SIAM J. OPTIM.
Vol. 19, No. 3, pp. 1150–1170

PARALLEL SPACE DECOMPOSITION OF THE MESH ADAPTIVE DIRECT SEARCH ALGORITHM*

CHARLES AUDET†, J. E. DENNIS JR.‡, AND SÉBASTIEN LE DIGABEL†

Abstract. This paper describes a parallel space decomposition (PSD) technique for the mesh adaptive direct search (MADS) algorithm. MADS extends a generalized pattern search for constrained nonsmooth optimization problems. The objective of the present work is to obtain good solutions to larger problems than the ones typically solved by MADS. 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 the advantages and limitations of PSD-MADS.

Key words. parallel space decomposition, mesh adaptive direct search (MADS), asynchronous parallel algorithm, nonsmooth optimization, convergence analysis.

AMS subject classifications. 90C56, 90C30, 65K05, 68W10

DOI. 10.1137/070707518

1. Introduction.

This paper considers optimization problems of the form

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

with the objective function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}\cup\{\infty\}$. Our motivation is to treat $\mathcal{P}$ when n grows large. The feasible region Ω is assumed to satisfy a nonsmooth constraint qualification, which we will discuss later, and we assume only 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 some $x \in \Omega$, because of the internal workings of the simulation used to drive the design. See [2, 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], frame-based methods [16, 44], the generalized pattern search (GPS) [7, 14, 49], the asynchronous parallel pattern search (APPS) approach [25, 29, 36, 34, 35], and the mesh adaptive direct search (MADS) [1, 8]. Related is the implicit filter method [31], though it does use a coarse difference gradient approximation. The reader may consult [31, 33, 37] for a survey of some of these direct search methods.

*Received by the editors November 6, 2007; accepted for publication (in revised form) May 28, 2008; published electronically October 31, 2008. Work of the first author was supported by FCAR grant NC72792 and NSERC grant 239436-05. The second author was supported by LANL 94895-001-04 34. Both were supported by AFOSR FA9550-07-1-0302, the Boeing Company, ExxonMobil Upstream Research Company, and the first and third authors were supported by the Consortium for Research and Innovation in Aerospace in Québec (CRIAQ).

http://www.siam.org/journals/siopt/19-3/70751.html

†GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal (Québec), H3C 3A7, Canada (Charles.Audet@gerad.ca, www.gerad.ca/Charles.Audet, Sebastien.Le.Digabel@gerad.ca).

‡Computational and Applied Mathematics Department, Rice University, Seattle, WA 98136 (dennis@rice.edu, http://www.caam.rice.edu/∼dennis).

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. 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:

> "... parallel variable distribution and parallel direct searches seem an interesting pairing...."

The present paper is based on this remark.

PVD is an evolution of the block-Jacobi technique of [11], which optimizes in parallel a series of reduced subproblems on the subspaces of the original variables of $\mathcal{P}$. Dennis and Torczon [17] described a first parallel version of GPS, which evaluates the black-box in parallel and synchronizes at each iteration to compare solutions and update the current iterates. The APPS [25, 36], 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-box has heterogeneous behavior that depends on the point where it is evaluated. A convergence analysis is presented in [35] 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 algorithm, called PSD-MADS (parallel space decomposition-MADS, can be interpreted as a particular instance of MADS, thus inheriting the main results of the MADS convergence analysis. The paper focuses on the definition of the PSD-MADS frameworks and on its convergence analysis, and not on the choice of the subproblem variables. In our practical implementation of the algorithm, a simple random strategy is used, and it performs well.

The paper is divided as follows: section 2 gives an overview of the PSD and MADS methods. Section 3 presents the new asynchronous parallel algorithm PSD-MADS, and section 4 ensures that the 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 PSD methods. The MADS, its convergence analysis, and a practical implementation are also described in detail.

2.1. PSD methods. PSD 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 $N = \{1, 2, \ldots, n\}$, where n is the number of variables of the optimization problem $\mathcal{P}$, and $Q = \{1, 2, \ldots, q\}$, 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

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

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 simple and random strategy to build the subsets N_p.

The block-Jacobi method in [11] is an iterative two-step algorithm and may be described in a very general way as follows. At each iteration, the first step, the *parallelization*, consists of solving the subproblems in parallel, and the second step, the *synchronization*, gathers the subproblem solutions and constructs 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 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. This strategy is untested as far as we know, and we prefer to avoid estimating penalty constants.

These are parallel synchronous algorithms because the synchronization step waits for all of the processes to end. The conclusion of [19] states 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.

The 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.

In the above references, no practical and generic strategy is given concerning the choice of the subproblem 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 PSD [22] the subspaces can be chosen differently at each iteration.

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

More recently, the multidisciplinary optimization via adaptive response surfaces (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 are not reasonable assumptions for our target class of engineering design problems, 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 con-

vergence analysis. It will also give us greater flexibility concerning the way to handle constraints, the amount of work devoted to the subproblems, the lack of necessity for a synchronization step, and for the choice of the subsets N_p. Concerning this last issue, we remind the reader that we will not propose an elaborated strategy for this, as the focus of the paper is first to define the new method.

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

The constraints defining Ω are handled by the extreme barrier approach, as in [8, 38, 39]. This means that trial points outside Ω are simply rejected by setting their objective function value to ∞. Of course, this requires that the user provides a feasible initial point $x_0 \in \Omega$. We make the standard assumption that all of 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 are rejected.

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

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

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 an $n \times n_D$ 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 nonsingular $n \times n$ matrix and Z is an $n \times n_D$ 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 MADS algorithm with a 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 set of normalized potential poll directions must be chosen from a finite set that is fixed across all iterations. In MADS, the normalized directions may be chosen to be asymptotically dense in the unit sphere, which allows better coverage. We use the

MADS
[0] INITIALIZATION
$x_0 \in \Omega$, $\Delta_0 > 0$, $k \leftarrow 0$
[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 $M(\Delta_k)$ points
POLL STEP (optional if the search step succeeded)
generate n_{dir} MADS directions $d_i \in \mathbb{R}^n$
evaluate the functions on the MADS frame
$P_k = \{x_k + \Delta_k d_i : i = 1, 2, \ldots, n_{dir}\} \subseteq M(\Delta_k)$
[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]** IF no stopping condition is verified

FIG. 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).*

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 is 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 $\omega^+ \geq 0$. Specific values for τ, ω^-, and ω^+ are suggested in section 2.4.

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

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 of the directions in the Clarke tangent cone at $\hat{x}$ are nonnegative. 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 zeroth 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 (lower-triangular MADS) implementation of the MADS algorithm. LTMADS uses positive bases to construct the poll sets. It is stated that the union of these 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. MADS is a general class of algorithms, where 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 be dense in the unit sphere. In [8], after the definition of the MADS framework, a practical implementation is given. This implementation is named LTMADS, since it implies the random construction of a lower-triangular matrix. At this moment, LTMADS is the only published MADS implementation, and all MADS codes in section 5.2 correspond to LTMADS.

LTMADS fixes τ to 4, $\omega^- = -1$, $\omega^+ = 1$, 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-triangular matrix. One of these directions is a special case and is fixed for each value of ℓ. This direction, called $b(\ell)$, has one coordinate (the largest in absolute value) 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 zeroth 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. PSD of MADS (PSD-MADS). This section describes the combination of MADS with a PSD 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 possibly new N_p and x^*.

3.1. General description of PSD-MADS. Although PSD-MADS is an asynchronous parallel algorithm, the notion of iteration is kept, and it 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 $\mathcal{P}_p(x^*)$. This task partition between the pollster and the regular slaves allows the convergence analysis of section 4, where it is shown that the pollster slave executes a valid MADS algorithm, thus inheriting the convergence results of [8]. Note that the pollster slave's task requires the fewest function values of any of the poll steps.

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 of 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. The 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). The number 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 for comparing the two other meshes $M(\Delta_k^1)$ and $M(\Delta_j^p)$. 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 $p \in \{2, 3, \ldots, q-2\}$, the PSD-MADS construction ensures that

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

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 $Q_{reg} = \{2, 3, \ldots, q-2\}$ is the set of the indices of the $q-3$ regular slaves. The qth 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 PSD method to MADS instead of another optimization method is that most of the conditions necessary for convergence in the other PSD methods mentioned in section 2.1 can be relaxed (the smoothness of the functions, the conditions on the constraints, no synchronization step, and no restrictions on the choice of the sets N_p).

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,

POLLSTER ($p = 1$)
Inputs : *pollster mesh size* Δ_k^1
starting point x_0
Output : *pollster solution* x_p
solve problem $\mathcal{P}$: MADS(pollster)
terminate after a single evaluation
send x_p to master

FIG. 2. *Pseudocode for pollster slave.* MADS*(pollster) considers all n variables with a single-poll direction and terminates after one iteration.*

and f is not assumed to be smooth. PSD-MADS also differs from the recent MOVARS algorithm [12], which does require N_p to partition the variables, because it provides a convergence analysis, it 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 the successive 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 of the normalized directions used throughout the algorithm are dense in the unit sphere and that the norms of those directions are in 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 found 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) refers 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 (see (2)).

The pollster pseudocode is shown in Figure 2. The pollster mesh size is updated by the master. The best obtained solution corresponds to 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 $\mathcal{P}$.

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 the 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 an $|N_p|$-variable problem, since all of 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 the trial points on meshes of sizes Δ_j^p, where j is the iteration counter of the subprob-

SLAVE s_p ($p \in Q_{reg}$)
Inputs : *initial mesh size* Δ_0^p
minimum mesh size $\Delta_{\min}^p$
starting point x_0
set of variables N_p
Outputs : *slave solution* x_p
final mesh size Δ_{stop}^p
solve subproblem $\mathcal{P}_p(x^*)$: MADS(s_p)
terminate when $\Delta_j^p < \Delta_{\min}^p$ OR after $bbe_{\max}$ evaluations
send x_p and Δ_{stop}^p to master

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

lem algorithm. The initial mesh size Δ_0^p for MADS(s_p) is set by the master. The value of the parameter $\Delta_{\min}^p$ 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 $\Delta_{\min}^p$ is reached. The final mesh size (Δ_{stop}^p) 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.

3.4. The cache server—$(q-1)$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. This is done to avoid unnecessary expensive function evaluations in case this point has already been evaluated. The cache server provides 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—qth 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 evaluates only 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-2}, the minimal mesh size $\Delta_{\min}^p$ is set to the current value of Δ_k^M. This, as is 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

MASTER
[0] INITIALIZATION
$x^* \leftarrow x_0 \in \Omega$, $\Delta_0^1 \leftarrow \Delta_0^M \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$, $\omega^- \leq -1$, $\omega^+ \geq 0$
start MADS(pollster) with (Δ_0^{user}, x_0) (Figure 2)
FOR ALL ($p \in Q_{reg}$)
construct N_p and set $\Delta_{\min}^p \leftarrow \Delta_0^M$
start MADS(s_p) with (Δ_0^{user}, $\Delta_{\min}^p$, x_0, N_p) (Figure 3)
[1] ITERATIONS
given values from a slave s_p (x_p, Δ_{stop}^p)
IF $\big(f(x_p) < f(x^*)\big)$ (success)
$x^* \leftarrow x_p$
IF ($p = 1$) (pollster, Δ_{stop}^p corresponds to Δ_k^1)
$\Delta_{k+1}^M \leftarrow \tau^{\alpha_k}\Delta_k^1 \leq \min\limits_{p \in Q_{reg}} \Delta_{\min}^p$, with $\alpha_k \in [0; \omega^+]$
$\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$ (Figure 5)
$k \leftarrow k + 1$
start MADS(pollster) with (Δ_k^1, x^*) (Figure 2)
ELSE (regular slave)
construct N_p
$\Delta_{\min}^p \leftarrow \Delta_k^M$
$\Delta_0^p \leftarrow \tau^\gamma \Delta_{stop}^p$, 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, $\Delta_{\min}^p$, x^*, N_p) (Figure 3)
GOTO **[1]** IF no stopping condition is verified

FIG. 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}^p is the last mesh size of a slave s_p. If $p = 1$, $\Delta_{stop}^p = \Delta_k^1 \leq \Delta_k^M$, else $\Delta_{stop}^p \geq \Delta_k^M$. The master evaluates the black-box just once for x_0.*

POLLSTER MESH SIZE UPDATE $\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$
IF (iteration success)
$\omega_k = \alpha_k \in [0; \omega^+]$ $\big(\Delta_{k+1}^1 \leftarrow \Delta_{k+1}^M\big)$
$\big($pollster mesh size increase, $\Delta_{k+1}^1 \geq \Delta_k^1\big)$
ELSE
$\omega_k \in [\omega^-; -1]$
$\big($pollster mesh size decrease, $\Delta_{k+1}^1 < \Delta_k^1\big)$

FIG. 5. *An update of the next pollster mesh size Δ_{k+1}^1. In any case, the pollster mesh size verifies $\Delta_k^1 \leq \Delta_k^M$.*

on meshes finer than $M(\Delta_k^M)$ and that all of 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 $\tau^{\alpha_k}\Delta_k^1 \leq \min_{p \in Q_{reg}} \Delta_{\min}^p$.*

APPARENT POLLSTER
[0] INITIALIZATION
$x_0 \in \Omega$, $\Delta_0^M \leftarrow \Delta_0^1 \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$
[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]** IF no stopping condition is verified

FIG. 6. *A 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.*

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 $\Delta_k^M = \tau^{\alpha_{k-1}}\Delta_{k-1}^1 \leq \min_{p\in Q_{reg}} \Delta_{\min}^p$, and as it corresponds to new values for $\Delta_{\min}^p$, $p \in Q_{reg}$, and 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, nonzero values are 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 pollster mesh and that all of 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.

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 $s_2, s_3, \ldots, s_{q-2}$ on meshes of 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 $\Delta_{stop}^p = \Delta_k^1$), 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 update 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, $\Delta_j^p = \tau^{\omega_{j-1}}\Delta_{j-1}^p = \cdots = \tau^{\sum_{i=0}^{j-1}\omega_i}\Delta_0^{user}$. 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 rth MADS instance performed by s_p. In particular, the last mesh size parameter of this instance can be written $\Delta_{stop}^p = \tau^{-\eta_{stop}}\Delta_0^{user}$, where η_{stop} is a nonnegative integer. From the algorithm described in Figure 4, the first mesh size parameter of the $(r+1)$th MADS instance performed by s_p is $\Delta_0^p = \tau^\gamma\Delta_{stop}^p$, with $\gamma \in \mathbb{Z}$. Then, at iteration j of the $(r+1)$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 $\Delta_{\min}^p$ 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), $M(\Delta_j^p) \subseteq M(\Delta_k^M) \subseteq M(\Delta_k^1)$. Since all MADS($s_p$) trial points are constructed on $M(\Delta_j^p)$, they also lie on $M(\Delta_k^1)$. ∎

This series of propositions ensures that all of 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 the black-box evaluates or is 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 zeroth order result of [8], i.e., $\hat{x}$ cannot be said 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. 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 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 $\tau^\omega z_i \in \mathbb{N}$, $i = 1, 2, \ldots, n_D$, 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 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 $b(\ell)$, $\ell = 1, 2, \ldots$, 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 subproblem variables. This paper does not focus on the choice of the subproblem variables. Rather, we use this very simple strategy: let the sets N_p, $p \in Q_{reg} = \{2, 3, \ldots, q-2\}$, be randomly generated by the master using a

uniform distribution before each subproblem parameter set is 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 $|N_2| = |N_3| = \cdots = |N_{q-2}| = ns$, where ns is a parameter chosen by the user (recall that, for the pollster, $N_1 = N$). Notice also that ns is not required to satisfy $(q-3)ns \geq N$. 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 LTMADS 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 as follows:

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

  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 $\Delta_{stop}^p = \Delta_j^p$ to the master. The next optimization performed by this slave will start with an initial mesh size parameter Δ_0^p equal to $4^\gamma \Delta_{stop}^p$, 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

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

- **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 $\Delta_{\min}^p$ values currently used by regular slaves. These $\Delta_{\min}^p$ 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 $\alpha_k = \eta_k = \log_4\left(\Delta_0^{user}/\Delta_k^1\right)$ (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 slave's mesh

size $\Delta^p_{\min}$, then Δ^M_{k+1} is set to the least $\Delta^p_{\min}$ values. This can be summarized by the following:

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

For example, if $\Delta_0^{user} = \Delta^p_{\min} = 1$ for each $p \in Q_{reg}$ and if the pollster instance fails with a pollster mesh size of $\Delta^1_k = 1/16$, then the master mesh size Δ^M_{k+1} is set to $1/4$ ($\eta_k = 2$, $\alpha_k = 1$).

- **Pollster mesh size $\boldsymbol{\Delta^1_k}$ at iteration $\boldsymbol{k}$ of PSD-MADS:** In the case of an iteration success, Δ^1_{k+1} is set to Δ^M_{k+1} ($\omega_k = \alpha_k \in \mathbb{N}$), or else $\Delta^1_{k+1} = \Delta^1_k/4$ ($\omega_k = -1$):

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

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 $M(\Delta^M_k)$.

Practical termination criteria. The regular slaves $p \in Q_{reg}$ terminate MADS(s_p) as soon as the mesh size parameter Δ^p_j drops below $\Delta^p_{\min} = \Delta^M_k$ (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 $bbe^{global}_{\max}$ black-box evaluations is reached.

5.2. Numerical experiments. The PSD-MADS implementation described in section 5.1 is tested here, on two different problems. The implementation of MADS used to optimize subproblems corresponds to LTMADS and 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 evaluations are made in parallel. Second, pMADS, which is the trivial adaptation of pGPS that uses LTMADS instead of GPS. pGPS and pMADS are both synchronous parallel algorithms. The third method is APPS version 5.0.1 [25, 36], 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:

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

$$\text{subject to}\begin{cases} -\prod_{i=1}^{n} x_i + 0.75 \le 0, \\ \sum_{i=1}^{n} x_i - 7.5n \le 0, \\ 0 \le x_i \le 10, \quad i = 1, 2, \ldots, n. \end{cases}$$

The problem is treated as a black-box, 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 $x_0 = [5\ 5\ \ldots\ 5]^T \in \Omega$. 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 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: $G \ge 250$, or $G \ge 500$ (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 three evaluations for $G \ge 250$, with the resulting feasible point x_0 giving $f(x_0) = 3678.35$ and 74 evaluations for $G \ge 500$, and $f(x_0) = 3014$. These evaluations costs are considered in Figure 8. The feasible starting points, our source code for Problem B, and our best points are available on the website www.gerad.ca/Charles.Audet (see [5]). 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 $G \ge 250$, and $f(x) = 245.866$ for $G \ge 500$.

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* represents the total number of black-box evaluations. One evaluation is counted for the calls to both the objective f and the constraints of Ω.

The most representative cost of a black-box optimization algorithm is the number of black-box evaluations. For this 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). Still, the durations of executions are given. 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 nondecomposing algorithm for problems with a large number of variables ($20 \le n \le 500$).

For all of 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]).

TABLE 1

Numerical results for problems A *and* B: z_{best}, z_{worst}, *and* z_{avg} *give information on the* 30 *runs performed for each* pMADS *and* PSD-MADS *test series,* S_{avg} *gives a measure of the area below the curves in Figures* 7 *and* 8, *and* t_{avg} *represents the average wall clock time, in seconds. Best values appear in bold.*

Algo.	Prob.	z_{best}	z_{worst}	z_{avg}	S_{avg}	t_{avg}	Prob.	z_{best}	z_{worst}	z_{avg}	S_{avg}	t_{avg}
pGPS		-0.450	-0.450	-0.450	1,002	7		-0.277	-0.277	-0.277	3,400	14
APPS	A	-0.519	**-0.519**	-0.519	782	**3**	A	-0.461	-0.461	-0.461	2,355	**6**
pMADS	n=20	**-0.775**	-0.434	-0.592	670	19	n=50	-0.498	-0.430	-0.457	1,939	33
PSD-MADS		-0.761	-0.430	**-0.666**	**595**	8		**-0.727**	**-0.528**	**-0.663**	**1,553**	29
pGPS		-0.089	-0.089	-0.089	18,336	**77**		-0.073	-0.073	-0.073	37,392	**179**
APPS	A	-0.196	-0.196	-0.196	16,934	137	A	-0.129	-0.129	-0.129	35,797	1,300
pMADS	n=250	-0.449	-0.438	-0.444	9,703	95	n=500	-0.447	-0.439	-0.443	19,380	275
PSD-MADS		**-0.698**	**-0.464**	**-0.603**	**8,568**	83		**-0.688**	**-0.461**	**-0.576**	**17,660**	277
pGPS		764.741	764.741	764.741	2,731,920	11		869.559	869.559	869.559	3,552,910	11
APPS	B	813.216	813.216	813.216	3,868,460	**6**	B	1,097.560	1,097.560	1,097.560	4,519,510	**6**
pMADS	$G \geq$	32.700	317.167	112.522	1,071,870	14	$G \geq$	417.049	948.768	662.841	2,892,140	14
PSD-MADS	250	**13.565**	**307.305**	**70.121**	**965,553**	14	500	**245.866**	**731.023**	**463.969**	**2,603,480**	19

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 have 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. The 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 the objective function value z at the $100n$th evaluation are reported. Another measure is S_{avg}, representing the area between a curve z versus *bbe* and the line $z = -0.8$ for Problem A (no run gave $z < -0.8$), and $z = 0$ for Problem B. Wall clock time expressed in seconds are reported in the column t_{avg}. Best runs are obtained with small values for all of 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 7 and 8 summarize the numerical results. For all instances of Problem A, APPS outperforms pGPS, but neither does 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 of 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 of the curves in Figures 7 and 8, one can notice that pMADS is always the fastest to descend, but PSD-MADS overtakes it and produces better solutions. Finally, we remark that APPS terminates in the least wall clock time on smaller problems, albeit with a less optimal function value. However, for problems with 250 and 500 variables, the wall clock time grows significantly worse. This is in accordance with the remark in [25] stating that APPS targets problems with less than 100 variables.

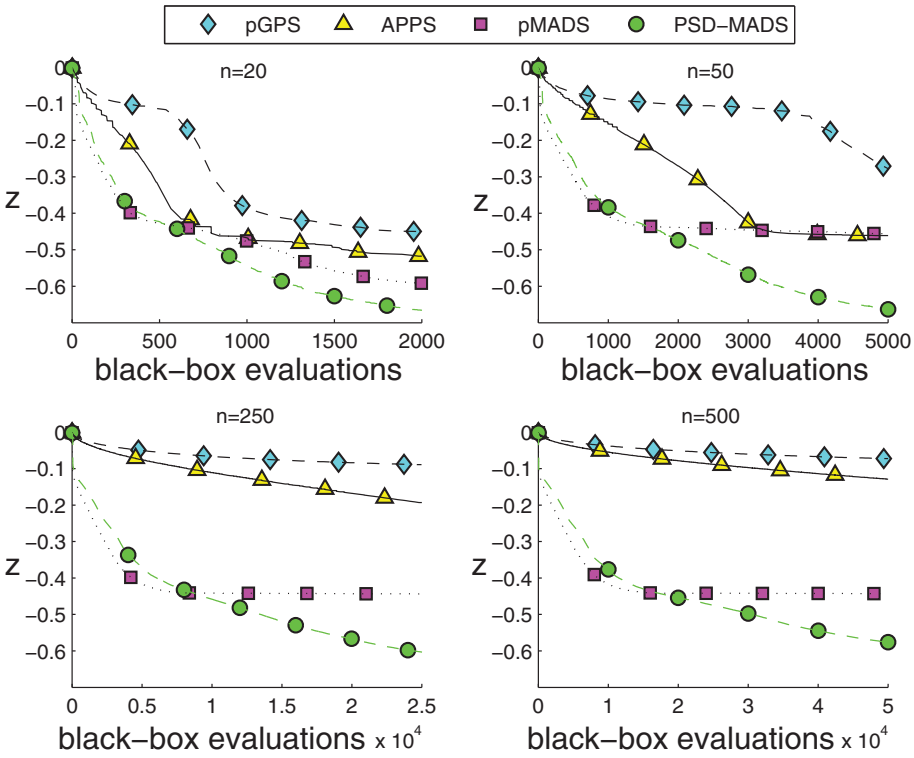

FIG. 7. *Problem* A*: graphs of the objective function value versus the number of evaluations for all test results.* PSD-MADS *and* pMADS *plots correspond to average values of the* 30 *runs performed for each test.*

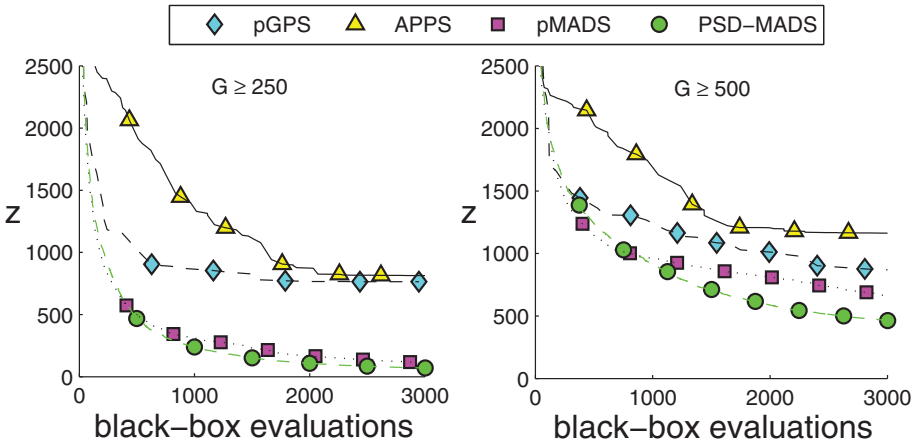

FIG. 8. *Problem* B*: graphs of the objective function value versus the number of evaluations for all test results.* PSD-MADS *and* pMADS *plots correspond to average values of the* 30 *runs performed for each test.*

We conclude this section with some advice for readers interested in testing Psd-Mads. First, we think that the Psd-Mads decomposition is beneficial for problems with more than 20 variables. For these problems, at least 3 processors are necessary. Furthermore, since the master and cache server processes are not demanding in terms of Cpu, 5 processes can be executed on the 3 processors, whose work will be mainly devoted to two regular slaves and the pollster. Two regular slaves is the minimum number to benefit from the decomposition. So, even if only a few processors are available, it is still worthwhile to try this method. Finally, if the user has no particular strategy to choose the subsets of variables in each subproblem, we recommend to equally distribute the variables to the regular slaves. If the user knows that some of the variables are more likely to produce descent than others, then some subproblems can be devoted to these variables, while single-poll Mads can be used on the subproblems of less important variables.

6. Discussion and possible extensions. This paper introduced Psd-Mads, a new Psd technique with less restrictive conditions than usual Psd methods on the functions to be optimized and on the sets of variables in the subproblems. It is shown that the algorithm, from any starting point, produces a subsequence of iterates converging to a solution satisfying local optimality conditions (global convergence to local solutions), based on Clarke calculus and on 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 with a very simple and generic strategy to choose the sets of variables. An obvious extension is a better strategy to decide on the sets of variables in the subproblems. Of course, it is not clear how to do this, in general, or we would have done it here. However, for some applications, 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 order to create a problem similar to the one in [19]. Finally, the 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.

Acknowledgments. We would like to thank anonymous referees for their constructive remarks and comments.

REFERENCES

[1] M. A. Abramson and C. Audet, *Convergence of mesh adaptive direct search to second-order stationary points*, SIAM J. Optim., 17 (2006), pp. 606–619.

[2] M. A. Abramson, *Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm*, Optim. Eng., 5 (2004), pp. 157–177.

[3] P. Alberto, F. Nogueira, H. Rocha, and L. N. Vicente, *Pattern search methods for user-provided points: Application to molecular geometry problems*, SIAM J. Optim., 14 (2004), pp. 1216–1236.

[4] C. Audet, V. Béchard, and S. Le Digabel, *Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search*, J. Global Optim., 41 (2008), pp. 299–318.
[5] C. Audet, G. Couture, and J. E. Dennis, Jr., *The* Nomad *Project*, www.gerad.ca/nomad.
[6] C. Audet, A. L. Custódio, and J. E. Dennis, Jr., *Erratum: Mesh adaptive direct search algorithms for constrained optimization*, SIAM J. Optim., 18 (2008), pp. 1501–1503.
[7] C. Audet and J. E. Dennis, Jr., *Analysis of generalized pattern searches*, SIAM J. Optim., 13 (2002), pp. 889–903.
[8] C. Audet and J. E. Dennis, Jr., *Mesh adaptive direct search algorithms for constrained optimization*, SIAM J. Optim., 17 (2006), pp. 188–217.
[9] C. Audet and J. E. Dennis, Jr., *A* Mads *Algorithm with a Progressive Barrier for Derivative-Free Nonlinear Programming*, SIAM J. Optim., submitted.
[10] C. Audet and D. Orban, *Finding optimal algorithmic parameters using derivative-free optimization*, SIAM J. Optim., 17 (2006), pp. 642–664.
[11] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Inc., Upper Saddle River, NJ, 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, in Proceedings of the 48th AIAA/ASME/ASCE/AHS/ASC Conference on Structures, Structural Dynamics, and Materials, 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, in Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, 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*, Struct. Optim., 17 (1999), pp. 1–13.
[15] F. H. Clarke, *Optimization and Nonsmooth Analysis*, Wiley, New York, 1983. Reissued in 1990 by SIAM, Philadelphia, as Vol. 5 in the series Classics Appl. Math.
[16] I. D. Coope and C. J. Price, *Frame-based methods for unconstrained optimization*, J. Optim. Theory Appl., 107 (2000), pp. 261–274.
[17] J. E. Dennis, Jr. and V. Torczon, *Direct search methods on parallel machines*, SIAM J. Optim., 1 (1991), pp. 448–474.
[18] J. E. Dennis, Jr. and Z. Wu, *Parallel continuous optimization*, Sourcebook of Parallel Computing, Morgan Kaufmann Publishers, San Francisco, CA, 2003, pp. 649–670.
[19] M. C. Ferris and O. L. Mangasarian, *Parallel variable distribution*, SIAM J. Optim., 4 (1994), pp. 815–832.
[20] D. Finkel and C. Kelley, *Convergence Analysis of the* Direct *Algorithm*, Technical report CRSC-TR04-28, Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, 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*, Optim. Eng., 5 (2004), pp. 207–234.
[22] A. Frommer and R. A. Renaut, *A unified approach to parallel space decomposition methods*, J. Comput. Appl. Math., 110 (1999), pp. 205–233.
[23] M. Fukushima, *Parallel variable transformation in unconstrained optimization*, SIAM J. Optim., 8 (1998), pp. 658–672.
[24] J. Gablonsky and C. T. Kelley, *A locally-biased form of the direct algorithm*, J. Global Optim., 21 (2001), pp. 27–37.
[25] G. A. Gray and T. G. Kolda, *Algorithm* 856: *Appspack* 4.0: *A synchronous parallel pattern search for derivative-free optimization*, ACM Trans. Math. Software, 32 (2006), pp. 485–507.
[26] S.-P. Han, *Optimization by updated conjugate subspaces*, in Numerical Analysis, Pitman Res. Notes Math. Ser. 140, D. Griffiths and G. Watson, eds., Longman Sci. Tech., Harlow, 1986, pp. 82–97.
[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*, Canad. J. Chem. Eng., 81 (2003), pp. 1192–1199.
[28] A. Hedar and M. Fukushima, *Derivative-free filter simulated annealing method for constrained continuous global optimization*, J. Global Optim., 35 (2006), pp. 521–549.
[29] P. D. Hough, T. G. Kolda, and V. J. Torczon, *Asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Sci. Comput., 23 (2001), pp. 134–156.

[30] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, J. Optim. Theory Appl., 79 (1993), pp. 157–181.
[31] C. T. Kelley, *Iterative Methods for Optimization*, Frontiers Appl. Math. 18, 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*, Optim. Eng., 2 (2001), pp. 5–29.
[33] T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Rev., 45 (2003), pp. 385–482.
[34] T. G. Kolda and V. Torczon, *Understanding asynchronous parallel pattern search*, in High Performance Algorithms and Software for Nonlinear Optimization, G. DiPillo and A. Murli, eds., Kluwer Academic Publishers, Norwell, MA, 2003, pp. 316–335.
[35] T. G. Kolda and V. J. Torczon, *On the convergence of asynchronous parallel pattern search*, SIAM J. Optim., 14 (2004), pp. 939–964.
[36] T. G. Kolda, *Revisiting asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Optim., 16 (2005), pp. 563–586.
[37] R. M. Lewis, V. Torczon, and M. W. Trosset, *Direct search methods: Then and now*, J. Comput. Appl. Math., 124 (2000), pp. 191–207.
[38] R. M. Lewis and V. Torczon, *Pattern search algorithms for bound constrained minimization*, SIAM J. Optim., 9 (1999), pp. 1082–1099.
[39] R. M. Lewis and V. Torczon, *Pattern search methods for linearly constrained minimization*, SIAM J. Optim., 10 (2000), pp. 917–941.
[40] C.-S. Liu and C.-H. Tseng, *Parallel synchronous and asynchronous space-decomposition algorithms for large-scale minimization problems*, Comput. Optim. Appl., 17 (2000), pp. 85–107.
[41] L. O. Mangasarian, *Parallel gradient istribution in unconstrained optimization*, SIAM J. Control Optim., 33 (1995), pp. 1916–1925.
[42] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin, *Optimal aeroacoustic shape design using the surrogate management framework*, Optim. Eng., 5 (2004), pp. 235–262.
[43] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Comput. J., 7 (1965), pp. 308–313.
[44] C. J. Price and I. D. Coope, *Frames and grids in unconstrained and linearly constrained optimization: A nonsmooth approach*, SIAM J. Optim., 14 (2003), pp. 415–438.
[45] C. A. Sagastizábal and M. V. Solodov, *Parallel variable distribution for constrained optimization*, Comput. Optim. Appl., 22 (2002), pp. 111–131.
[46] M. V. Solodov, *New inexact parallel variable distribution algorithms*, Comput. Optim. Appl., 7 (1997), pp. 165–182.
[47] M. V. Solodov, *On the convergence of constrained parallel variable distribution algorithms*, SIAM J. Optim., 8 (1998), pp. 187–196.
[48] B. Tang, *Orthogonal array-based latin hypercubes*, J. Amer. Statist. Assoc., 88 (1993), pp. 1392–1397.
[49] V. Torczon, *On the convergence of pattern search algorithms*, SIAM J. Optim., 7 (1997), pp. 1–25.
[50] P. Tseng, *Dual coordinate ascent methods for non-strictly convex minimization*, Math. Program., 59 (1993), pp. 231–247.
[51] E. Yamakawa and M. Fukushima, *Testing parallel variable transformation*, Comput. Optim. Appl., 13 (1999), pp. 253–274.