Optimization Methods and Software
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/goms20

On sequential and parallel non-monotone derivative-free algorithms for box constrained optimization

Ubaldo M. García-Palomares [a b], Ildemaro J. García-Urrea [c] & Pedro S. Rodríguez-Hernández [b]

[a] Universidad Simón Bolívar, Procesos y Sistemas, Caracas, 1080, Venezuela

[b] Universidade de Vigo, Ingeniería Telemática, 36310, Vigo, Spain

[c] Universidad Simón Bolívar, Cómputo Científico y Estadística, Caracas, 1080, Venezuela

Published online: 21 Jun 2012.

To cite this article: Ubaldo M. García-Palomares, Ildemaro J. García-Urrea & Pedro S. Rodríguez-Hernández (2013) On sequential and parallel non-monotone derivative-free algorithms for box constrained optimization, Optimization Methods and Software, 28:6, 1233-1261, DOI: 10.1080/10556788.2012.693926

To link to this article: http://dx.doi.org/10.1080/10556788.2012.693926

Optimization Methods & Software, 2013
Vol. 28, No. 6, 1233–1261, http://dx.doi.org/10.1080/10556788.2012.693926

On sequential and parallel non-monotone derivative-free algorithms for box constrained optimization

Ubaldo M. García-Palomares[a,b]*, Ildemaro J. García-Urrea[c] and Pedro S. Rodríguez-Hernández[b]

[a]*Universidad Simón Bolívar, Procesos y Sistemas, Caracas 1080, Venezuela;* [b]*Universidade de Vigo, Ingeniería Telemática, 36310 Vigo, Spain;* [c]*Universidad Simón Bolívar, Cómputo Científico y Estadística, Caracas 1080, Venezuela*

(*Received 29 November 2010; final version received 29 April 2012*)

This paper proposes a space decomposition scheme for non-monotone (NM) derivative-free parallel and sequential algorithms for solving the box-constrained optimization problem (BCOP). Convergence to Karush Kuhn Tucker points is proved under the same conditions for NM and monotone algorithms for solving unconstrained and BCOPs. The parallel algorithm has two unique features: all processors exchange information on discrete quasi-minimal points, and are able to sample function values on the whole set of directions that conform to non-negative spanning sets for each decomposed subspace. The parallel algorithm has a high degree of fault tolerance: convergence prevails even if only one processor remains running. Preliminary results are encouraging for solving small- and medium-sized problems.

Keywords: space decomposition; parallel; derivative free; non-monotone; box constraints; discrete quasi-minimal points

AMS Subject Classification: 65K05; 90C56

1. Introduction

This paper is concerned with the elaboration of a pattern search (PS) algorithm for solving the box-constrained optimization problem (BCOP) defined as

$$\min_{x\in\mathcal{F}} f(x), \quad \mathcal{F} = \{x \in \mathbb{R}^n : s \le x \le t\},$$

where $f(\cdot) : \mathcal{F} \to \mathbb{R}$ is a real-valued function of n variables, $s < t$, and vector inequalities hold component wise. In the sequel, we assume that derivative information is either unreliable or inexistent and all information about $f(\cdot)$ must be derived from its function values.

One objective of this paper is to prove convergence of a non-monotone (NM) PS algorithm under the same standard assumptions required by its monotone counterpart. Another objective is to ensure that the parallel version of the algorithm can be easily implemented in modern multi-core desktop computers. This objective was achieved using a variable separation (VS) scheme that is as well useful in other multi-processing architectures. The parallel algorithm has a high degree of

*Corresponding author. Email: garciap@usb.ve, ubaldo@det.uvigo.es

fault tolerance and it is able to solve the BCOP under the extreme condition of a major breakdown. The algorithm works even if only one processor is running.

The paper is structured as follows: the remainder of this section is divided into subsections, each conveying basic material on a topic related to the work described in subsequent sections. Section 2 introduces the particular space decomposition strategy considered for the algorithms proposed in this paper and analyses the convergence of the sequential version. It also pinpoints the basic iteration that will play a fundamental role in all algorithms. Section 3 analyses a new parallel derivative-free (DF) algorithm and highlights its fault tolerance feature. Section 4 discusses implementation issues and gives additional details to improve the algorithmic performance. Numerical tests on small and medium-sized problems are reported in Section 5. Finally, Section 6 ends this paper with closing remarks.

1.1 *Notation*

We use a rather standard notation with some peculiarities. Throughout the paper $f(\cdot) : \mathcal{F} \to \mathbb{R}$ stands for the objective function. Other small Latin letters denote vectors in some Euclidean space, except for letters i, j, k, m, n, p, q, r which are used as integers that take on different roles along the paper; vector components are indicated by super indices, x_i^k stands for the kth component of the vector x_i; $e_k, k = 1, \ldots, n$ are the canonical vectors in $\mathbb{R}^n$, i.e. $e_k^k = 1, e_k^j = 0, j \neq k$; $\mathbb{N} = \{1, 2, \ldots\}$ is the set of natural numbers. Capital Latin letters denote sets of integers, specifically $N \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$, the difference set $M - L = \{j \in M : j \notin L\}$. Cursive Latin letters will denote sets of vectors. If $M \subseteq N, \mathcal{R}^M = \{x \in \mathbb{R}^n : x^j = 0, j \notin M\}$. Notationally $\mathcal{R}^N = \mathbb{R}^n$ the n dimensional Euclidean space. If $\mathcal{Q}$ is a finite set of vectors in $\mathbb{R}^n$, say $\mathcal{Q} = \{d_j \in \mathbb{R}^n, j = 1, \ldots, m\}$, we denote by $\mathcal{Q}^+$ the non-negative linear combinations of vectors in $\mathcal{Q}$, i.e. $\mathcal{Q}^+ = \{x \in \mathbb{R}^n : x = \sum_{j=1}^m \alpha_j d_j, d_j \in \mathcal{Q}, \alpha_j \geq 0, j = 1, \ldots, m\}$. Lowercase Greek letters are scalars or scalar functions. Sequences of entities generated by the algorithm will be distinguished with the subindex i. If $\mathcal{Q}_i = \{d_{ij} \in \mathbb{R}^n, j = 1, \ldots, m\}, i \in \mathbb{N}$, then $\{\mathcal{Q}_i\}_{i \in J} \to \mathcal{Q}_*$ means $\{d_{ij}\}_{i \in J} \to d_{*j}, j = 1, \ldots, m$. Also, $\{(x_i, \mathcal{Q}_i, \tau_i)\}_{i \in J} \to (x_*, \mathcal{Q}_*, \tau_*)$ means $\{x_i\}_{i \in J} \to x_*, \{\mathcal{Q}_i\}_{i \in J} \to \mathcal{Q}_*$, and $\{\tau_i\}_{i \in J} \to \tau_*$. Indices i will be dropped in pseudo codes and when it is clear from the context.

1.2 *Derivative free*

A brief account on earlier methods that neither computed nor approximated derivatives can be seen in [8, Chapter 7, Section 1], albeit no formal proof of convergence is given. First convergence results are found in [10,18], but it was Torczon's dissertation [49] that sprung a vivid interest in DF methods. Roughly we can distinguish two approaches for solving an optimization problem without explicit use of derivatives: model-based methods, which iteratively solve a quadratic – or linear – approximation to the objective function and PS methods, which seek information about the behaviour of the objective function on a sample of points around the given estimate [11,44]. Some researchers argue that model-based methods are more effective than PS methods when there is enough smoothness in the objective function, but there is not yet a strong evidence to support this claim, specially for large-scale models. This paper presents a PS algorithm that mixes, in a simple but effective way, features that have appeared in previous works. Besides, it includes a new parallel scheme that allows less communication load among processors and exhibits a high degree of fault tolerance. Convergence is ensured for functions strictly differentiable at their limit points.

Convergence theory for monotone PS methods for unconstrained optimization seems now well understood [3,12,13,20,31,37]. The key elements stem from the following definition and proposition adapted from [13–15].

DEFINITION 1.1 *Let $M \subseteq N$. A finite set of unitary directions $\mathcal{Q} = \{d_j \in \mathcal{R}^M : j = 1, \ldots, m\}$ is a non-negative spanning set for $\mathcal{R}^M$ if $\mathcal{Q}^+ = \mathcal{R}^M$ in other words, any vector $v \in \mathcal{R}^M$ can be expressed as $v = \sum_{j=1}^{m} \alpha_j d_j$ with $\alpha_j \geq 0, j = 1, \ldots, m$.*

PROPOSITION 1.2 *Let $\mathcal{Q} = \{d_j \in \mathcal{R}^M,\ j = 1, \ldots, m\}$ be a finite set of unitary directions and let $0 \neq v \in \mathcal{R}^M$. If $\mathcal{Q}^+ = \mathcal{R}^M$ there always exists some $d \in \mathcal{Q}$ such that $v^{\mathrm{T}} d < 0$.*

If $f(\cdot)$ is continuously differentiable, this proposition implies that any finite set $\mathcal{Q}$ that spans non-negatively $\mathbb{R}^n$ contains a descent direction d at any $x \in \mathbb{R}^n$; in other words, $(\forall x \in \mathbb{R}^n)(\exists d \in \mathcal{Q}, \tau > 0) : f(x + \tau d) < f(x)$; therefore a lower function value must be found if we search along all $d \in \mathcal{Q}$, unless $\nabla f(x) = 0$.

Remark 1.3 In [13, Definition 2.2] Coope and Price define $(x, \mathcal{Q}, \tau)$ as a minimal frame if and only if $\mathcal{Q}^+ = \mathbb{R}^n$ and $f(x + \tau d) \geq f(x), \forall d \in \mathcal{Q}$. For convergence purposes, they introduce [13, Definition 2.3] the quasi-minimal frame $(x, \mathcal{Q}, \tau)$, which satisfies the weaker condition

$$\begin{gathered} \mathcal{Q}^+ = \mathbb{R}^n, \quad \text{and} \\ f(x + \tau d) \geq f(x) - \epsilon, \forall d \in \mathcal{Q}. \end{gathered} \tag{1}$$

A point x satisfying (1) is called a quasi-minimal point. We will extend these concepts to the BCOP.

Nowadays, extensions of PS methods from unconstrained optimization to the BCOP and the more general constrained problems exist [4,5,12,31,36,38,39]. To ensure convergence, the set $\mathcal{Q}$ of search directions must conform to the geometry of the problem [24,32,40]. It has also been stated that a sufficient decrease condition (SDC) on the function values allows more freedom to choose the set of directions $\mathcal{Q}$ [13, Section 3, last paragraph]. SDC has been suggested for unconstrained optimization problems [13,18,20,37], and also for solving linearly constrained problems [32, Section 5]. This paper considers an SDC on a reference value greater than the function value at the current iterate, as it was proposed in [21]. The next estimate is any $x_{i+1} \in \mathcal{F}$ that fulfils

$$f(x_{i+1}) \leq \varphi_i - \sigma_i, \tag{2}$$

where $\varphi_i \geq f(x_i)$ is a controlled reference value and σ_i is a positive value satisfying suitable conditions.

1.3 *Non-monotone*

Early iterative optimization algorithms that computed derivatives were monotone. Given an estimate x_i, a direction of descent d_i is computed, and a stepsize $\tau_i > 0$ is found such that $f(x_{i+1}) = f(x_i + \tau_i d_i) < f(x_i)$. To our knowledge NM algorithms were introduced by Grippo *et al.* [25] for solving the unconstrained optimization problem. Given an estimate x_i, a *quasi-Newton* direction d_i and a fixed q, a stepsize τ_i is determined such that

$$f(x_{i+1}) = f(x_i + \tau_i d_i) \leq \max_{0 \leq j \leq q} f(x_{i-j}) + 0.001 \tau_i \nabla f(x_i)^{\mathrm{T}} d_i. \tag{3}$$

This idea has been incorporated since then into the trust region and other techniques for solving both constrained and unconstrained problems [16,26,48]. It has been stated that NM algorithms

could be useful for dealing with noisy problems [31] and they are as well beneficial to prevent premature convergence on narrow valleys or local minima. Currently, almost any optimization algorithm that computes derivatives has an NM counterpart that converges under the same assumptions ([19,48] and many others). By contrast, few NM PS deterministic algorithms have been proposed in the DF optimization [17,21]. This paper continues the research on NM algorithms proposed in [21] and extends their results for solving the BCOP. A new estimate $x_{i+1} \in \mathcal{F}$ fulfils (2) on a reference value $\varphi_i \geq f(x_i)$. Of course, σ_i is a positive expression that does not imply the use of derivatives. It is worth mentioning that $x_{i+1} \in \mathcal{F}$ satisfying (2) can be obtained by any means. If our primary intention is to locate a global minimum we could incorporate heuristics that do not force a decrease on $f(\cdot)$, like simulated annealing [27], and others. In a multi-processing environment, the next iterate x_{i+1} can be provided by any processor. This is the approach followed in this paper.

1.4 *Space decomposition*

Space Decomposition, also denoted as variable separation (VS), is an old strategy in optimization problems. It has been used to separate variables of different nature. It is for instance the approach commonly used for solving mixed integer problems ([1,35] and references therein). VS is also used for solving, mainly in parallel, large-scale models. A good account on the convergence and development of space decomposition techniques is found in [6]. Earlier approaches split $N = \bigcup_1^p M_j$ and solved subproblems of smaller size iteratively. Given $M \subset N$, VS methods are concerned with the problem of minimizing a function on a restricted subspace, namely

$$\min f(x), x \in \{x \in \mathcal{F} := y + v, v \in \mathcal{R}^M\}, \tag{4}$$

where $y \in \mathbb{R}^n$ is generally obtained from a similar subproblem on a different subspace. In [47], the authors showed that for a convex $\mathcal{F}$ there is no need to solve problem (4) accurately. A Newton step, which implicitly calls for an SDC, suffices to maintain convergence. None of these previous works include NM algorithms, which is one of the goals of this paper. To the authors' knowledge, the first convergence results for a monotone DF algorithm with space decomposition were given in [6].

In real problems, a certain decomposition may naturally arise that brings about a significant reduction in the processing time needed to compute function values [22]. It will be pointed out in the next section that space decomposition is justified for *partially* separable functions, because it enables multi-processing in a natural way; but in general, there is no visible strategy to an efficient decomposition [6, Section 6]. The following remark suggests a scheme that becomes relevant in VS.

Remark 1.4 Let $L \subseteq N$ be a given set, let $M_j \subseteq L$, and let $\mathcal{D}_j^+ = \mathcal{R}^{M_j}, j = 1, \ldots, p$.
If $\bigcup_{j=1}^p M_j = L$ and $\mathcal{D} = \bigcup_{j=1}^p \mathcal{D}_j$ then $\mathcal{D}^+ = \mathcal{R}^L$. In particular, if $L = N, \mathcal{D}^+ = \mathbb{R}^n$.

1.5 *Necessary conditions*

This section gathers necessary conditions on subspaces. Let $M \subseteq N, \delta \geq 0$ and let

$$\begin{aligned} B(x,\delta) &= \{k \in M : s^k + \delta < x^k < t^k - \delta\}, \\ S(x,\delta) &= \{k \in M : s^k \leq x^k \leq s^k + \delta\}, \\ T(x,\delta) &= \{k \in M : t^k \geq x^k \geq t^k - \delta\}. \end{aligned} \tag{5}$$

When $f \in C^1$ the point $x_* \in \mathcal{F}$ is a first-order Karush Kuhn Tucker (KKT) point for problem (4) if

$$\nabla f(x_*)^k \begin{cases} = 0 & \text{if } k \in B(x_*, 0), \\ \geq 0 & \text{if } k \in S(x_*, 0), \\ \leq 0 & \text{if } k \in T(x_*, 0), \end{cases} \tag{6}$$

We point out that there are no conditions on $\nabla f(x_*)^k$, for $k \notin M$.

DEFINITION 1.5 *We say that $x_* \in \mathcal{F}$ satisfies the zero-order necessary condition on a set of unitary directions $\mathcal{D} = \{d_j \in \mathcal{R}^{B(x_*,0)} : j = 1, \ldots, m\}$ if $\mathcal{D}^+ = \mathcal{R}^{B(x_*,0)}$ and for all small enough $\tau > 0$*

$$\begin{aligned} d \in \mathcal{D} &\Rightarrow f(x_* + \tau d) - f(x_*) \geq 0, \\ k \in S(x_*, 0) &\Rightarrow f(x_* + \tau e_k) - f(x_*) \geq 0, \\ k \in T(x_*, 0) &\Rightarrow f(x_* - \tau e_k) - f(x_*) \geq 0. \end{aligned} \tag{7}$$

LEMMA 1.6 *If $x_* \in \mathcal{F}$ satisfies* (7) *and $f(\cdot)$ is Gateaux differentiable at x_*, then x_* satisfies* (6).

Proof Trivially $\nabla f(x_*)^k \geq 0$, for $k \in S(x_*, 0)$, and $\nabla f(x_*)^k \leq 0$, for $k \in T(x_*, 0)$. It remains to prove that $\nabla f(x_*)^k = 0$ for $k \in B(x_*, 0)$. Let $v^k = \nabla f(x_*)^k$, for $k \in B(x_*, 0)$, and $v^k = 0$, otherwise. By (7) $v^T d \geq 0$ for all $d \in \mathcal{D}$. By proposition 1.2 $v = 0$, i.e. $\nabla f(x_*)^k = 0$ for $k \in B(x_*, 0)$. ■

Next lemma considers the case when a sequence $\{x_i\}_{i \in I}$ is converging to x_*. We need a more stringent condition on $f(\cdot)$.

LEMMA 1.7 *Let $\lim_{\tau \to 0} \sigma(\tau)/\tau = 0$, let $\mathcal{D}_i = \{d_{i1}, \ldots, d_{im}\}, i \in \mathbb{N}$ be a sequence of finite sets of unitary directions and let $\{x_i, \mathcal{D}_i, \tau_i\}_{i \in J} \to (x_*, \mathcal{D}_*, 0)$ for some $J \subseteq \mathbb{N}$.*

If $\begin{cases} f(x_i + \tau_i d_{ij}) - f(x_i) \geq -\sigma(\tau_i), j = 1, \ldots, m; i \in J, \\ \mathcal{D}_*^+ = \mathcal{R}^{B(x_*,0)}, \text{and} \\ f(\cdot) \text{ is strictly differentiable at } x_* \end{cases}$

then $\nabla f(x_*)^k = 0, k \in B(x_*, 0)$.

Proof By strict differentiability, we obtain for any $j \in \{1, \ldots, m\}$ that

$$\nabla f(x_*)^{\mathrm{T}} d_{*j} = \lim_{\substack{x_i \to x_* \\ d_{ij} \to d_{*j} \\ \tau_i \to 0}} \frac{f(x_i + \tau_i d_{ij}) - f(x_i)}{\tau_i} \geq \lim_{\tau_i \to 0} -\frac{\sigma(\tau_i)}{\tau_i} = 0,$$

so $d_{*j} \in \mathcal{D}_* \Rightarrow \nabla f(x_*)^T d_{*j} \geq 0$. By the previous lemma, we conclude that $\nabla f(x_*)^k = 0$, for $k \in B(x_*, 0)$. ■

COROLLARY 1.8 *If $\{x_i, \mathcal{D}_i, \tau_i\}_{i \in J} \to (x_*, \mathcal{D}_*, 0)$, if $\mathcal{D}_*^+ = \mathcal{R}^{B(x_*,0)}$, if $f(\cdot)$ is strictly differentiable at x_*, and for large enough $i \in J$*

$$\begin{aligned} d_{ij} \in \mathcal{D}_i &\Rightarrow [f(x_i + \tau_i d_{ij}) - f(x_i) \geq -\sigma(\tau_i)], \\ k \in S(x_*, 0) &\Rightarrow [f(x_i + \tau_i e_k) - f(x_i) \geq -\sigma(\tau_i)], \\ k \in T(x_*, 0) &\Rightarrow [f(x_i - \tau_i e_k) - f(x_i) \geq -\sigma(\tau_i)], \end{aligned} \tag{8}$$

then x_* satisfies (6).

Proof It is trivial. ■

Remark 1.9 When $M = N$ the conditions stated above become necessary conditions for the BCOP.

We should be aware that descent feasible directions might exist at x_* when $f(\cdot)$ is non-smooth, even when x_* satisfies the necessary conditions given above. It has been argued that this situation may persist as long as the number of search directions is finite. Recent algorithms have been proposed that asymptotically generate a dense set of search directions $\mathcal{D}$ in the unit hypersphere [4–6]. They assume Lipschitz continuity of $f(\cdot)$ instead of strict differentiability and prove the stronger necessary condition

$$\limsup_{\tau_i \downarrow 0, x_i \to x_*} \frac{f(x_i + \tau_i d) - f(x_i)}{\tau_i} \geq 0, \quad \text{for all feasible direction } d \in \mathcal{D}. \tag{9}$$

Despite this interesting result, we will focus our efforts on functions that are strictly differentiable at their limit points. This will allow us to deal with a small number of search directions at all iterations and to develop a similar convergence theory for both sequential and parallel algorithms.

1.6 *Assumptions for BCOP*

When $B(x_*, 0) = N \stackrel{\text{def}}{=} \{1, \ldots, n\}$, the BCOP becomes, locally, an unconstrained minimization problem. Since this is not a fortuitous case, it is desirable to maintain the same convergence assumptions for both problems. Furthermore, all strategies suggested in previous works to improve the performance of NM PS algorithms for unconstrained optimization should be easily incorporated into algorithms for solving BCOP. We now list the convergence assumptions of NM PS algorithms for BCOP:

A1 $\{x_i\}_1^\infty$ remains in a compact set.
A2 f is bounded below on $\mathcal{F}$ and it is strictly differentiable at limit points of $\{x_i\}_1^\infty$.
A3 $\sigma : \mathbb{R}_+ \to \mathbb{R}_+$ is a positive strictly increasing function and $\lim_{\tau \downarrow 0} \sigma(\tau)/\tau = 0$.
A4
- $\varphi_i \geq f(x_i)$ for $i \in \mathbb{N}$.
- If the index set $I = \{i \in \mathbb{N} : x_{i+1} \in \mathcal{F}; f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i)\}$ is infinite, there exists an infinite set $J = \{i \in I : \varphi_{i+1} \leq \varphi_i - \sigma'(\tau_i)\}$, with $\sigma(\cdot)$ and $\sigma'(\cdot)$ satisfying A3.

A5 Given $\delta > 0, \emptyset \neq M_i \subseteq N$. Construct $\mathcal{Q}_i, \mathcal{D}_i$ as

$$\begin{aligned}
&\bullet\ A_i \subseteq B(x_i, \delta), \\
&\bullet\ \mathcal{Q}_i^+ = \mathcal{R}^{A_i}, \\
&\bullet\ \mathcal{D}_i = \mathcal{Q}_i \cup \mathcal{E}_i, \mathcal{E}_i = \{\pm e_j : j \in (M_i - A_i)\}, \\
&\bullet\ \text{Let } K = \{i \in \mathbb{N} : M_i = M, \text{ for some } M\} \\
&\qquad \mathcal{D}_*^+ = \mathcal{R}^M \text{ for any accumulation point of } \{\mathcal{D}_i\}_{i \in K}
\end{aligned} \tag{10}$$

A5 holds if $A_i = \emptyset$ for all i. It also holds if we choose $\mathcal{D}_i = \{\pm e_k : k \in M_i\}$. In the implementation section (Section 4.9), other choices are discussed.

Remark 1.10 It is obvious that $\mathcal{D}_i^+ = \mathcal{R}^{M_i}$. Besides, the set of directions $\mathcal{Q}_i \cup \{\pm e_j : j \in (L - A_i)\}$ positively spans $\mathcal{R}^L$ when $M_i \supseteq L \supseteq A_i$. This holds in particular when $L = B(x_*, 0)$, as defined in Section 1.5.

Sections 2 and 3 sketch the convergence proofs for sequential and parallel algorithms that solve BCOP under Assumptions A1–A5.

2. Sequential algorithms

Besides efficiently solving the unconstrained problem, a good algorithm must consider with care the *near* active constraints; otherwise convergence failures may occur [39]. The way we have defined $\mathcal{D}_i$ in (10) naturally takes into account the closeness of x_i to a bound constraint. If the variable x_i^k is δ-active, i.e. $k \notin B(x_i, \delta)$, the algorithm samples $f(\cdot)$ along directions with a null k-component and at $x_i \pm \tau_i e_k$. This strategy ensures convergence while allowing maximum flexibility in choosing directions.

A fixed large value of δ may impose $B(x_i, \delta) = \emptyset$, and $\mathcal{D}_i = \{\pm e_k : k \in M\}$. This is a natural extension of PS algorithms from unconstrained minimization to the BCOP that ensures convergence when $M = N$ [20,41]; however, numerical results show that this choice of $\mathcal{D}_i$ is not recommended [20,21], although some improvement can be achieved with linear interpolation or with a model function approximating $f(\cdot)$ [36,38]. The nucleus of the NM PS algorithm for solving the BCOP is the iterative application of

$$
\left.
\begin{array}{l}
\text{Given } x_i \in \mathcal{F}, \tau_i > 0, \varphi_i \geq f(x_i), \gamma > 1, \mu \in (0\ 1), \delta > 0 \\
\text{Construct } \mathcal{D}_i \text{ satisfying A5} \\
\text{IF} \exists d \in \mathcal{D}_i : y \stackrel{\text{def}}{=} (x_i + \tau_i d) \in \mathcal{F} \text{ AND} f(y) \leq \varphi_i - \sigma(\tau_i) \\
\qquad \text{Choose any } x_{i+1} \in \mathcal{F} \text{ such that } \begin{cases} f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i), \\ \|x_{i+1} - y\| \leq \gamma \tau_i \end{cases} \\
\qquad \text{Update } \varphi_i \\
\qquad \tau_{i+1} = \tau_i \\
\text{ELSE} \\
\qquad x_{i+1} = x_i, \varphi_{i+1} = \varphi_i \\
\qquad \tau_{i+1} = \mu \tau_i \\
\text{ENDIF} \\
i = i + 1
\end{array}
\right\} \tag{11}
$$

We can pick $x_{i+1} = y$ found in (11). For unconstrained problems it is a common practice to locate x_{i+1} along d; i.e. $x_{i+1} = x_i + \eta d$ for some $\eta > 0$. Linear interpolation is suggested in [17,18], and quadratic interpolation in [21].

Remark 2.1 We could allocate some of our resources to find x_{i+1} by heuristic means within a reasonable time span or number of function evaluations. We could also adopt the strategy suggested in [4,39] and [13, Algorithm 3.2]. Essentially they compute $f(y), y \in (\mathcal{F} \cap \mathcal{M})$, where $\mathcal{M}$ is a set of points located on a mesh around x_i. In a multi-processing, $\mathcal{M}$ may be provided by any of the threads or processors.

Remark 2.2 When the IF clause in (11) is false and $\mathcal{D}_i^+ = \mathcal{R}^{M_i}$, the triplet $(x_i, \mathcal{D}_i, \tau_i)$ will be denoted as a discrete quasi-minimal frame, and x_i will be a (discrete quasi-minimal point (DQP)) on $\mathcal{R}^{M_i}$. This terminology is adapted from monotone algorithms for unconstrained minimization (Remark 1.3) to NM algorithms for box constrained problems. In short, given any $M \subseteq N$, $(x, \mathcal{D}, \tau)$ and x are, respectively, a discrete quasi-minimal frame and a DQP on $\mathcal{R}^M$ when

$$
\mathcal{D}^+ = \mathcal{R}^M \quad \text{and} \quad d \in \mathcal{D} \Rightarrow \left\langle \begin{array}{l} (x + \tau d) \notin \mathcal{F} \quad \text{OR} \\ f(x + \tau d) > \varphi - \sigma(\tau) \end{array} \right. \tag{12}
$$

THEOREM 2.3 *Under assumptions* A2–A4, *the iterative application of* (*11*) *ensures that* $\{\tau_i\}_1^\infty \downarrow 0$.

Proof Let I be the index set identified by assumption A4. We assume that $\{\tau_i\}_1^\infty \downarrow \epsilon > 0$ and show a contradiction. This implies that the ELSE part of (11) only occurs a finite number of times; hence $x_{i+1} \in \mathcal{F}$ and $f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i)$ holds for all large enough i, which means that exists $\bar{\imath}$: $(i > \bar{\imath} \Rightarrow i \in I)$. By A5 we identify an infinite subset $J = \{i_1, \ldots, i_k, \ldots\}$ and

$$\varphi_{i_k+1} \leq \varphi_{i_1} - \sum_{j=1}^{k} \sigma'(\tau_{i_j}).$$

By A3 we assert that $\sigma'(\tau_i) > \sigma'(\epsilon) > 0$ for all $i \in \mathbb{N}$; therefore

$$\varphi_{i_k+1} \leq \varphi_{i_1} - \sum_{j=1}^{k} \sigma'(\epsilon),$$

which generates a sequence $\{\varphi_i\}_{i\in J}$ unbounded from below, but since $f(x_i) \leq \varphi_i$ we deduce that $f(\cdot)$ is unbounded from below, contradicting A2. This contradiction ensures the validity of the theorem. ■

A5 was not needed to prove that $\{\tau_i\}_1^\infty \downarrow 0$. However, for convergence to a KKT point, special attention must be paid to the choice of the directions of search. Next theorem assumes $M_i = M$ for all i and proves convergence under assumptions A1–A5.

THEOREM 2.4 *Let $\{x_i, \mathcal{D}_i, \tau_i, \varphi_i\}_1^\infty$ be the sequence generated by iterative application of* (11) *starting at $x_0 \in \mathcal{F}$ with $M_i = M$ for all i. Under Assumptions* A1–A5, *there exists an infinite subsequence of DQPs on $\mathcal{R}^M$ converging to a first-order KKT point of the problem*

$$\begin{aligned} &\min f(x) \\ \textit{subject to:}\quad &x^k = x_0^k,\ k \in \{N - M\}, \\ &s^k \leq x^k \leq t^k,\ k \in M. \end{aligned}$$

Proof Since $\{\tau_i\}_1^\infty \downarrow 0$ the ELSE part of the IF clause in (11) will be executed an infinite number of times. Since $\mathcal{D}_i^+ = \mathcal{R}^M$ by A5, there exists an infinite index sequence $K \subseteq \mathbb{N}$ and a sequence $\{x_i\}_{i\in K}$ of DQPs on $\mathcal{R}^M$ satisfying (12). We now extract an infinite subsequence $J \subseteq K$ with the following characteristics: (i) $\{(x_i, \mathcal{D}_i, \tau_i)\}_{i\in J} \to (x_*, \mathcal{D}_*, 0)$. and (ii) the index set A_i defined in A5 is fixed, say, $A_i = A \forall i \in J$,

For large enough $i \in J$ and $k \in M$ we have three cases to consider:

(i): $x_*^k = s^k$

This implies that $k \notin B(x_i, \delta)$. By construction (10), we obtain that $k \notin A$. Consequently $e_k \in \mathcal{E}_i$. Let $\delta = t^k - s^k$. For i large enough we have $x_i^k + \tau_i \leq (s^k + \delta/3) + \delta/3 = s^k + 2/3 \cdot \delta < t^k$; hence $(x_i + \tau_i e_k) \in \mathcal{F}$. Since (12) holds we have that

$f(x_i + \tau_i e_k) > \varphi_i - \sigma(\tau_i)$.

(ii): $x_*^k = t^k$

With an akin argument, we obtain that

$$f(x_i - \tau_i e_k) > \varphi_i - \sigma(\tau_i).$$

(iii): $s^k < x_*^k < t^k$

Remark 1.10 asserts that $\mathcal{C}_i = \mathcal{Q}_i \cup \{\pm e_j : j \in (B(x_*, 0) - A)\}$ positively spans $\mathcal{R}^{B(x_*,0)}$. Since $(x_i + \tau_i d) \in \mathcal{F}$ for all $d \in \mathcal{C}_i$ and 12 holds, we obtain

$$d \in \mathcal{C}_i \Rightarrow f(x_i + \tau_i d) > \varphi_i - \sigma(\tau_i).$$

We now resort to Lemma 1.7 and Corollary 1.8 and assert the validity of the theorem. ■

As an immediate consequence of this theorem, we have

COROLLARY 2.5 *Assume that* A1–A5 *hold and* $M = N$ *in construction* (10). *The iterative application of* (11) *generates a subsequence that converges to a KKT point of the BCOP.*

Figure 1 describes a scheme whose iterative application from lines 1–17 defines an VS algorithm that finds a KKT point to the BCOP when $N = \bigcup_{j=1}^{p} M_j$, and these subsets are known or determined in advanced. For the sake of completeness, we now sketch its convergence proof, but we follow the same line of arguments used in the previous theorems. We ask the reader to pay attention to line numbers in Figure 1 for our proofs.

PROPOSITION 2.6 *Let* $M_1, \ldots, M_p$ *be given sets, and* $\bigcup_1^p M_j = N$. *Under Assumptions* A2–A4 *an iterative application of the scheme described in Figure 1 generates* $\{\tau_i\}_1^\infty \downarrow 0$.

Proof If the proposition is false, that is, there exists $\epsilon > 0$, and $(\tau_i > \epsilon)$ for all sufficiently large i, the line 17 is never executed from an iteration onwards, which implies that line 13 is executed finitely many times and line 8 generates a sequence of $f(\cdot)$ values unbounded from below, contradicting A2. ■

We have still not proved convergence to a KKT point of the BCOP. We need a previous lemma.

LEMMA 2.7 *Under Assumption* A1–A5, *the iterative application of the scheme described in Figure 1 generates a DQP on* $\mathbb{R}^n$ *whenever* $L = N$.

Proof Note that $L = \emptyset$ when the IF clause holds (lines 7–10). On the other hand, when the IF clause fails, the algorithm updates $L = L \cup M_i$ (lines 12–13). By the way M_i is built on line 5, the algorithm makes sure that $L = \bigcup_{j=i}^{i+p-1} M_j = N$ when the IF clause fails on p consecutive iterations starting at the ith iteration. Moreover, (x_i, τ_i, φ_i) do not change throughout these p

1. Given $x_i \in \mathcal{F}, \tau_i > 0, \varphi_i \geq f(x_i), \gamma > 1, \mu \in (0\ 1), \delta > 0,$
2. $\quad M_1, \ldots, M_p$
3. $L = \emptyset$
4. WHILE $(L \subset N)$
5. $\quad$ Choose $M_i = M_{(i \bmod p)+1}$
6. $\quad$ Construct $\mathcal{D}_i$ satisfying **A5**
7. $\quad$ IF $\exists d \in \mathcal{D}_i : y \stackrel{def}{=} (x_i + \tau_i d) \in \mathcal{F}$ AND $f(y) \leq \varphi_i - \sigma(\tau_i)$
8. $\quad\quad$ Choose any $x_{i+1} \in \mathcal{F}$ such that $\begin{cases} f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i), \\ ||x_{i+1} - y|| \leq \gamma\tau_i \end{cases}$
9. $\quad\quad$ Update $\varphi_i, \tau_{i+1} = \tau_i$
10. $\quad\quad L = \emptyset$
11. $\quad$ ELSE
12. $\quad\quad x_{i+1} = x_i,\ \varphi_{i+1} = \varphi_i,\ \tau_{i+1} = \tau_i$
13. $\quad\quad L = L \cup M_i$
14. $\quad$ ENDIF
15. $\quad i = i + 1$
16. ENDWHILE
17. $\tau_i = \mu\tau_i$

Figure 1. Scheme to find DQPs on a split $N = \bigcup_{j=1}^{p} M_j$.

iterations (line 12). Remark 1.4 and A5 ensure that $(x_i, \bigcup_{j=i}^{i+p-1} \mathcal{D}_j, \tau_i)$ is a discrete quasi-minimal frame and x_i is a DQP on $\mathbb{R}^n$. ■

THEOREM 2.8 *Under Assumptions* A1–A5, *the iterative application of the scheme described in Figure 1 generates an infinite subsequence of DQPs on* $\mathbb{R}^n$ *converging to a first-order KKT point of the BCOP.*

Proof Since $\{\tau_i\}_1^\infty \downarrow 0$, line 17 is executed an infinite number of times. As this happens when $L = N$, the previous lemma ensures that an infinite number of DQPs on $\mathbb{R}^n$ is generated and when x_i is a DQP, $\bigcup_{j=i}^{i+p-1} D_j$ positively spans $\mathbb{R}^n$. With this choice of directions and replacing M by N, the proof follows almost verbatim from Theorem 2.4. ■

The iterative application of the scheme described in Figure 1 might work well for structured problems, but in general, it might not be easy to choose a good splitting for N. Fortunately, convergence does not depend on an *a priori* choice of $M_1, \ldots, M_p$, and line 2 of Figure 1 can define different subsets before entering the WHILE loop on line 4. Convergence is preserved as long as the sequence $\{M_j\}_1^\infty$ satisfies

$$(\forall x_i \text{ DQP on } \mathbb{R}^n)(\exists p \text{ finite}) : \bigcup_{j=i}^{i+p-1} M_j = N. \tag{13}$$

So, we now propose an algorithm with dynamic space decomposition. At x_i, the $f(\cdot)$ values are sampled on the subspace $\mathcal{R}^{M_i} \neq \emptyset$. If x_i is a DQP on $\mathcal{R}^{M_j}$, a new subspace $\mathcal{R}^{M_{j+1}}$ is explored and a set L, defined as $L = \bigcup_{k=i}^{j+1} M_k$, indicates that x_i is a DQP on $\mathcal{R}^L$. When $L = N$, we obviously obtain that x_i is a DQP on $\mathbb{R}^n$. When x_i is not a DQP on $\mathcal{R}^{M_j}$, we set $L = \emptyset$ and the process is repeated for x_{i+1}. Note that M_j can be randomly generated and that p, the minimum number of sets whose union is N, might depend on the iteration i.

This procedure is included in the algorithm described in Figure 2, which is intended to be a pseudocode that can be translated into a computer program. As it is customary, the index i is dropped and in order to force finite termination, the procedure is enclosed by the outer WHILE loop that will be executed as long as $\tau > \epsilon$, where ϵ, the accuracy of the solution, must be supplied by the user. As $\{\tau_i\}_1^\infty \downarrow 0$ we are sure the algorithm will stop for any ϵ strictly positive.

Along the same arguments used for the previous sequential algorithms, we claim that the following propositions are valid.

PROPOSITION 2.9 *If* $\epsilon = 0$, *Equation* (13) *holds*, *and assumptions* A2–A4 *hold*, *the SADSD, described in Figure 2, generates the infinite sequence* $\{(x_i, \mathcal{D}_i, \tau_i, \varphi_i, M_i, L_i)\}_1^\infty$ *with* $\{\tau_i\}_1^\infty \downarrow 0$.

PROPOSITION 2.10 *If* $\epsilon = 0$, *Equation* (13) *holds*, *and assumptions* A1–A5 *hold*, *then there exists an infinite subsequence* K *such that* $\{x_i\}_{i \in K}$ *is a sequence of DQPs on* $\mathbb{R}^n$.

Remark 2.11 As N is finite we can extract $I \subseteq K$ with fixed values of p and subsets $M_1, \ldots, M_p$ such that

$$i \in I \Rightarrow \left\langle \begin{array}{l} \bigcup_{j=1}^{p} M_j = N, \quad \text{and} \\ M_{i+j-1} = M_j,\ j = 1, \ldots, p \end{array} \right.$$

PROPOSITION 2.12 (Convergence theorem) *If* $\epsilon = 0$, *Equation* (13) *holds*, *and Assumptions* A1–A5 *hold*, *SADSD generates a* (*sub*)*sequence of DQPs that converges to a KKT point of the BCOP.*

1. **input:** $x \in \mathcal{F}, \tau > 0, \varphi \geq f(x), \gamma > 1, \mu \in (0\ 1), \delta > 0,$
2. WHILE $(\tau > \epsilon)$
3. $\quad L = \emptyset$
4. $\quad$ WHILE $(L \subset N)$
5. $\quad\quad$ Choose $M \subseteq N : (M - L) \neq \emptyset$
6. $\quad\quad$ Construct $\mathcal{D}$ satisfying **A5**
7. $\quad\quad$ IF $\exists d \in \mathcal{D} : y = (x + \tau d) \in \mathcal{F}$ AND $f(y) \leq \varphi - \sigma(\tau)$
8. $\quad\quad\quad$ Choose any $x \in \mathcal{F}$ such that $\begin{cases} f(x) \leq \varphi - \sigma(\tau), \\ ||x - y|| \leq \gamma\tau \end{cases}$
9. $\quad\quad\quad$ Update φ
10. $\quad\quad\quad L \leftarrow \emptyset$
11. $\quad\quad$ ELSE
12. $\quad\quad$
13. $\quad\quad\quad L \leftarrow L \cup M$
14. $\quad\quad$ END IF
15. $\quad$
16. $\quad$ END WHILE
17. $\quad \tau \leftarrow \mu\tau$
18. END WHILE

Figure 2. SADSD: sequential algorithm with dynamic space decomposition.

3. Parallel algorithms

Partially separable functions commonly appear in the discretization of physical optimization problems [7, Section 4]. NM PS algorithms applied to this kind of functions are naturally parallelizable. Let us illustrate this claim with an example that describes a parallel scheme for minimizing a function of $n = 2p + 2$ variables with p partial sums, given by

$$f(x) = \sum_{j=1}^{p} f_j(x^{2j-1}, x^{2j}, x^{2j+1}, x^{2j+2}).$$

Define the VS with the constant sets $M_j = \{2j - 1, 2j, 2j + 1, 2j + 2\}, j = 1, \ldots, p$. Let $\mathcal{D}_{ij}^{+} = \mathcal{R}^{M_j}, j = 1, \ldots, p$ and $(\forall i \in \mathbb{N})(\varphi_i = f(x_i))$. Our algorithm only needs to compute differences in

function values, and for any $d \in \mathcal{D}_{ij}$ we have that

$$f(x+\tau d)-f(x)=\begin{cases}\displaystyle\sum_{k=1}^{2} f_k(x+\tau d)-f_k(x), & j=1\\ \displaystyle\sum_{k=j-1}^{k=j+1} f_k(x+\tau d)-f_k(x), & 1<j<p\\ \displaystyle\sum_{k=p-1}^{p} f_k(x+\tau d)-f_k(x), & j=p\end{cases} \tag{14}$$

which represents huge computational savings for large p. Besides, differences in function values on the subspace $\mathcal{R}^{M_j}$ are quite independent from differences in function values on $\mathcal{R}^{M_k}$ for $|k-j|\geq 3$. This suggests the following parallel scheme:

Given x_i:

Generate x_{i+1} in parallel on subspaces $\mathcal{R}^{M_1}, \mathcal{R}^{M_4}, \mathcal{R}^{M_7}, \ldots$
Generate x_{i+2} in parallel on subspaces $\mathcal{R}^{M_2}, \mathcal{R}^{M_5}, \mathcal{R}^{M_8}, \ldots$
Generate x_{i+3} in parallel on subspaces $\mathcal{R}^{M_3}, \mathcal{R}^{M_6}, \mathcal{R}^{M_9}, \ldots$
IF $x_{i+3}=x_i$, reduce τ

To ensure convergence, we must complete the work in parallel on subspaces $\mathcal{R}^{M_j}, \mathcal{R}^{M_{j+3}}, \ldots,$ before proceeding with the work in parallel on $\mathcal{R}^{M_{j+1}}, \mathcal{R}^{M_{j+4}}, \ldots$. Keeping this in mind, the convergence analysis is similar to its sequential counterpart and it is, therefore, omitted. This example somehow reveals that the space decomposition must conform to the structure of the problem at hand.

On PS methods, one can do some kind of parallelization in a straightforward manner. Earlier works constructed $\mathcal{D}_i$ such that $\mathcal{D}_i^+ = \mathbb{R}^n$ and processor j, say, computed $f(x_i+\tau_i d_j)$ for some specific $d_j \in \mathcal{D}_i$. A better estimate is then chosen to start the next iteration. Let $y=\operatorname{argmin}_{d\in\mathcal{D}_i} f(x_i+\tau_i d)$. If $f(y)<f(x_i)$ then $x_{i+1}=y$; otherwise τ_i is reduced [28, Figure 2.2]. A *minor* generalization is to link each processor with several directions of search as suggested in [20]. Nowadays it has been recognized that asynchronous versions are better suited for solving practical engineering problems in modern platforms. Recent software developments obey this paradigm for both unconstrained and constrained PS algorithms (see [6,23,28–30] and references therein).

Audet *et al.* [6] proposed a monotone PS asynchronous parallel algorithm with a space decomposition strategy for solving constrained non-smooth optimization problems. The current work differs from theirs in several ways: (i) We employ an NM strategy; (ii) Processors search successively on different subspaces and directions satisfying Remark 1.4, (iii) a processor only exchanges information with others when it finds a DQP; that is, when (12) holds, and (iv) only a finite set of directions of search is needed at all iterations. The algorithm suggested in [6] is monotone, it splits the variables in subsets, each determining a subproblem that is solved to a given accuracy, and then the corresponding results are shared. This gives rise to an increase in communication load. Theoretically, our algorithm converges for strictly differentiable functions and possesses a high degree of fault tolerance, a relevant issue that has already attracted the attention of other researchers [28]. On the other hand, Audet *et al.* [6] ensure convergence for general non-smooth functions, but they require an asymptotically dense set of directions; besides, their algorithm is based on a master-slave configuration, which does not offer a high degree of fault tolerance.

Finally, it is relatively easy to adapt the sequential algorithm to a multi-processing environment, specially to modern off the shelf PCs with two or four core, since all iterations in the partial subsets of variables can be regarded as a search step that tries to locate a feasible point satisfying (2).

When a processor does not find such a point on all subspaces, it communicates with the rest of the processors.

The piece of information that is exchanged among processors is the duplet (z,f_z), which represents the best prospective DQP z and its function value $f_z = f(z)$. We use $f_z = \infty$ to mean that the global variable does not exist at that moment, because it has been taken by another processor.

Figure 3 is a pseudo code that describes the PADSD. It is exactly the SADSD (Figure 2) with a communication phase in lines 17–25. At line 18, a processor fetches the DQP point $(z,f(z))$ and if it satisfies the SDC on line 19, it is taken as the next iterate on line 20. If, on the other hand, $f(x) < f(z)$, the processor informs the rest (line 24) that a better DQP is at hand. Line 22 is an artifice to prevent other computers from taking the same iterate.

Since each processor is running a SADSD, the convergence of the parallel algorithm is omitted. We claim that all processors converge to a KKT point of the BCOP; however, we may force termination as soon as any of the processors generates $\tau \leq \epsilon$, where $\epsilon > 0$ is a tolerance that can be set by the user. The parallel algorithm is a replica of the sequential algorithm except for the communication phase, that may occur in two critical points. In line 10, we may check if any other processor can provide an acceptable point, as suggested by Remark 2.1, but this option has not been implemented in this paper; mainly because the communication load can increase severely. Depending upon the parallel architecture, the global value $(z,f(z))$ may be a shared

1. **Initial values:** $\gamma > 1, \mu \in (0,\ 1), \epsilon > 0$
2. Global estimate (z, f_z)
3. $x \in \mathcal{F}, f_x \leftarrow f(x), \varphi \geq f_x, \tau > \epsilon, L \leftarrow \emptyset$
4. **i-th Iteration:** $(x, f_x, \varphi, \tau, L)$
5. WHILE $(\tau > \epsilon)$
6. **sequential iteration**
7. Choose $M \subseteq N : (M - L) \neq \emptyset$
8. `Build` $\mathcal{D}$ satisfying **A5**
9. IF $\exists d \in \mathcal{D} : y \leftarrow (x + \tau d) \in \mathcal{F},$ and $f(y) \leq \varphi - \sigma(\tau)$
10. `Choose any` $x \in \mathcal{F}$ such that $\begin{cases} f(x) \leq \varphi - \sigma(\tau), \\ ||x - y|| \leq \gamma\tau \end{cases}$
11. Update φ
12. $L \leftarrow \emptyset$
13. ELSE
14. $L \leftarrow L \cup M$
15. IF $(L = N)$
16. $\tau \leftarrow \mu\tau,\ L \leftarrow \emptyset$
17. **Communication**
18. **Fetch** (z, f_z)
19. IF $f_z < \varphi - \sigma(\tau)$
20. $x \leftarrow z, f_x \leftarrow f_z$
21. Update φ
22. $f_z \leftarrow \infty$
23. ELSEIF $f_x < f_z$
24. **Broadcast or deposit** (x, f_x)
25. ENDIF
26. ENDIF
27. ENDIF
28. END WHILE

Figure 3. PADSD: parallel algorithm with dynamic space decomposition (single process iteration and communication).

memory, or it can be provided by any other means best appropriated for the multi-processing architecture.

We emphasize that the communication phase (lines 17–25) is not needed for convergence. If this phase fails, the algorithm becomes embarrassingly parallel. All processors will run until termination. Convergence is not impaired if some processor(s) fail to execute its communication phase. We stress this fact because the algorithm still converges even if only one processor is running, so it exhibits a high degree of fault tolerance.

4. Implementation issues

We have described an algorithmic framework with several parameter values that may become relevant for a good performance. This section discusses and justifies the parameters chosen in our implementation, albeit these values can be freely set by the user.

4.1 *Input data*

The algorithm needs the input variables x_0, τ_0, φ_0, the bounds s, t and a function to compute $f(\cdot)$ to start the first iteration; $\tau_0 = 1, \varphi_0 = f(x_0) + 0.2(|f(x_0)| + 1)$ are the default values in our implementation; for unconstrained problems, the algorithm chooses $t^k = 100, k = 1, \ldots, n$ and $s = -t$.

If a global minimum is desired then τ_0 and φ_0 should be big enough in order to explore more broadly the space surrounding x_0. A value of μ close to 1 and thrifty reductions to $\{\varphi_i\}$ have a similar effect. We should keep in mind that these strategies inevitably increase the number of function evaluations, so moderate values are preferable in general.

4.2 *Stopping criteria*

It has been proved that $\{\tau_i\}_1^\infty \downarrow 0$ under suitable conditions; hence, $\tau_i < \epsilon_i = \bar{\epsilon} \max(\|x_i\|, \bar{\epsilon})$ will eventually hold after a finite number of iterations for any given $\bar{\epsilon} > 0$. Unless otherwise stated, we use $\bar{\epsilon} = 10^{-5}$. As a safeguard, we limit the number of function evaluations to 20K, in case some assumption is not fulfilled.

In parallel processing, we should force a termination for all processors as soon as one of them converges to a solution. There are other termination schemes suitable to users that prefer further exploration; we could for instance force termination only when a predefined number of processors have converged with $\tau_i < \epsilon_i$.

4.3 *Choosing x_{i+1}*

Once we found $x_{i+1} \neq x_i$ by (11) or Remark 2.1, we accept it only if

$$|f(x_i) - f(x_{i+1})| \geq \hat{\epsilon} \max(|f(x_i)|, \hat{\epsilon}).$$

We chose $\hat{\epsilon} = 10^{-4}$. This artifice avoids large number of function evaluations on ridges.We advise the reader to be cautious with the choice of $\hat{\epsilon}$. Smaller values may degrade the performance of the algorithm for some of the benchmark problems.

4.4 *Long steps*

Once we identify $y = x_i + \tau_i d \in \mathcal{F}$ with $f(y) \leq \varphi_i - \sigma(\tau_i)$ for some $d \in \mathcal{D}_i$, we search for a feasible point that also ensures this sufficient decrease on φ_i. We tried to force a long step along d.

Let $\mathcal{P}_{\mathcal{F}}(v)$ be the projection of v on the feasible set $\mathcal{F}$, and let $z_j = \mathcal{P}_{\mathcal{F}}(x_i + 2^j\tau_i d)$, for $j \in \{0,\ 1,\ 2,\ 3\}$. For any given j, this projection is merely the component wise median of vectors s, $(x_i + 2^j\tau_i d)$, and t. We chose $x_{i+1} = z_j$, where j is the smallest integer for which $f(z_j) < f(z_{j+1})$. If this inequality does not occur, we chose $x_{i+1} = z_3$. We did not employ more elaborate schemes, because they generally demand extra function evaluations; whichever scheme is adopted, it is advisable that $\tau_i \leq \|x_{i+1} - x_i\| \leq \gamma\tau_i$, for some $\gamma > 1$.

4.5 *Updating of* τ

As soon as a DQP on $\mathbb{R}^n$ is detected τ_i is reduced, namely, $\tau_{i+1} = \mu\tau_i$. We chose $\mu = 0.3$, although $\mu \in [0.2\ 0.6]$ does not seem to influence the performance of the algorithm significantly. On the other hand, when x_{i+1} is found by (11) or Remark 2.1, μ may be increased, as long as it is kept finitely bounded. The proofs of convergence remain valid with minor changes, but we decided not to include this extra feature in our theory.

4.6 *Updating of* φ

We suggest $\varphi_{i+1} = (1-\beta)f(x_{i+1}) + \beta\varphi_i$, for some $\beta \in [0\ 0.9]$. With this, updating the basic iteration (11) of the algorithm makes sure that

$$\left.\begin{array}{l} \text{IF some } (y \in \mathcal{F}) \text{ satisfying SDC is found} \\ \quad \text{Choose } x_{i+1} \in \mathcal{F} : f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i) \\ \quad \text{Choose } \beta \in [0\ 0.9], \text{ and} \\ \quad \varphi_{i+1} = (1-\beta)f(x_{i+1}) + \beta\varphi_i \\ \text{ELSE} \\ \quad x_{i+1} = x_i \\ \quad \varphi_{i+1} = \varphi_i \\ \text{END} \end{array}\right\} \tag{15}$$

PROPOSITION 4.1 *Let us assume that* $\sigma(\cdot)$ *satisfies* A3. *If* $\varphi_0 \geq f(x_0)$, *the scheme* (15) *generates a sequence* $\{\varphi_i\}_{i\in\mathbb{N}}$ *satisfying* A4.

Proof We first prove by induction that $\varphi_i \geq f(x_i)$ for all $i \in \mathbb{N}$. When the IF clause in (15) holds, we have that $\varphi_i \geq f(x_{i+1}) + \sigma(\tau_i) \geq f(x_{i+1})$, and

$$\varphi_{i+1} = (1-\beta)f(x_{i+1}) + \beta\varphi_i \geq (1-\beta)f(x_{i+1}) + \beta f(x_{i+1}) = f(x_{i+1}).$$

When the ELSE part holds, we have by the induction hypothesis that $\varphi_{i+1} = \varphi_i \geq f(x_i) = f(x_{i+1})$.

For the rest of the proof, we have from (15) that *either* $\varphi_{i+1} = \varphi_i$ or for some $\beta \in [0\ 0.9]$

$$\begin{aligned} \varphi_{i+1} &= (1-\beta)f(x_{i+1}) + \beta\varphi_i \\ &\leq (1-\beta)(\varphi_i - \sigma(\tau_i)) + \beta\varphi_i \\ &= \varphi_i - (1-\beta)\sigma(\tau_i). \end{aligned}$$

We now define $\sigma'(\tau) = (1-\beta)\sigma(\tau)$ and assert that A4 is fulfilled with $J = I$. ■

A value of β close to 1 is preferable when we want to escape from a local minimum; otherwise, a value close to zero is recommended. Note that $\beta = 0$ implies a sufficient decrease of the objective function. We also point out that $\beta = 1$ can be occasionally chosen, as long as we can identify a

set J satisfying A4. This strategy however, may induce a larger number of function evaluations because the algorithm is prone to frequent *hill climbing*, but it is an acceptable option if we want to escape from a local minimum. We recall that in [21], the authors searched for a global minimum. They used $\beta = 0$ every five iterations where $f(x_{i+1}) \leq \varphi_i - \sigma(\tau_i)$ held, and for the rest of the iterations they chose $\beta = 1$.

We implemented two updatings: $\beta = 0$ for the monotone version, and β randomly chosen in the interval [0 0.2] for the NM version.

4.7 *NM version*

The algorithm always records (x_b, τ_b), the best point found and its associate τ_b. When the stopping criterium 4.2 holds at x_i, we compare $f(x_i)$ with $f(x_b)$. If $f(x_i) > f(x_b)$ the algorithm jumps back to x_b and forces a sufficient decrease on $f(x_b)$. Specifically, $x_{i+1} = x_b, \varphi_{i+1} = f(x_b), \tau_{i+1} = \tau_b$. This artifice ensures $f(x_j) \leq f(x_b)$, for all $j \geq i + 1$.

4.8 *Space decomposition*

For partially separable functions and other structured problems, we split N into p convenient sets $M_1, \ldots, M_p, \bigcup_{j=1}^{p} M_j = N$ and follow Figure 1. When no structure is apparent, there seems to be no efficient way to separate the variables [6, section 9] and we randomly generate $M_i \not\subseteq L_i$ with a predefined number of elements. We point out that some problems demand some specific decomposition for better performance (see problem 16 in Section 5.2.4). Therefore, there is no general suitable decomposition scheme for all cases.

4.9 *Construction of $\mathcal{D}$ by* (10)

We generate a unitary vector $u \in \mathcal{R}^A$ and define the set of directions

$$d_j = e_j - 2u^j u, j \in M$$

Observe that the set of directions $d_j, j \in M$ is orthonormal, although orthogonality is not necessary. Finally, we define $\mathcal{D} = \{d_j, -d_j : j \in M\}$, because previous numerical results have shown that this is a good strategy [4,20], and it is a good safeguard for non-smooth $f(\cdot)$. Observe that any accumulation point u_* will generate $\mathcal{D}_*^+ = \mathcal{R}^M$ as required.

We obtain $u_i \in \mathcal{R}^A$ as the normalized difference of the last two DQP points. If they coincide, we randomly generate a unitary vector u.

4.10 *Communication*

Fetching (z, f_z) depends upon the computer architecture. For a multi-thread processor, information could be shared among all threads and (z, f_z) is a global memory that can be accessed by all threads. However, global variables are ruled out in an environment made up by a distributed set of processors running on networked computers.

In MatlabMPI, each processor fetches (z, f_z) from its own private queue, which contains the DQPs broadcast by other processors. Once a processor fetches one element, the queue is emptied. It was evident that many processors fetch the same point (z, f_z). From there, identical sample values would be generated unless directions and subspaces are randomly chosen. A strategy that we did not include in the preliminary numerical tests is to assign a probability to fetch (z, f_z).

5. Numerical results

We report our results for both the sequential and parallel versions of our algorithm and compare them with the NOMAD package [33], an implementation of the MADS algorithm that can be freely downloaded from [2]. In all tests, we followed the guidelines described in Section 4. In all tables $f_{\min}$ represents the minimum function value and f_{vals} stands for the number of function evaluations. The majority of the tables report mean values.

The NOMAD package is highly customizable and its performance is heavily linked to the chosen parameters and settings. Initially, we worked with the deterministic version and $\mathcal{D}_i = \{\pm e_j, j = 1, \ldots, n\}$ for all $i \in \mathbb{N}$. We obtained discouraging results that are not reported here. Thereafter, we run NOMAD in batch mode with the default settings and tolerances described in the NOMAD user guide [34, Chapter 5]. In particular, we selected an orthogonal set of poll directions, which are generated and sampled randomly, a convergence tolerance of 10^{-5}, a maximum number of function evaluations of 20,000 and a cache tolerance of 10^{-4}.

5.1 *Results for the sequential algorithm*

To observe the performance of our sequential algorithm, we coded it in Matlab version 7 and run it on Windows 7 with a 1.73 GHz QuadCore. We solved the benchmarking set of 22 problems for DF optimization algorithms proposed in [42,43], but we set $30 \leq n \leq 300$ on those instances, where the number of variables n can be chosen by the user. The initial points were those recommended for the original problems filled with additional zeros. The algorithm performed well for values of n up to 300 in the problems tested. We do not report the timing for the sequential algorithms, but it is worth mentioning that our algorithm only needed less than 30 s to solve the smooth unconstrained version of the whole set of benchmarking problems, while NOMAD spent more than 217 min. Unless otherwise stated, NOMAD was run 20 times and our algorithm 100 times for each experiment.

5.1.1 *Unconstrained problems*

Table 1 gives the results on the 22 problems used in benchmarking DF optimization algorithms. Our monotone algorithm never failed and, except for problems 11 and 17, it solved the problems with an impressive reduction of function evaluations with respect to MADS. In only five problems, MADS found a better minimum function value, on average. Table 1 also includes as a reference the minimum function value found by fminunc, a Matlab function that uses derivative information.

5.1.2 *Bound constraints*

We used the same set of benchmarking problems suggested for DF optimization. We add the additional bound constraints $x_0 - v \leq x \leq x_0 + v$, where x_0 is the initial point and $v^k = 3, k = 1, \ldots, n$. This ensures a non-empty feasible set. The results of our experiments for the minimum function value and for the number of function evaluations are given in Table 2. Again, the monotone version clearly dominates NOMAD. As we implemented the NM algorithm with moderate values, $\mu = 0.3$ and $\beta \leq 0.2$, the differences between the monotone and the NM versions are not too pronounced. The space decomposition version gives accurate results, although the number of function evaluations increases.

5.1.3 *Noisy functions*

We also solve the same set of problems with the high frequency deterministic noise suggested in [42,43]. Let $\phi = 0.9 \sin(100\|x\|_1) \cos(100\|x\|_\infty) + 0.1 \cos(\|x\|_2)$, and let $\vartheta = \phi(4\phi^2 - 3)$. The

Table 1. Performance on UNCONSTRAINED Moré Wild benchmarking for differentiable functions (average minimum function values and function evaluations).

			MATLAB	MADS algorithm		Monotone algorithm	
Prob.	n	m	fminunc	$f_{\min}$	f_{vals}	$f_{\min}$	f_{vals}
1	300	500	200.00	2.00e+02	14,400.6	2.00e+02	8681.5
2	300	500	1.25e+02	1.29e+02	14,429.1	1.45e+02	6289
3	300	500	1.26e+02	1.70e+04	15,098.6	1.28e+02	6444.6
4	2	2	2.85e−11	6.38e−04	2265.8	1.09e−06	345.2
5	3	3	2.25e−11	2.62e+00	2553.7	2.12e−07	599.1
6	4	4	6.39e−09	2.19e−05	2708.1	5.74e−08	1317.5
7	2	2	4.89e+01	5.62e+01	16,681.1	4.90e+01	155.2
8	3	15	8.21e−03	8.66e−03	11,138.3	8.22e−03	571
9	4	11	3.08e−04	3.31e−04	8668.4	3.08e−04	1478.1
10	3	16	1.12e+05	7.12e+04	6797.3	1.82e+09	129.1
11	9	31	3.84e−05	1.19e−03	17,732	8.88e−04	16,707.5
12	30	100	9.19e−05	2.52e−04	12,522.2	3.61e−06	1917
13	10	30	2.74e+04	5.89e+03	2131.1	6.56e+03	554.2
14	20	30	9.77e+08	9.98e+08	17,890.15	9.78e+08	1372.4
15	30	50	2.36e−02	4.58e−02	19,673.4	3.05e−02	5487.6
16	30	50	4.40e−07	3.78e−01	FAIL[a]	4.28e−01	16,140.1
17	5	33	8.62e−03	3.63e−03	4780.6	2.86e−01	4994.1
18	11	65	4.01e−02	4.07e−02	19,798.8	4.02e−02	3335.3
19	300	592	1.18e+03	2.65e+03	FAIL[a]	1.18e+03	18,623
20	2	2	1.05e−10	3.88e−01	4024.6	7.01e−07	599.8
21	30	30	6.72e−04	6.05e+04	FAIL[a]	2.11e+01	1874.1
22	8	8	9.78e−09	3.94e−02	FAIL[a]	2.06e−05	5690.4

[a]More than 20,000 evaluations.

noisy function $f_n(\cdot)$ is given by

$$f_n(x) = (1 + 0.001\vartheta)f(x).$$

The noise may introduce a large number of minima, and the purpose of this experiment was to find out whether monotone and NM versions of the algorithm were able to escape from a local minimum. Table 3 shows the results where we have highlighted with *bold figures* the best result obtained for each problem. The NM version reported the best value for $f_{\min}$ in 9 problems, whereas the monotone version gave the best value for f_{vals} in 16 problems. MADS performance was encouraging for this experiment.

5.2 *Results for the parallel algorithm*

We report the performance of the parallel algorithm on the subset of the benchmarking DF problems with $n \geq 30$, and compare our results with PSD-NOMAD on a test with 100 runs for each problem. We also solved a structured problem with more than 1000 variables and a problem subject to nonlinear constraints. We used several environments and strategies that are described below.

5.2.1 *MatlabMPI*

We simulate four processes on the MatlabMPI software developed by [45], which is claimed to be suitable for algorithms with intensive parallel loops [9]. In the communication phase, each processor fetches a DQP from a queue of DQPs broadcast by other processors. The concept of a shared global variable is not present. These experiments were run with Windows XP professional on an Intel(R) Core(TM)2 Quad CPU Q9400 at 2.66 GHz.

Table 2. Performance on box constrained problems.

#	MADS	Our algorithm		
		Monotone	p subspaces[a]	NM
(a) Average and range min : max of $f_{\min}$ values: average / min : max				
1	2.00e+02 2.00e+02 : 2.00e+02	2.00e+02 2.00e+02 : 2.00e+02	2.00e+02 2.00e+02 : 2.00e+02	2.00e+02 2.00e+02 : 2.00e+02
2	1.46e+02 1.25e+02 : 3.70e+02	1.25e+02 1.25e+02 : 1.26e+02	1.26e+02 1.25e+02 : 1.29e+02	1.25e+02 1.25e+02 : 1.25e+02
3	4.52e+02 1.26e+02 : 6.55e+02	1.26e+02 1.26e+02 : 1.26e+02	1.27e+02 1.26e+02 : 1.42e+02	1.26e+02 1.26e+02 : 1.26e+02
4	4.36e−04 5.26e−08 : 2.46e−03	8.24e−07 1.03e−10 : 6.52e−06	1.14e−04 7.83e−05 : 1.51e−04	7.13e−08 6.58e−11 : 7.76e−07
5	7.92e−05 4.84e−07 : 2.66e−04	1.12e−07 1.08e−09 : 1.32e−06	3.09e−01 0.00e+00 : 6.19e+00	2.66e−08 1.02e−09 : 2.52e−07
6	2.41e−05 1.35e−06 : 1.20e−04	6.71e−08 3.70e−10 : 2.03e−07	9.66e−08 9.66e−08 : 9.66e−08	6.22e−08 2.21e−10 : 2.67e−07
7	7.47e+01 7.47e+01 : 7.47e+01	7.47e+01 7.47e+01 : 7.47e+01	7.47e+01 7.47e+01 : 7.47e+01	7.47e+01 7.47e+01 : 7.47e+01
8	8.84e−03 8.21e−03 : 1.12e−02	8.22e−03 8.21e−03 : 8.26e−03	8.84e−03 8.28e−03 : 1.36e−02	8.22e−03 8.21e−03 : 8.23e−03
9	3.56e−04 3.08e−04 : 5.70e−04	3.08e−04 3.08e−04 : 3.08e−04	3.09e−04 3.08e−04 : 3.09e−04	3.28e−04 3.08e−04 : 7.22e−04
10	4.58e+06 4.18e+06 : 8.26e+06	1.31e+09 4.28e+06 : 1.45e+09	1.45e+09 1.45e+09 : 1.45e+09	1.23e+09 6.09e+06 : 1.45e+09
11	9.54e−03 3.91e−04 : 5.16e−04	5.55e−04 8.51e−06 : 2.98e−03	2.21e−03 5.59e−05 : 1.22e−02	5.16e−04 2.37e−05 : 3.47e−03
12	8.40e+02 8.39e+02 : 8.47e+02	8.39e+02 8.39e+02 : 8.39e+02	8.39e+02 8.39e+02 : 8.47e+02	8.39e+02 8.39e+02 : 8.39e+02
13	7.22e+03 5.88e+03 : 8.58e+03	6.29e+03 5.89e+03 : 8.58e+03	6.02e+03 5.89e+03 : 8.57e+03	6.42e+03 5.89e+03 : 8.58e+03
14	1.81e+10 1.80e+10 : 1.81e+10	1.80e+10 1.80e+10 : 1.80e+10	1.80e+10 1.80e+10 : 1.80e+10	1.80e+10 1.80e+10 : 1.80e+10
15	2.93e−02 2.43e−02 : 4.58e−02	3.37e−02 2.36e−02 : 4.75e−02	2.97e−02 2.36e−02 : 5.99e−02	3.33e−02 2.31e−02 : 4.83e−02
16	1.66e+00 1.38e−01 : 5.55e+00	2.57e−01 3.77e−06 : 1.29e+00	3.88e+00 2.59e+00 : 6.77e+00	6.41e−02 2.35e−06 : 1.28e+00
17	6.24e−02 3.27e−04 : 4.99e−01	2.84e−01 5.60e−05 : 1.11e+00	1.25e−01 6.51e−05 : 1.11e+00	5.01e−01 5.71e−05 : 1.11e+00
18	2.08e−01 4.66e−02 : 2.17e−01	4.66e−02 4.66e−02 : 4.67e−02	4.90e−02 4.66e−02 : 9.29e−02	6.53e−02 4.66e−02 : 2.13e−01
19	4.04e+04 3.07e+04 : 5.33e+04	1.19e+03 1.18e+03 : 1.29e+03	1.18e+03 1.18e+03 : 1.18e+03	1.18e+03 1.18e+03 : 1.19e+03
20	1.48e−03 1.23e−05 : 4.70e−03	3.21e−06 2.88e−10 : 3.61e−05	8.58e−04 6.55e−04 : 1.06e−03	1.56e−07 4.57e−11 : 2.20e−06
21	2.31e+09 2.25e+09 : 2.52e+09	2.24e+09 2.24e+09 : 2.24e+09	2.24e+09 2.24e+09 : 2.24e+09	2.24e+09 2.24e+09 : 2.24e+09
22	6.01e−02 2.44e−03 : 3.11e−01	1.92e−05 1.24e−06 : 8.51e−05	2.39e−04 1.83e−06 : 1.17e−03	9.96e−06 1.17e−06 : 4.91e−05

(*Continued*)

Table 2. Continued

#	MADS	Our algorithm		
		Monotone	p subspaces[a]	NM
(b) Average and range `min:max` of f_{vals} values:		`average` / `min : max`		
1	8218 8036 : 8758	9428 5637 : 13,174	11,370 6987 : 14,461	15,186 10,131 : FAIL[b]
2	11,518 8043 : FAIL[b]	8636 7251 : 9794	6671 5673 : 7771	17,557 12,549 : FAIL[b]
3	13,256 10,067 : FAIL[b]	8857 7130 : 10,830	6526 5441 : 7945	16,159 12,693 : FAIL[b]
4	3338 230 : 13,758	373 198 : 598	FAIL[b] FAIL[b] : FAIL[b]	503 292 : 1254
5	2567 373 : 6502	454 240 : 999	8726 95 : 12,685	662 477 : 998
6	3215 1373 : 6538	1388 557 : 7017	3850 3821 : 3867	1407 574 : 3160
7	124 75 : 161	131 84 : 191	106 84 : 121	196 133 : 264
8	15,976 2561 : FAIL[b]	526 206 : 1083	13,146 2197 : FAIL[b]	648 386 : 1033
9	13,766 1981 : FAIL[b]	1372 323 : 2385	5936 3092 : 9668	3119 568 : 9634
10	1270 96 : 4546	44 43 : 50	46 42 : 48	89 62 : 128
11	19,556 14,523 : FAIL[b]	16,507 5971 : FAIL[b]	19,603 11,978 : FAIL[b]	16,895 7107 : FAIL[b]
12	6152 1053 : FAIL[b]	820 650 : 1110	745 640 : 852	1295 943 : 2217
13	8777 480 : FAIL[b]	583 262 : 850	1146 345 : 1692	826 351 : 1349
14	2420 561 : 11,050	550 472 : 664	467 380 : 571	1007 590 : 1366
15	8717 5224 : 17,850	5091 2236 : 10,141	9076 4035 : 14,493	10,993 4481 : FAIL[b]
16	FAIL[b] FAIL[b] : FAIL[b]	17193 10229 : FAIL[b]	FAIL[b] FAIL[b] : FAIL[b]	17,175 7723 : FAIL[b]
17	7501 1250 : FAIL[b]	5468 188 : FAIL[b]	6626 222 : FAIL[b]	4781 223 : FAIL[b]
18	1951 1537 : 9714	2443 1551 : 3479	6163 2499 : FAIL[b]	3394 1791 : 7170
19	FAIL[b] FAIL[b] : FAIL[b]	18,336 14613 : FAIL[b]	FAIL[b] FAIL[b] : FAIL[b]	FAIL[b] 16,698 : FAIL[b]
20	5437 106 : 15,235	555 135 : 921	FAIL[b] FAIL[b] : FAIL[b]	596 297 : 822
21	8079 3243 : 18,911	878 783 : 997	890 828 : 949	1782 1487 : 2056
22	19,986 14,598 : FAIL[b]	5561 2671 : 11,740	18,858 14,355 : FAIL[b]	6834 2845 : 11,787

[a] $p = \text{ceiling}(n/4)$.
[b] More than 20,000 evaluations.

For any given $q > 0$ let $M_j = \{(j-1)q + i, i = 1, \ldots, q\}, j = 1, \ldots, p$, with $p = \text{ceiling}(n/q)$. If necessary M_p is completed randomly to make sure $|M_p| = q$. As we assume no previous knowledge about the structure of the problem, we arbitrarily chose $q = 7$.

Table 3. Performance on NOISY Moré Wild benchmarking for unconstrained differentiable functions (function value and function evaluations, average results).

			Our algorithm			
	MADS algorithm		Monotone		NM	
#	$f_{\min}$	f_{vals}	$f_{\min}$	f_{vals}	$f_{\min}$	f_{vals}
1	2.00e+02	18,147	2.00e+02	**9673**	2.00e+02	16,896
2	1.46e+02	14,741	2.19e+02	**6199**	**1.27e + 02**	14,812
3	1.42e+02	18,003	**1.27e + 02**	**6450**	1.33e+02	13,954
4	9.44e-01	1710	4.90e−07	**412**	**2.71e − 08**	527
5	6.88e+00	1430	3.72e−08	**560**	**3.69e − 08**	762
6	2.73e−05	2329	4.39e−08	1632	**4.27e − 08**	**1499**
7	1.35e+02	468	4.92e+01	**166**	**4.91e + 01**	288
8	**1.11e − 02**	**333**	1.27e−02	445	6.42e−02	535
9	4.18e−04	**909**	**3.26e − 04**	1244	6.25e−04	981
10	**2.34e + 05**	3624	3.89e+09	**49**	3.80e+09	184
11	6.52e−03	18,036	**3.17e − 04**	16,203	1.31e−03	**15,777**
12	**2.00e − 04**	13,571	1.54e−02	**1664**	9.44e−03	1986
13	**6.01e + 03**	2170	6.95e+03	**456**	7.41e+03	746
14	9.78e+08	6928	9.78e+08	**1611**	9.78e+08	1768
15	6.63e−02	14,066	**2.90e − 02**	**4787**	3.12e−02	8758
16	**2.50e − 01**	FAIL[a]	2.95e−01	16,124	5.14e−01	**15,998**
17	**7.55e − 02**	3855	2.34e−01	6160	7.24e−01	**1712**
18	**4.12e − 02**	10,992	4.40e−02	**2665**	5.27e−02	3419
19	2.65e+03	FAIL[a]	1.23e+03	**18, 685**	**1.21e + 03**	FAIL[a]
20	2.89e−02	1382	2.37e−06	**532**	**2.35e − 07**	677
21	3.60e+04	FAIL[a]	2.23e+01	**2064**	**2.14e + 01**	3370
22	3.86e−02	13,217	1.45e−05	**5779**	**7.84e − 06**	7266

[a]More than 20,000 evaluations.

The results are shown in Table 4. We remind the reader that once one processor ends, the minimization stops and other processors do not necessarily return the minimum function value. In general, the total number of function evaluations is bigger than the sequential algorithm, whilst the computing time, roughly estimated with the maximum number of function evaluations computed by any processor, decreases. It is worth mentioning that the MatlabMPI–Windows combination may overfill some buffers that causes a processor to break down. This unpleasant

Table 4. Performance of PARALLEL algorithm on BOUNDED Moré Wild benchmarking functions.

	Parallel algorithm with four machines in MatlabMPI							
	Processor 0		Processor 1		Processor 2		Processor 3	
Prob(n)	$f(x)$	f_{vals}	$f(x)$	f_{vals}	$f(x)$	f_{vals}	$f(x)$	f_{vals}
1(300)	2.01e+02	17,337	2.01e+02	17,490	2.01e+02	17,019	2.01e+02	16,929
2(300)	1.25e+02	5995	1.25e+02	6482	1.25e+02	6328	1.25e+02	6961
3(300)	1.26e+02	7189	1.26e+02	6034	1.26e+02	6443	1.26e+02	7229
12(30)	8.39e+02	872	8.39e+02	764	8.39e+02	688	FAIL[a]	
15(30)	2.59e−02	2387	FAIL[a]		2.58e−02	2438	2.59e−02	2651
16(30)	1.47e−03	FAIL[b]	4.51e−04	FAIL[b]	5.19e−04	FAIL[b]	6.18e−04	FAIL[b]
19(300)	1.21e+03	FAIL[b]	1.21e+03	FAIL[b]	1.43e+03	FAIL[b]	1.19e+03	FAIL[b]
21(30)	FAIL[a]		2.24e+09	1001	2.24e+09	988	FAIL[a]	

Notes: N was decomposed in $p = \text{ceiling}(n/7)$ subsets of equal size (function value and function evaluations).
[a]Overfilled buffer.
[b]More than 20,000 evaluations.

situation occurred for processor 1 on problem 15; fortunately, this is irrelevant for our high level of fault tolerance exhibited by the parallel algorithms described.

In order to obtain different paths to convergence, we seed the sequence of random numbers with the instruction `rand('state',sum(100*(clock+rank)))`, where rank is the processor number.

5.2.2 *PSD*-NOMAD

We also coded the parallel algorithm in C compiled with gcc version 4.4.5 on Ubuntu Linux 2.6 on a PC equipped with Intel(R) Xeon(R) CPU X5460 at 3.16 GHz with 8 cores. Finally, we used shared memory to access and write the global estimate defined in algorithm 2 and employed a MUTual EXclusion device (MUTEX) to protect the global variable from concurrent modifications, which prevent data inconsistency. We compare our algorithm with PSD-NOMAD, the parallel version of MADS. We used all of its default parameters; in particular,

$$\text{INITIAL\_MESH\_SIZE} = 1, \quad \text{and} \quad \text{MIN\_MESH\_SIZE} = 10^{-5}.$$

In both algorithms, we omit the limit of 20 K function evaluations.

Due to the randomness involved and to get a better comparison test, we run the parallel algorithm 100 times. We also generate M randomly with a fixed number of components satisfying line 7 of the parallel algorithm described in Figure 3.

The results on 100 runs for each problem are collected in Tables 5 and 6. We observe in Table 5 that on average the PSD-NOMAD algorithm never found a better value than ours. Also, the range of values found by PSD-NOMAD algorithm was wider than that found by ours; however, on average PSD-NOMAD required fewer function evaluations.

PSD-NOMAD reports lower function evaluations but worse function values than our algorithm on problems 1, 2, 3, 15, 16 and 19. However, the PSD-NOMAD performance on problems 2 and 3 was rather erratic. It generally failed on these problems. We repeat the tests on problems 1, 15, 16 and 19, but we forced termination of our algorithm at the – greater – function value found by the PSD-NOMAD. The results for this experiment are given in Table 6. Both the function value and function evaluations are better for our algorithm.

We must point out that the elapsed time is always lower for our algorithm and, except for problems 19 and 21, the elapsed time for PSD-NOMAD is at least 100 times the elapsed time spent by our algorithm. It is not clear how to explain this gap, but it is worth noting that PSD-NOMAD has a *cache* processor that is consulted each time a trial point must be evaluated [6, Section 3.4]. Our algorithm, on the other hand, only exchanges information at DQPs, which only appear when τ is reduced. If the initial τ value is 1, if the final τ value is 10^{-6}, and if the reduction factor is $\mu = 0.3$, we are considering no more than 12 DQPs.

Upon summarizing, we are certain that our parallel algorithm is a clear competitor.

5.2.3 *Medium-sized problem*

To observe the performance of the algorithm on medium-sized problems, we use MatlabMPI with 4 processors to solve the torsion problem with more than 1000 variables. Details on problem formulation and its physical interpretation can be found in the open literature [7], though we now try to accurately formulate the problem solved. Let $p > 0$ be a positive integer that defines a regular grid of $(p+2) * (p+2)$ points on the square $[0\ \ 1] \times [0\ \ 1]$

Table 5. Range of values obtained in 100 runs of PSD-NOMAD with eight processors and our algorithm with six threads ($f_{\min}$ value, function evaluations, and elapsed time).

	Range of values		
	PSD-NOMAD	Our algorithm ($q = 2$)	Our algorithm ($q = n/6$)
	Average min:max	Average min:max	Average min:max
Problem 1 ($n = 300$)			
$f_{\min}$	4.42e+02	2.01e+02	2.01e+02
	2.21e+02:1.30e+03	2.01e+02:2.01e+02	2.01e+02:2.02e+02
f_{vals}	6.71e+03	9.86e+04	2.29e+05
	702:11,865	77,660:125,368	210,397:247,036
Elapsed time	4.73	0.39	1.01
(seconds)	1.39:8.28	0.30:0.50	0.96:1.06
Problem 2 ($n = 300$)			
$f_{\min}$	1.15e+15	1.38e+02	1.85e+02
	1.25e+02:3.49e+16	1.25e+02:3.49e+02	1.25e+02:1.69e+03
f_{vals}	1.13e+03	2.38e+04	2.86e+04
	480:1385	18,751:31,577	21,197:34,603
Elapsed time	1.59	0.1	0.23
(seconds)	0.57:2.60	0.08:0.13	0.20:0.26
Problem 3 ($n = 300$)			
$f_{\min}$	1.89e+15	1.35e+02	1.55e+02
	1.27e+02:3.57e+16	1.27e+02:2.58e+02	1.27e+02:7.71e+02
f_{vals}	1.12e+02	2.43e+04	2.86e+04
	426:1419	18,199:32,716	22,724:35,284
Elapsed time	1.61	0.10	0.23
(seconds)	1.25:2.63	0.07:0.13	0.20:0.26
Problem 12 ($n = 30$)			
$f_{\min}$	8.39e+02	8.39e+02	8.39e+02
	8.39e+02:8.39e+02	8.39e+02:8.39e+02	8.39e+02:8.39e+02
f_{vals}	3.79e+02	3.51e+03	3.37e+03
	332:541	2752:4445	2688:4019
Elapsed time	1.28	0.02	0.02
(seconds)	1.22:2.27	0.02:0.02	0.01:0.02
Problem 15 ($n = 30$)			
$f_{\min}$	3.66e−02	2.77e−02	2.78e−02
	2.58e−02:4.79e−02	2.39e−02:5.03e−02	2.36e−02:4.59e−02
f_{vals}	5.15e+03	1.40e+04	1.25e+04
	7.02e+02:1.19e+04	7.77e+04:1.25e+05	2.10e+05:2.47e+05
Elapsed time	60.43	0.05	0.05
(seconds)	8.56:151.47	0.02:0.13	0.02:0.08
Problem 16 ($n = 30$):			
$f_{\min}$	1.47	0.130	0.113
	2.05e−04:14.62	5.66e−06:1.31	1.73e−05:3.41
f_{vals}	7.56e+03	2.49e+04	2.59e+04
	2128:16,626	14,502:38,003	11,963:49,181
Elapsed time	76.94	0.05	0.05
(seconds)	7.76:257.32	0.03:0.07	0.02:0.08
Problem 19 ($n = 300$):			
$f_{\min}$	1.71e+04	1.18e+03	1.18e+03
	2.44e+03:6.07e+04	1.18e+03:1.18e+03	1.18e+03:1.18e+03
f_{vals}	2.85e+03	1.52e+05	2.58e+05
	619:9343	105,640:200,115	227,962:284,554
Elapsed time	2.47	0.64	1.02
(seconds)	1.33:6.43	0.45:0.86	0.91:1.09

(*Continued*)

Table 5. Continued

	Range of values		
	PSD-NOMAD	Our algorithm ($q = 2$)	Our algorithm ($q = n/6$)
	Average min:max	Average min:max	Average min:max
Problem 21 ($n = 30$):			
$f_{\min}$	2.24e+09	2.24e+09	2.24e+09
	2.24e+09:2.28e+09	2.24e+09:2.25e+09	2.24e+09:2.25e+09
f_{vals}	7.11e+02	4.38e+03	4.13e+03
	445:1481	3515:5445	3495:4781
Elapsed time	122.42	0.19	0.18
(seconds)	1.47:4.45	0.15:0.24	0.15:0.21

Table 6. Comparison of PSD-NOMAD solution with our algorithm stopped prematurely at the same function value (100 runs for problems 1, 15, 16 and 19).

	Range of values	
	PSD-NOMAD Solution	Our algorithm ($q = 2$) Early stop
	Average min:max	Average min:max
Problem 1 ($n = 300$):		
$f_{\min}$	442.81	441.40
	2.21e+02:1.30e+03	4.37e+02:4.42e+02
f_{vals}	6709.1	6593.4
	702:11,865	6038:7139
Elapsed time	4.73	0.03
(seconds)	1.39:8.28	0.03:0.03
Problem 15 ($n = 30$):		
$f_{\min}$	0.036608	0.036254
	2.58e−02:4.79e−02	3.22e−02:4.57e−02
f_{vals}	5145.33	3487.78
	7.02e+02:1.19e+04	1.79e+03:1.32e+04
Elapsed time	60.43	0.01
(seconds)	8.56:151.47	0.01:0.05
Problem 16 ($n = 30$):		
$f_{\min}$	1.47260024	1.409270
	2.05e−04:14.62	1.07:1.47
f_{vals}	7566.32	4638.87
	2128:16,626	2655:10,063
Elapsed time	76.94	0.01
(seconds)	7.76:257.32	0.01:0.02
Problem 19 ($n = 300$):		
$f_{\min}$	17,115.8891	16,680.96
	2441.71:60,661.80	10,300.35:17,099.87
f_{vals}	2845.78	1780.18
	619:9343	1417:3530
Elapsed time	2.47	0.01
(seconds)	1.33:6.43	0.01:0.02

in $\mathbb{R}^2$. Let

$$\begin{aligned} \zeta_L^{i,j}(v) &= (v^{i+1,j} - v^{i,j})^2 + (v^{i,j+1} - v^{i,j})^2 \\ \zeta_U^{i,j}(v) &= (v^{i-1,j} - v^{i,j})^2 + (v^{i,j-1} - v^{i,j})^2 \end{aligned} \quad 1 \le i \le p, 1 \le j \le p,$$

and let

$$d^{i,j} = \frac{\min(i, p+1-i, j, p+1-j)}{p}, \quad 0 \le i \le p+1, 0 \le j \le p+1,$$

$$f(v) = \frac{1}{4}\sum_{i,j=1}^{p}[\zeta_L^{i,j}(v) + \zeta_U^{i,j}(v)] - \frac{25}{p^2}\sum_{i,j=1}^{p} v^{i,j}.$$

Formally, the problem to be solved is

$$\underset{v\in\mathbb{R}^{(p+2)\times(p+2)}}{\text{minimize}}\, f(v), \{v : 0 \le v^{i,j} \le d^{i,j},\ i \in \{0,\dots,p+1\},\ j \in \{0,\dots,p+1\}\},$$

but the bound constraints force $v^{i,j} = 0$ for $i \in \{0, p+1\}$, or $j \in \{0, p+1\}$, so we really deal with $p \times p$ variables. The starting point $v_1^{i,j} = \alpha_{i,j} d^{i,j}$, where $\alpha_{i,j}$ are randomly generated in the [0, 1] interval. For ease of interpretation, we define $k = (j-1) * p + i$ for both i and j in the set $\{1,\dots,p\}$, and work with variables $x \in \mathbb{R}^{p^2}$ and index sets $M_k = \{(k-1)p+1,\dots,kp\}$, $k = 1,\dots,p$. Our algorithms performed very well on experiments with $p = 32$, that is, 1024 variables, although in [12, Section 1.3] the authors claimed that it is usually not reasonable to use serial DF algorithms on problems with more than a few tens of variables. For the sake of completeness, we should mention that recent techniques can handle unconstrained problems in hundreds of variables [46].

Space decomposition was clearly superior for the run exemplified in the Table 7 when function evaluations are replaced by differences in function values (recall comments at the beginning of Section 3). The saving in computing time is bigger than 80% for the sequential processing. Parallel processing results with no space decomposition are attractive when function evaluations are not expensive and computing time is the prevailing issue. Note that processor failures occur, but due to the fault tolerance quality of our algorithm, this issue becomes irrelevant. Figure 4 is the solution obtained.

5.2.4 *Nonlinear constraints*

We use the multi-core architecture to solve the nonlinearly constrained problem suggested in [5]. This test was carried out with the aim of observing the performance of our algorithm on problems that are not BCOP. We treat the nonlinear constraints by the extreme barrier approach that forbids violations; that is, the SADSD algorithm described in Figure 2 is run without any modification. We

Table 7. Performance on torsion function, with and without space decomposition, $p = 32, n = p^2$ (function value and function evaluations).

	No space Decomposition		$M_k = \{(k-1)p+1,\dots,kp\}$ $k = 1,\dots,p$	
Process	$f(\cdot)$	f_{vals}	$f(\cdot)$	f_{vals}
Sequential	−4.05	85,937	−4.04	88,787
Parallel:				
Proc. 0	−4.05	56,897	FAIL[a]	
Proc. 1	FAIL[a]		−3.84	60,624
Proc. 2	−4.05	72,341	−4.05	100,178
Proc. 3	−4.05	71,345	−4.05	88,420

[a]Overfilled buffer.

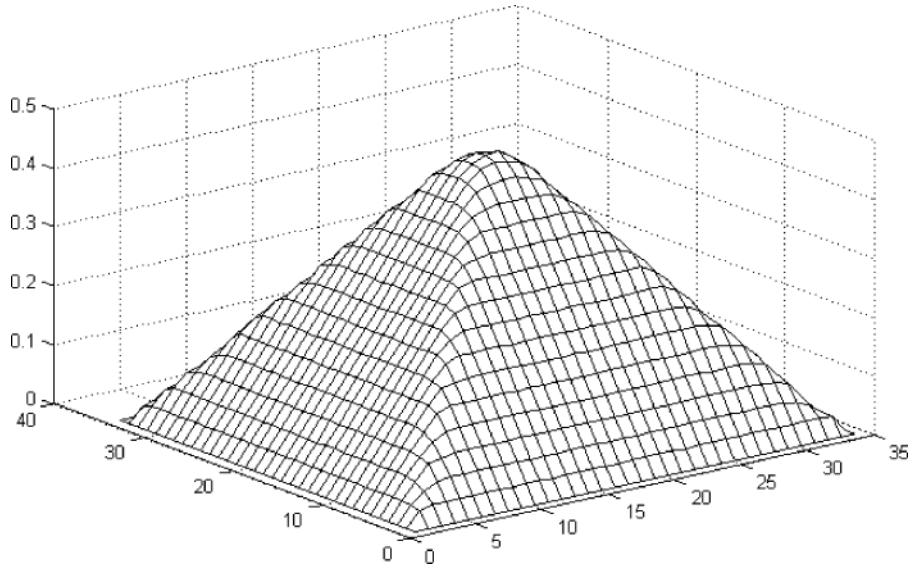

Figure 4. Solution to the torsion problem.

simply must take into account the new definition of the feasible set $\mathcal{F}$. We believe that convergence theory of our algorithm can be extended to general non-smooth constraints if we incorporate sets of directions asymptotically dense. We now state the problem and the parameters used for solving it. The initial input was $x_0 = (n\,0\ldots0)$, $\tau_0 = 1, \mu = 0.75, \epsilon = 10^{-10}$.

$$\min_{x\in\mathcal{F}} x^n, \mathcal{F} = \{x \in S : -n \le x^k \le n,\ k = 1,\ldots,n\},$$
$$S = \left\{x \in \mathbb{R}^n : \sum_{k=1}^{n}(x^k - 1)^2 \le n^2 \le \sum_{k=1}^{n}(x^k + 1)^2\right\} \tag{16}$$

Although this is not a BCOP, we tried a monotone version of our algorithms for solving it; i.e. $\varphi_i = f(x_i)\ \forall i$, which introduces an extra nuisance. It is not possible to find a sufficient decrease of $f(\cdot)$ when $n \notin M$.

We carried out four runs: the sequential algorithm, the parallel algorithm with no decomposition, or equivalently $M = N$, the parallel algorithm with $|M| = 4$ randomly generated and the parallel algorithm with $|M| = 4, n \in M$, and the other three elements randomly chosen. The latter strategy allows the possibility of sufficient decrease at all iterations.

The results given in Table 8 show that the worst result was obtained when M is randomly chosen and $n \in M$ is not guaranteed. The best result was obtained with parallelism and no decomposition, which again seems to confirm that VS is problem-dependent.

Table 8. Solution quality on problem (16).

Scenario	Solution (x_*^n)
1 thread. No decomposition	−8.951
4 threads. No decomposition	−8.998
4 threads. $\|M\| = 4$, random	−8.517
4 threads. $\|M\| = 4, n \in M$	−8.971

Note: True solution: $x^n = -9$.

6. Final remarks

This paper presents sequential and parallel NM algorithms for solving optimization problems with box constraints without the explicit use of derivatives. They are extensions of the NM DF algorithms proposed in [20,21], for solving unconstrained problems. Convergence results are proved for strictly differentiable functions that are sampled at a finite set of points at each iteration. It seems plausible to extend the convergence to the non-smooth case when an asymptotic set of search directions is used, as suggested by Audet and Dennis [4]. We also incorporate space decomposition techniques, an issue recently proposed in [6]. Our preliminary numerical results seem to imply that an efficient decomposition is strongly related to the structure and characteristics of the problem. Although it has been claimed that it is usually not reasonable to use serial DF algorithms for solving medium-sized problems [12], the algorithm solved a problem with 1024 variables and recent techniques are being proposed that can handle unconstrained problems in hundreds of variables [46].

The parallel algorithm proposed is a novel approach. It only considers communication of DQPs among processors, and all processors are able to sample function values in the whole space. This feature introduces a high degree of fault tolerance. The algorithm may theoretically work even in the extreme case when only one processor remains in operation.

We report preliminary numerical results and compare our algorithm with NOMAD, a state of the art algorithm recently proposed [2,6,33]. It is worth mentioning that the parallel version of our algorithm runs much faster than PSD-NOMAD. In several cases, the elapsed time was 1% of PSD-NOMAD's. Though these results are encouraging, more testing is desirable for higher dimensional problems and different multi-processor architectures. It has been pointed out that heuristic programming tools may be included in the implementation, which might improve its performance, specially in global optimization. The present research seems to imply that the proposed parallel algorithm is worth exploring in future studies.

Acknowledgements

The authors appreciate the efforts done by two anonymous referees, whose contributions enhanced the quality of this paper. They detected a flaw in a previous version, and pointed to us references [1,2,33]. We deeply acknowledge the careful handling of this paper by the associate editor. This search was supported by MICINN, Spain TEC 2010-21405-C02-01.

References

[1] M.A. Abramson, C. Audet, J.W. Chrissis, and J.G. Walston, *Mesh adaptive direct search algorithms for mixed variable optimization*, Optim. Lett. 3(1) (2007), pp. 35–47.
[2] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., and S. Le Digabel, *The NOMAD project*. Available at www.gerad.ca/nomad.
[3] C. Audet and J.E. Dennis, Jr., *Analysis of generalized pattern searches*, SIAM J. Optim. 13 (2003), pp. 889–903.
[4] C. Audet and J.E. Dennis, Jr., *Mesh adaptive direct search algorithms for constrained optimization*, SIAM J. Optim. 17(1) (2006), pp. 188–217.
[5] C. Audet and J.E. Dennis, Jr., *A MADS algorithm with a progressive barrier for derivative-free nonlinear programming*, SIAM J. Optim. 20(1) (2009), pp. 445–472.
[6] C. Audet, J.E. Dennis, Jr., and S. Le Digabel, *Parallel space decomposition of the mesh adaptive direct search algorithm*, SIAM J. Optim. 19 (2008), pp. 1150–1170.
[7] B.M. Averick, R.G. Carter, and J.J Moré, *The minpack-2 test problem collection*, TM 150, Mathematics and Computer Science Division, Argonne National Laboratory, IL, USA (1991).
[8] R.P. Brent, *Algorithms for Minimization Without Derivatives*, Prentice Hall, Englewood Cliffs, NJ, 1973, ISBN 0-13-022335-2.
[9] R.L. Cariño, I. Banicescu, and W. Gao, *Dynamic load balancing with matlabmpi*, in *6th International Conference on Computational Science*, Lecture Notes in Computer Science, Vol. 3992, Springer, Reading, UK, 2006, pp. 430–437.
[10] J. Céa, *Optimisation: Théorie et Algorithms*, Dunod, Paris/Springer Verlag, West Germany, 1971.

[11] A.R. Conn, K. Scheinberg, and P.L. Toint, *Recent progress in unconstrained nonlinear optimization without derivatives*, Math. Program. 79 (1997), pp. 397–414.
[12] A.R. Conn, K. Scheinberg, and L.N. Vicente, *Introduction to Derivative-Free Optimization*, MPS-SIAM series on optimization, SIAM, Philadelphia, PA, USA, ISBN 978-0-898716-68-9 edition, 2009.
[13] I.D. Coope and C.J. Price, *Frame based methods for unconstrained optimization*, Optim. Theory Appl. 107(2) (2000), pp. 261–274.
[14] I.D. Coope and C.J. Price, *Positive basis in numerical optimization*, Comput. Optim. Appl. 21(2) (2002), pp. 169–175.
[15] C. Davis, *Theory of positive linear dependence*, Amer. J. Math. 76 (1954), pp. 733–746.
[16] N.Y. Deng, Y. Xiao, and F.J. Zhou, *Nonmonotonic trust region algorithm*, J. Optim. Theory Appl. 76(2) (1993), pp. 259–285.
[17] M.A. Diniz-Ehrhardt, J.M. Martínez, and M. Raydán, *A derivative-free nonmonotone line-search technique for unconstrained optimization*, J. Comput. Appl. Math. 219 (2008), pp. 383–397.
[18] U.M. García Palomares, *Análisis y teorema de convergencia de un algoritmo de minimización sin el cálculo de derivadas*, Acta Cient. Venezolana 27 (1976), pp. 187–189.
[19] U.M. García Palomares, *Non monotone algorithms for unconstrained minimization: Upper bounds on function values*, in *Proceedings of the 22nd IFIP TC7 Conference*, Torino, Italy, F. Ceragioli, *et al*., eds., Springer, Berlin, Heidelberg, 2006, pp. 91–100, ISBN 0-387-32774-6.
[20] U.M. García Palomares and J.F. Rodríguez, *New sequential and parallel derivative-free algorithms for unconstrained minimization*, SIAM J. Optim. 13(1) (2002), pp. 79–96.
[21] U.M. García Palomares, F.J. González Castaño, and J.C. Burguillo Rial, *A combined global & local search (CGLS) approach to global optimization*, J. Global Optim. 34 (2006), pp. 409–426.
[22] F.J. González Castaño, E. Costa Montenegro, J.C. Burguillo Rial, and U.M. García Palomares, *Outdoor WLAN planning via non-monotone derivative free optimization: Algorithm adaptation and case study*, Comput. Optim. Appl. 40(3) (2008), pp. 405–419, ISSN 0926-6003.
[23] G.A. Gray and T.G. Kolda, *Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization*, ACM Trans. Math. Software 32(3) (2006), pp. 485–507.
[24] J.D. Griffin, T.G. Kolda, and R. Michael Lewis, *Asynchronous parallel generating set search for linearly-constrained optimization*, SIAM J. Sci. Comput. 30(4) (2008), pp. 1892–1924.
[25] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal. 23(4) (1986), pp. 707–716.
[26] L. Grippo and M. Sciandrone, *Nonmonotone globalization techniques for the Barzilai-Borwein gradient method*, Comput. Optim. Appl. 23 (2002), pp. 143–169.
[27] A.-R. Hedar and M. Fukushima, *Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization*, Optim. Methods Softw. 19 (2004), pp. 291–308.
[28] P.D. Hough, T.G. Kolda, and V.J. Torczon, *Asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Sci. Comput. 23 (2001), pp. 859–869.
[29] T.G. Kolda, *Revisiting asynchronous parallel pattern search for nonlinear optimization*, SIAM J. Optim. 16(2) (2005), pp. 563–586.
[30] T.G. Kolda and V.J. Torczon, *On the convergence of asynchronous parallel pattern search*, SIAM J. Optim. 14(4) (2004), pp. 939–964.
[31] T.G. Kolda, R. Michael Lewis, and V.J. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Rev. 45(3) (2003), pp. 385–482.
[32] T.G. Kolda, R. Michael Lewis, and V.J. Torczon, *Stationarity results for generating set search for linearly constrained optimization*, SIAM J. Optim. 17(4) (2006), pp. 943–968.
[33] S. Le Digabel, *Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm*, ACM Trans. Math. Software 37(4) (2011), pp. 44:1–44:15.
[34] S. Le Digabel, *Nomad User Guide*, Version 3.5.0, 2011. Available at: http://www.gerad.ca/NOMAD/Downloads/user_guide.pdf.
[35] S. Leyffer, *Integrating SQP and branch and bound for mixed integer nonlinear programming*, Comput. Optim. Appl. 18(3) (2001), pp. 295–309.
[36] Y. Li and Q.-H. Zhou, *A modified derivative-free algorithm for bound constrained optimization*, Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, China, 2006, pp. 2242–2245.
[37] S. Lucidi and M. Sciandrone, *On the global convergence of derivative-free methods for unconstrained optimization*, SIAM J. Optim. 13(1) (2002), pp. 97–116.
[38] S. Lucidi and M. Sciandrone, *A derivative-free algorithm for bound constrained optimization*, Comput. Optim. Appl. 21(2) (2002), pp. 119–142.
[39] R. Michael Lewis and V.J. Torczon, *Pattern search methods for bound constrained minimization*, SIAM J. Optim. 9 (1999), pp. 1081–1099.
[40] R. Michael Lewis and V.J. Torczon, *Pattern search methods for linearly constrained minimization*, SIAM J. Optim. 10 (2000), pp. 917–941.
[41] R. Michael Lewis and V.J. Torczon, Rank ordering and positive bases in pattern search algorithms, CRPC-TR96674, Rice University, Houston, USA, November 1996.
[42] J.J. Moré and S.M. Wild, *Benchmarking derivative-free optimization algorithms*, SIAM J. Optim. 20(1) (2009), pp. 172–191.
[43] J.J. Moré and S.M. Wild, *Benchmarking derivative-free optimization algorithms* (2009). Available at http://www.mcs.anl.gov/more/dfo/.

[44] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., Springer, New York, USA, 2006, ISBN 0387303030.
[45] *Parallel Programming with MatlabMPI*, Proceedings of the high performance embedded computing, September 2001. MIT Lincoln Laboratory. Available at www.ll.mit.edu/MatlabMPI.
[46] M.J.D. Powell, *Developments of NEWUOA for minimization without derivatives*, IMA J. Numer. Anal. 28(4) (2008), pp. 649–664.
[47] C.A. Sagastizábal and M.V. Solodov, *Parallel variable distribution for constrained optimization*, Comput. Optim. Appl. 22(1) (2002), pp. 111–131.
[48] P.L. Toint, *Non-monotone trust region algorithms for nonlinear optimization subject to convex constraints*, Math. Program. 77 (1997), pp. 69–94.
[49] V.J. Torczon, *Multi-directional search: A direct search algorithm for parallel machines*, Ph.D. thesis, Rice University, Houston, TX, 1989.