large-scale minimization problems. *Comput. Optim. Appl.*, 17(1):85–107, 2000.

[41] O. L. Mangasarian. Parallel gradient distribution in unconstrained optimization. *SIAM Journal Control Optim.*, 33(6):1916–1925, 1995.

[42] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin. Optimal Aeroacoustic Shape Design Using the Surrogate Management Framework. *Optimization and Engineering*, 5(2):235–262, 2004.

[43] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[44] C. J. Price and I. D. Coope. Frames and grids in unconstrained and linearly constrained optimization: A nonsmooth approach. *SIAM Journal on Optimization*, 14(2):415–438, 2003.

[45] C. A. Sagastizábal and M. V. Solodov. Parallel variable distribution for constrained optimization. *Comput. Optim. Appl.*, 22(1):111–131, 2002.

[46] M. V. Solodov. New inexact parallel variable distribution algorithms. *Comput. Optim. Appl.*, 7(2):165–182, 1997.

[47] M. V. Solodov. On the convergence of constrained parallel variable distribution algorithms. *SIAM Journal on Optimization*, 8(1):187–196, 1998.

[48] B. Tang. Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, 88(424):1392–1397, 1993.

[49] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, February 1997.

[50] P. Tseng. Dual coordinate ascent methods for non-strictly convex minimization. *Math. Program.*, 59(2):231–247, 1993.

[51] E. Yamakawa and M. Fukushima. Testing parallel variable transformation. *Comput. Optim. Appl.*, 13(1-3):253–274, 1999.