Comp. Appl. Math.
DOI 10.1007/s40314-017-0505-2

Parallelized hybrid optimization methods for nonsmooth problems using NOMAD and linesearch

G. Liuzzi[1] · **K. Truemper**[2]

Received: 18 October 2016 / Revised: 6 August 2017 / Accepted: 28 August 2017

Abstract Two parallelized hybrid methods are presented for single-function optimization problems with side constraints. The optimization problems are difficult not only due to possible existence of local minima and nonsmoothness of functions, but also due to the fact that objective function and constraint values for a solution vector can only be obtained by querying a black box whose execution requires considerable computational effort. Examples are optimization problems in Engineering where objective function and constraint values are computed via complex simulation programs, and where local minima exist and smoothness of functions is not assured. The hybrid methods consist of the well-known method NOMAD and two new methods called DENCON and DENPAR that are based on the linesearch scheme CS-DFN. The hybrid methods compute for each query a set of solution vectors that are evaluated in parallel. The hybrid methods have been tested on a set of difficult optimization problems produced by a certain seeding scheme for multiobjective optimization. We compare computational results with solution by NOMAD, DENCON, and DENPAR as stand-alone methods. It turns out that among the stand-alone methods, NOMAD is significantly better than DENCON and DENPAR. However, the hybrid methods are definitely better than NOMAD.

Keywords Derivative-free optimization · Nonsmooth optimization · Parallel algorithms

Mathematics Subject Classification 90C30 · 90C56 · 49J52 · 68W10

Communicated by Ernesto G. Birgin.

✉ G. Liuzzi
giampaolo.liuzzi@iasi.cnr.it

K. Truemper
klaus@utdallas.edu

[1] Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", Consiglio Nazionale delle Ricerche, Rome, Italy

[2] University of Texas at Dallas, Dallas, TX, USA

Published online: 08 September 2017

1 Introduction

This paper describes hybrid methods combining mesh search of NOMAD (Abramson et al. 2014; Le Digabel 2011) with linesearch based on CS-DFN (Fasano et al. 2014) for the following optimization problem.

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & l \leq x \leq u \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$, and $l, u \in \mathbb{R}^n$ such that $-\infty < l_i < u_i < +\infty$. Let us denote

$$X = \{x \in \mathbb{R}^n : \ l \leq x \leq u\}.$$

It is assumed that the objective $f(x)$ and the constraints $g_i(x)$ are not explicitly available and possibly are nonsmooth. For evaluation of these functions, a black box is provided that for any input vector x supplies their values. It is further assumed that the run time of the black box for each query is large.

Given Problem (1), we introduce the penalty function

$$Z_\epsilon(x) = f(x) + \frac{1}{\epsilon} \sum_{i=1}^{m} \max\{0, g_i(x)\}$$

and, thus, consider the following problem with upper and lower bounds

$$\begin{array}{ll} \min & Z_\epsilon(x) \\ \text{s.t.} & l \leq x \leq u. \end{array} \tag{2}$$

In Fasano et al. (2014), it has been shown that a threshold value $\bar{\epsilon} > 0$ exists such that, for all $\epsilon \in (0, \bar{\epsilon}]$, Problem (2) is equivalent to Problem (1).

In that setting, solution methods are attractive that in each query ask for function values of several, say k, vectors, which are then computed in parallel by k black boxes running on k processors. The proposed hybrid methods are of that type. We begin with a brief review of related prior work.

2 Brief review

Optimization methods that do not use derivatives or finite differences approximations are generally defined as Direct Search (DS) methods, since the optimization process is guided only by function values or *zero-order* information of the functions.

A substantial portion of DS methods are heuristics, that is algorithms that, though possibly quite efficient, are not supported by a convergence theory.

Well-known heuristics are the nature-inspired genetic algorithms (GA) (Goldberg 1989). GA methods have attracted the interest of researchers for many decades mainly for three reasons: they are easy to implement, often produce good results, and are easily and quite efficiently parallelized. The third reason is intimately connected with the main characteristic of GA methods, which is evolution of a population of individuals toward the most fitted ones.

We turn to DS methods with convergence proof. Many attempts have been made over the past two decades to construct efficient parallel versions. The first attempt in this direction is found in Dennis and Torczon (1991), where a general approach is described that produces

easily parallelized DS algorithms. The parallelization is done at a high level and in such a way that any number of processors can be utilized. Specifically, at any given iteration, the method looks ahead to subsequent iterations of the algorithm until a sufficient number of function evaluations have been generated that keep the available processors busy.

In García-Palomares and Rodríguez (2002), sequential and parallel DS methods are proposed that in each iteration search for a sufficiently large decrease of the objective function instead of a simple decrease. The number of processors that can be profitably exploited is limited to the number of variables. However, it is also argued that if one has many more processors than variables, it is possible to define processor sets for a parallel scheme that exhibits some degree of fault tolerance.

In Hough et al. (2001), the first attempt to parallelize a pattern search method is proposed. This approach has been successively pursued in Kolda and Torczon (2004), Kolda (2005), Gray and Kolda (2006), Meza et al. (2007) and extended to linearly constrained problems in Griffin et al. (2008). The method is initialized by choosing a set of directions that form a positive spanning set in $\mathbb{R}^n$. Each direction is assigned to at least one processor that is in charge of the search along that particular direction. The entire process is asynchronous, so coordination of the processors is not necessary. The parallelism is managed by broadcast messages which communicate progress achieved by one process to the other ones.

In Audet et al. (2008), a space decomposition technique is proposed within the framework of the mesh adaptive direct search method NOMAD Abramson et al. (2009), Audet and Dennis (2006), and Le Digabel (2011). The proposed method is an asynchronous parallel algorithm in which the processes solve problems over subsets of variables. Here, too, the number of processors that can efficiently be employed is bounded by the number of variables of the problem. NOMAD also offers a second parallelization option where a set of vectors is passed to the black box. The latter version is used in the hybrid methods proposed here.

In García-Palomares et al. (2013), the space decomposition approach is extended to sequential and parallel non-monotone algorithms, i.e., algorithms that do not enforce decrease of the objective function in each iteration (see, e.g., Grippo et al. 1986). The proposed parallel algorithm is structured in such a way that relevant information is exchanged between the processors, and each processor samples function values on each decomposed subspace.

3 Motivation

A certain seeding method for hard multiobjective minimization problems [27] requires solution of possibly very difficult instances of (1). Initially, we used NOMAD to solve these instances. While that method often performed quite well, at times it did not provide a satisfactory solution. This motivated the search for hybrid methods that use NOMAD together with other schemes. Since NOMAD uses mesh search, which is a grid-based approach, we conjectured that adding linesearch along certain directions should enhance performance. The method CS-DFN is a candidate for such linesearch. Unfortunately, CS-DFN evaluates one vector at a time and cannot carry out parallelized queries.

We thus were faced with two tasks: parallelizing the query process of CS-DFN and combining the resulting parallelized schemes with NOMAD. This paper reports the results of that effort.

Specifically, Sect. 4.1 through Sect. 4.3 provide a summary of CS-DFN followed by description of two parallelized versions of CS-DFN called DENCON and DENPAR. Method DENCON is structured in such a way that the trajectory of improving solution vectors is identical to that of CS-DFN. Hence, the convergence proof of Fasano et al. (2014) applies. In contrast, DENPAR generally produces different trajectories, and we sketch a proof of convergence that is based on that for CS-DFN.

Section 5 describes selection of the test problems used throughout. Sections 6 and 7 compare the performance of DENCON, DENPAR, and NOMAD on a single processor and their effectiveness on multiple processors. Section 8 combines NOMAD, DENCON, and DENPAR to two hybrid methods. Section 9 has computational results for these hybrid methods. Section 10 summarizes the results. Finally, an appendix contains tables that are too large and unwieldy to be incorporated into the text.

The parallelization ideas used for DENCON and DENPAR apply not just to CS-DFN, but generally can be utilized for any methods where, roughly speaking, (1) linesearch is used along sets of search directions and (2) candidate solution vectors along directions are determined according to a fixed pattern that does not depend on associated objective function and constraint values.

Instead of a general discussion of the two parallelization ideas, we introduce them while describing their implementation in CS-DFN Fasano et al. (2014). We review that method next.

4 The algorithms

For the sake of clarity, all of the algorithm will be presented assuming (instead of problem (2)) the following unconstrained problem

$$\min_{x \in \mathbb{R}^n} f(x), \tag{3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is assumed to be radially unbounded; that is, for every sequence $\{xk\}$ satisfying $\lim_{k\to\infty} \|x_k\| = +\infty$, $\lim f(x_k) = +\infty$.

Handling of the problem (2), where general nonlinear constraints are tackled by an exact penalty and bound constraints are present, requires more details and technicisms thus making the presentation considerably more cumbersome. Instead, by considering problem (3), we can considerably simplify the presentation of the various methods thus allowing the reader to concentrate on the main aspects which are connected with proposals for their parallelization.

4.1 CS-DFN

In this subsection, we briefly recall method CS-DFN from reference Fasano et al. (2014). CS-DFN uses linesearches along directions of direction sets. Define a *search pattern* to be a collection of points that lie on direction vectors emanating from a given solution. For a current solution and given direction set, a search pattern is selected. Using that pattern, a better solution is found, or it is determined that no improvement is possible. In the former case, a new search pattern is employed with the same direction set. In the latter case, convergence conditions are tested. If convergence cannot be claimed, a new direction set together with a new search pattern is selected, and optimization continues; otherwise the method stops. CS-DFN does not use derivatives, indeed allows the function $f(x)$ and $g_i(x)$ to be not smooth as assumed for (1).

Here is a summary of the method.

Algorithm 1: CS-DFN from Fasano et al. (2014)

Input: Problem (3).
Output: Solution of (3) meeting convergence criteria.
Choose $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, n$, a sequence $\{d_k \in \mathbb{R}^n : \|d_k\| = 1\}$, and set $k \leftarrow 0$

repeat
  Given d_k, form an orthonormal basis $D_k = \{d_k^1, \ldots, d_k^n\}$ of $\mathbb{R}^n$, where $d_k^1 = d_k$
  Set $y_k^1 \leftarrow x_k$
  for $i = 1, 2, \ldots, n$ **do**
    if $f(y_k^i + \tilde{\alpha}_k^i d_k^i) > f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ and $f(y_k^i - \tilde{\alpha}_k^i d_k^i) > f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ **then**
      Set $\tilde{\alpha} \leftarrow \alpha_k^i \leftarrow 0$, $\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2$, $d^+ \leftarrow d_k^i$
    else if $f(y_k^i + \tilde{\alpha}_k^i d_k^i) \leq f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ **then** Set $d^+ \leftarrow d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
    else Set $d^+ \leftarrow -d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
    end if
    if $\tilde{\alpha} > 0$ **then**
      Compute $\hat{\alpha} =$ `Expansion Step`$(\tilde{\alpha}, y_k^i, d^+)$ and set $\alpha_k^i \leftarrow \hat{\alpha}$, $\tilde{\alpha}_{k+1}^i \leftarrow \hat{\alpha}$
    end if
    Set $y_k^{i+1} \leftarrow y_k^i + \alpha_k^i d^+$
  end for
  Find $x_{k+1} \in \mathbb{R}^n$ s.t. $f(x_{k+1}) \leq f(y_k^{n+1})$ or set $x_{k+1} = y_k^{n+1}$, set $k = k + 1$.
until convergence

Algorithm 2: `Expansion Step`(α, y, d)

Input: $\alpha > 0$, $y, d \in \mathbb{R}^n$, $\gamma > 0$
Output: $\hat{\alpha} \geq \alpha$

```
Let β ← 2α
if f(y + βd⁺) > f(y) − γβ² then
  return α
end if
Set α ← β and go to step 1
```

For every k, an iteration of CS-DFN starts by forming the orthonormal basis $D_k = \{d_k^1, \ldots, d_k^n\}$ (where $d_k^1 = d_k$) for exploration in the neighborhood of x_k. For this purpose, the inner for-cycle is initialized by setting $y_k^1 = x_k$. Then, for each direction $d_k^i \in D_k$, $i = 1, \ldots, n$, sufficient improvement is sought, with respect to the current best point y_k^i, along the current direction. For this purpose, the points $y_k + \tilde{\alpha}_k^i d_k^i$ and $y_k - \tilde{\alpha}_k^i d_k^i$ are used to determine if sufficient improvement is attainable along d_k^i or $-d_k^i$, respectively. If no such improvement can be obtained, the tentative step size is reduced and the search proceeds to the next direction. If, on the other hand, sufficient improvement is attainable, then, through use of the `Expansion Step` procedure, the tentative step size is (possibly) augmented and the current best point is updated producing $y_k^{i+1} = y_k^i + \alpha_k^i d^+$.

Hence, every iteration of the for-loop performs a twofold task. On the one hand, it decides whether or not a given direction (or its opposite) is a good descent direction (on the basis of the current tentative step), in the sense that the improvement is above the threshold $\gamma(\tilde{\alpha}_k^i)^2$.

On the other hand, given a good descent direction, it computes a sufficiently large step size along such a direction, thus yielding sufficient improvement of the objective function.

Roughly speaking, we can summarize the behavior of CS-DFN as follows. Imagine a current solution point in n-dimensional space (i.e., y_k^i). Emanating from that solution are lines representing the current (yet unexplored) search directions (i.e., d_k^j, $j = i, \ldots, n$). The points of the current search pattern (i.e., those tentative points considered by the algorithm and by the `Expansion Step` procedure) divide the lines into line segments. We evaluate the points closest to the current solution, which we call the *neighborhood points*, to decide which lines provide an improvement over the current solution. The directions and lines producing such improvement are *good directions* and *good lines*. Starting from the current solution and moving along any such line, we stop at the first search pattern point for which the immediate successor point has an inferior value. That stopping point is the *good point* of the good line.

4.2 DENCON

The first method derived from CS-DFN, called DENCON, a simple modification of CS-DFN, is as follows. When a set of directions is given, parallel evaluation is used to decide whether any of the directions produces an improvement over the current solution. If an improvement is found, the method explores the search pattern involving that direction in parallel. If no direction produces an improvement, a new direction set is selected. The modifications are done in such a way that DENCON produces the same sequences $\{x_k\}$ and $\{y_k^i\}$, $i = 1, \ldots, n+1$ as the original method CS-DFN. Hence, convergence of DENCON trivially follows from that of CS-DFN.

To better understand DENCON, given D_k and an index $i \in \{1, \ldots, n\}$, let us introduce

$$U_k^i = \{d_k^i, d_k^{i+1}, \ldots, d_k^n\}.$$

Then, we recall that for any iteration k and index $i \in \{1, \ldots, n\}$, at the beginning of the ith iteration of the inner for-loop of Algorithm CS-DFN, we have that:

- the set $\bar{D}_k = D_k \backslash U_k^i$ contains all of the already explored directions, with $\bar{D}_k = \emptyset$ if no direction has already been explored, i.e., we are at the very beginning of the for-loop in CS-DFN, i.e., $i = 1$;
- the set $U_k^i \subseteq D_k$ contains all of the directions that have not yet been explored.

Obviously, we have $\bar{D}_k \cap U_k^i = \emptyset$ and $D_k = \bar{D}_k \cup U_k^i$. Given a current point $y_k^{\bar{\imath}}$ with $\bar{\imath} \in \{1, \ldots, n\}$, the set of unexplored directions $U_k^{\bar{\imath}}$, and the tentative step sizes $\tilde{\alpha}_k^i$, for all i such that $d_k^i \in U_k^{\bar{\imath}}$, the following set of points are considered

$$\mathcal{M}(y_k^{\bar{\imath}}, U_k^{\bar{\imath}}, \tilde{\alpha}_k) = \{z \in \mathbb{R}^n : z = y_k^{\bar{\imath}} \pm \tilde{\alpha}_k^j d_k^j, \ \forall\, d_k^j \in U_k^{\bar{\imath}}\}.$$

Thus, $|\mathcal{M}(y_k^{\bar{\imath}}, U_k^{\bar{\imath}}, \tilde{\alpha}_k)| = 2|U_k^{\bar{\imath}}|$. Algorithm CS-DFN explores points $z \in \mathcal{M}(y_k^{\bar{\imath}}, U_k^{\bar{\imath}}, \tilde{\alpha}_k)$ trying to determine the smallest integer $i \in \{\bar{\imath}, \ldots, n\}$ such that a sufficient decrease is attained at one of the points $y_k^{\bar{\imath}} + \tilde{\alpha}_k^i d_k^i$ or $y_k^{\bar{\imath}} - \tilde{\alpha}_k^i d_k^i$. If such an index does not exist, meaning that no direction in $U_k^{\bar{\imath}}$ is a descent direction with the current step sizes, all the $\tilde{\alpha}_k^i$, $\bar{\imath} \le i \le n$ are reduced by a constant factor, i.e., $\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2$. If, on the contrary, an index $\bar{\jmath}$ exists such that either

$$f(y_k^{\bar{\imath}} + \tilde{\alpha}_k^{\bar{\jmath}} d_k^{\bar{\jmath}}) < f(y_k^{\bar{\imath}}) - \gamma(\tilde{\alpha}_k^{\bar{\jmath}})^2, \tag{4}$$

or

$$f(y_k^{\bar{\imath}} - \tilde{\alpha}_k^{\bar{\jmath}} d_k^{\bar{\jmath}}) < f(y_k^{\bar{\imath}}) - \gamma(\tilde{\alpha}_k^{\bar{\jmath}})^2, \tag{5}$$

then

$$y_k^i = y_k^{\bar{\imath}}, \quad \text{for all } \bar{\imath} \le i \le \bar{\jmath}, \quad \text{and}$$
$$\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2, \quad \text{for all } \bar{\imath} \le i < \bar{\jmath}.$$

Furthermore,

(a) if (4) holds, then direction $d_k^{\bar{\jmath}}$ is further explored;
(b) if (5) holds, then direction $-d_k^{\bar{\jmath}}$ is further explored.

Hence, let us denote

$$d^+ = \begin{cases} d_k^{\bar{\jmath}} & \text{if (4) holds} \\ -d_k^{\bar{\jmath}} & \text{if (5) holds,} \end{cases}$$

and define the set

$$\mathcal{E}(y_k^{\bar{\imath}}, d^+, \tilde{\alpha}_k) = \{z \in \mathbb{R}^n : z = y_k^{\bar{\imath}} + \beta_k d^+, \beta_k = 2^h \tilde{\alpha}_k^{\bar{\jmath}}, h = 0, 1, \ldots\},$$

which contains all the points possibly explored by the `Expansion Step` procedure of Algorithm CS-DFN.

Note that all points possibly considered during the execution of every iteration are computed in advance and evaluated in parallel before the algorithm actually needs them. Now the set $\mathcal{E}(y, d, \alpha)$ is an infinite set of points. Thus, it is not possible to evaluate all the points of $\mathcal{E}$ in advance. Rather, we only consider points $y + 2^h \alpha d$, for $h = 0, 1, \ldots, \bar{h}$, with $\bar{h} = 32$. Let n_p denote the number of available processors. Then the sets $\mathcal{M}$ and $\mathcal{E}$ are processed by evaluating n_p points at a time. Obviously, there is no need for the algorithm to wait for evaluation of all possible points, since only a few of them are needed to perform the necessary computations. Hence, as soon as the first n_p points have been computed, the algorithm proceeds. If more points need to be evaluated, then a new set of n_p points are evaluated in parallel.

Below, we summarize DENCON.

Algorithm 3: DENCON

Input: Problem (3).
Output: Solution of (3) meeting convergence criteria.
Choose $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, n$, a sequence $\{d_k \in \mathbb{R}^n : \|d_k\| = 1\}$, and set $k \leftarrow 0$

1: **repeat**
2: Given d_k, form an orthonormal basis $\{d_k^1, \ldots, d_k^n\}$ of $\mathbb{R}^n$, where $d_k^1 = d_k$
3: Set $y_k^1 \leftarrow x_k$, $U_k^1 \leftarrow D_k$ and evaluate in parallel points in $\mathcal{M}(y_k^1, U_k^1, \tilde{\alpha}_k)$
4: **for** $i = 1, 2, \ldots, n$ **do**
5: **if** $f(y_k^i + \tilde{\alpha}_k^i d_k^i) > f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ and $f(y_k^i - \tilde{\alpha}_k^i d_k^i) > f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ **then**
6: Set $\tilde{\alpha} \leftarrow \alpha_k^i \leftarrow 0$, $\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2$, $d^+ \leftarrow d_k^i$
7: **else if** $f(y_k^i + \tilde{\alpha}_k^i d_k^i) \le f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$ **then** Set $d^+ \leftarrow d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
8: **else** Set $d^+ \leftarrow -d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
9: **end if**
10: Set $U_k^{i+1} \leftarrow U_k^i \backslash \{d_k^i\}$
11: **if** $\tilde{\alpha} > 0$ **then**
12: Evaluate in parallel points in $\mathcal{E}(y_k^i, d^+, \tilde{\alpha})$ and compute $\hat{\alpha}$ = `Expansion Step`$(\tilde{\alpha}, y_k^i, d^+)$
13: Set $\alpha_k^i \leftarrow \hat{\alpha}$, $\tilde{\alpha}_{k+1}^i \leftarrow \hat{\alpha}$, $y_k^{i+1} \leftarrow y_k^i + \alpha_k^i d^+$

14: Evaluate in parallel points in $\mathcal{M}(y_k^{i+1}, U_k^{i+1}, \tilde{\alpha}_k)$
15: **else** Set $y_k^{i+1} \leftarrow y_k^i$
16: **end if**
17: **end for**
18: Find $x_{k+1} \in \mathbb{R}^n$ s.t. $f(x_{k+1}) \leq f(y_k^{n+1})$ or set $x_{k+1} \leftarrow y_k^{n+1}$, set $k = k + 1$.
19: **until** convergence

Note the parallel evaluations in steps 3, 12, and 14. In particular, set $\mathcal{M}$ is used to carry out a parallel evaluation of points whenever the algorithm has to determine a direction, among those in U_k^i, with $1 \leq i \leq n$, of sufficient decrease; set $\mathcal{E}$, on the other hand, is used to carry out a parallel evaluation of points whenever the algorithm has already identified a direction d^+ of good descent but still has to determine the step size along that direction through use of the `Expansion Step` procedure.

4.3 DENPAR

The second method, called DENPAR, is more complicated. Here, a search pattern based on all directions of a given direction set is evaluated in parallel for improved solutions, which are then combined to a convex combination that becomes the next solution point.

Algorithm 4: DENPAR

Input: Problem (3).
Output: Solution of (3) meeting convergence criteria.
Choose $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, n$, a sequence $\{d_k \in \mathbb{R}^n : \|d_k\| = 1\}$, and set $k \leftarrow 0$
1: **repeat**
2: Given d_k, form an orthonormal basis $D_k = \{d_k^1, \ldots, d_k^n\}$ of $\mathbb{R}^n$, where $d_k^1 = d_k$
3: Evaluate in parallel points in $\mathcal{M}(x_k, D_k, \tilde{\alpha}_k) \cup_{i=1}^n \mathcal{E}(x_k, d_k^i, \tilde{\alpha}_k) \cup_{i=1}^n \mathcal{E}(x_k, -d_k^i, \tilde{\alpha}_k)$
4: **for** $i = 1, 2, \ldots, n$ **do**
5: **if** $f(x_k + \tilde{\alpha}_k^i d_k^i) > f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ and $f(x_k - \tilde{\alpha}_k^i d_k^i) > f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ **then**
6: Set $\tilde{\alpha} \leftarrow \alpha_k^i \leftarrow 0$, $\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2$, $\hat{d}_k^i \leftarrow d_k^i$
7: **else if** $f(x_k + \tilde{\alpha}_k^i d_k^i) \leq f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ **then** Set $\hat{d}_k^i \leftarrow d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
8: **else** Set $\hat{d}_k^i \leftarrow -d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
9: **end if**
10: **if** $\tilde{\alpha} > 0$ **then**
11: compute $\hat{\alpha} =$ `Expansion Step`$(\tilde{\alpha}, x_k, \hat{d}_k^i)$
12: Set $\alpha_k^i \leftarrow \hat{\alpha}$, $\tilde{\alpha}_{k+1}^i \leftarrow \hat{\alpha}$, $\tilde{d}_k^i \leftarrow d^+$
13: **end if**
14: **end for**
15: **if** $\sum_{i=1}^n \alpha_k^i > 0$ **then** (*Success step*)
16: Compute $x_c \leftarrow \frac{\sum_{i=1}^n (x_k + \alpha_k^i \hat{d}_k^i)\alpha_k^i}{\sum_{i=1}^n \alpha_k^i}$ (*convex combination*)
17: Produce new point $x_{k+1} \leftarrow x_c$
18: **else** (*Failure step*)
19: Set $x_{k+1} \leftarrow x_k$
20: **end if**
21: Set $k \leftarrow k + 1$.
22: **until** convergence

In general, the convex combination point computed at Line 16 of DENPAR is not guaranteed to be better than the current solution. Of course, we could evaluate the convex combination point by itself, and in case it proves to be inferior, select some other point among the good points on hand as the next current solution. Thus, the evaluation of the convex combination point would impose a further synchronization in algorithm DENPAR just before starting a new iteration. This would be inefficient in the parallel evaluation environment, and this is the reason why we choose to immediately accept the convex combination point and defer its evaluation to the beginning of the next iteration. Experiments have shown that for a large collection of test problems the convex combination virtually always beats the current solution. Hence, in our implementation we simply accept the convex combination point without additional testing.

The direct use of convex combinations at Line 17 of DENPAR may seem simplistic. Why not define a direction from the current point to the convex combination and then search for an improved solution along that direction? In tests, that idea proved to be ineffective. This is likely due to the fact that the convex combination is derived from good points of the search pattern.

Convergence of DENPAR can be proved if we demand quasi-convexity Ponstein (1967) of $f(x)$ and of the penalty terms $\max\{0, g_i(x)\}$ in addition to the assumptions made in the convergence proof of CS-DFN. Note that $\max\{0, g_i(x)\}$ is quasi-convex if this is so for $g_i(x)$.

We sketch a proof of convergence under these assumptions. The arguments are based on those for CS-DFN, which in turn rely on two key ideas that are well known for proofs of convergence to stationary points for derivative-free methods. We review these ideas of CS-DFN.

1. In Algorithm CS-DFN, the rule producing points of the search pattern is structured in such a way that the points gradually cluster ever closer to the current point as iterations proceed.
2. In Algorithm CS-DFN, the sequence of sets of search directions defining the search pattern is selected in such a way that, in the limit, the entire space is covered. Indeed, this property holds for the subsequence of sets of search directions related to a particular limit point of the sequence of iterates.

With these observations, we can list the key arguments for a convergence proof of DENPAR.

1. In Algorithm DENPAR, it can be shown (see Appendix B) that the selected sets of search directions have the same exploration property as those of CS-DFN, thus satisfying the condition of idea #2.
2. It can be shown (see Appendix B) that the rule used to define the points of the search pattern is the same as that of CS-DFN. Hence, clustering of the points belonging to the search pattern around the current iterate is obtained in the limit, as demanded by idea #1 above.
3. For Line 17 of DENPAR, it can be proved (see Appendix B) that, under the quasi-convexity assumption, the next point satisfies the condition of better solutions imposed at Line 15 of CS-DFN (i.e., that $f(x_{k+1}) \leq f(x_k)$).

5 Test problems

The earlier mentioned seeding method of [27] solves a multiobjective minimization problem with k objective functions by first defining and then solving k single-function minimization instances. Each of the latter instances is some linear combination of the k objective functions.

We use the seed subroutine defining the single-function minimization instances to create difficult test cases, as follows.

We start with five well-known functions where the number of variables n can be varied and where the difficulty increases substantially as n grows. One of the functions is maxl of Pillo et al. (1993), and the remaining four functions are nf3, rg, sin, and zkv of Laguna et al. (2009). From the computational results of the cited references, we know that these functions become difficult to minimize as n grows.

The domains and bounds on variables of the stated functions vary, but we revise them by scaling and translation so that the modified functions are defined on the same domain and their variables have the same bounds. Next, we pair up the five functions in all possible ways to obtain ten multiobjective minimization problems where minimization of each function is already difficult. For each of the ten multiobjective instances, we select three cases for the number of variables: $n = 10$, 20, and 30. This results in a total of 30 multiobjective problems. We run the seed subroutine of [27] to obtain 2 instances for each multiobjective instance, thus obtaining a total of 60 single-function minimization problems. The latter instances are used in all tests.

For the comparison of results produced by various algorithms, we rely on the procedure of Moré and Wild (2009), which is reviewed next following some definitions.

In the case of single-processor runs, define the *budget* for an algorithm to be a bound on the black box evaluations that may be used while processing a problem instance. In the case of multiple processors, let a *cycle* be one parallel evaluation of a set of vectors by black boxes running on the given processors. Thus, in a single cycle evaluation multiple function evaluations are performed. The *budget* is then a bound on the number of cycles that may be used.

6 Comparison of DENCON, DENPAR, and NOMAD

In a first use of the data profile and performance profile, we compare the performance of DENCON, DENPAR, and NOMAD when used as stand-alone methods on a single processors or several processors. For the latter case, we select 64 processors since it is a reasonable and widely available configuration.

The function value $f(x_0)$ of the initial point x_0 is crucial for deciding whether the results of a given algorithm satisfy the inequality

$$f(x_0) - f(x_a) \geq (1 - \tau)(f(x_0) - f_L) \tag{6}$$

where $f(x_a)$ is the best value reached by the given algorithm and f_L is the best value obtained by any algorithm on the given problem, and thus whether the algorithm has solved a problem. It seems reasonable to do so, since the improvement made by the best algorithm, which is $f(x_0) - f_L$, is compared with the improvement $f(x_0) - f(x_a)$ of algorithm a.

But for the difficult problems considered here, this viewpoint creates a difficulty. The reason is that a number of problems have feasible solutions x with huge function values, say on the order of 10^{10} and up, while the optimal function is comparatively small, say less than 10^3. Suppose the starting point x_0 happens to be a solution with huge function value. Then even a small tolerance such as $\tau = 10^{-5}$ implies that an algorithm achieving $f(x_a) \leq 10^5$ is declared to solve the problem, yet the result may be useless for the given application. The same difficulty arises from practical problems, for example in certain engineering applications where feasible outlier solutions may produce huge function values.

If we were dealing with just one problem, we could manually adjust the tolerance τ to eliminate this pitfall. But when this is done for each problem of a large test collection, the manual approach amounts to an undesirable manipulation of problem evaluation.

There is a consistent, simple remedy. From a preliminary investigation, we know that NOMAD processes the problem of the test collection while using at least 1400 and sometimes more than 30,000 evaluations. During the first 500 evaluations of any of these problems, NOMAD typically manages to compute a solution that is not outlandish as defined above. Thus, we apply NOMAD to each instance with a budget of 500 evaluations and get reasonable starting solutions. These solutions are used in all runs.

The reader may object to the use of NOMAD to get starting solutions. Why not use some other method? Now the hybrid methods we evaluate later always use NOMAD in the first step and would have produced the starting solution anyway. Thus, the starting solutions improve the usefulness of the performance and data profiles without artificial introduction of another method.

But the objection of the reader is well placed when we compare DENCON, DENPAR, and NOMAD as stand-alone methods as done in this section. Indeed, DENCON and DENPAR do not contain a reasonable subroutine for finding a starting solution; if no start solution is provided, they use the midpoint between the upper and lower bound for each variable to define the starting point. Thus, we should emphasize that we are comparing NOMAD with DENCON and DENPAR where the latter two methods are allowed to use start-up steps of NOMAD.

We discuss details of the runs. Both DENCON and DENPAR require selection of just one input parameter α that controls termination. We set it at a reasonable value 10^{-4}. In contrast, the most recent version of NOMAD, 3.6.2, used here offers a number of options. We select a configuration based on earlier computational experience with difficult engineering problems. Specifically, we use `LH_SEARCH 5 5`, `MIN_POLL_SIZE` 10^{-6}, `INITIAL_MESH_SIZE 0.125`, `MIN_MESH_SIZE` 10^{-6}, `VNS_SEARCH 0.75`, `SNAP_TO__BOUNDS 0`, and `DIRECTION_TYPE ORTHO 2N`. All other NOMAD parameters use the default values. For details about these parameters, see the NOMAD manual [1].

For each test problem, a budget of 50,000 evaluations is specified, not counting the 500 evaluations spent to obtain the starting solution with NOMAD. The data and performance profiles are constructed using two tolerances, $\tau = 10^{-3}$ and $\tau = 10^{-5}$. As concerns the construction of data and performance profiles for multi-processor runs, we remark that, for each problem and for each solver, the output file used to build the profiles is a 50,000 (i.e., the overall budget) line-long text file (as it can be seen at www.iasi.cnr.it/~liuzzi/hybridDF). Hence, there is exactly one line per function (for single-processor runs) or cycle (for multi-processor runs) evaluations. Since a cycle evaluation amounts to the computation in parallel of a set of function values, in the history file used to build data and performance profiles we output the best function value among those computed in the cycle evaluation. Furthermore, we specify that, when using 64 processors, the x-axis of data profiles represents the number of *parallel* simplex gradients evaluations κ, which is equal to 64 times the number of simplex gradients, i.e., $\kappa = 64 \cdot \nu$.

We truncate the graphs of the data profiles $d_a(\kappa)$ ($d_a(\nu)$) when using 64 processors (a single processor) for the three methods at the value κ (ν) where all three functions have reached their maximum under the given budget. In Moré and Wild (2009), it is argued that performance and data profiles give insights of the relative performances of the compared algorithms. Three aspects of the profiles are particularly relevant, namely

(i) for performance profiles, the values $\rho_a(0)$;
(ii) how steeply the curves rise;
(iii) how high the curves rise.

In particular, aspect (ii) helps evaluate the efficiency of the algorithms, and aspect (iii) the robustness. Then, for each of the truncated curves, we compute the area under the curve, arguing that the performance (or data) profile (of an algorithm) which climbs very rapidly and rises very high has a relatively large area under the curve. Further, we propose to compare methods on the basis of the area under their performance or data profiles. The methods are then compared using the percentage improvements where the smallest area is the base case at 100%. Thus, the method producing the smallest area has 0% improvement, and the other two methods have $\geq$0% improvement. The same truncation and display rules apply to the performance profiles $\rho_a(\alpha)$.

It turns out that the results produced by NOMAD are superior to, or at least match, those obtained with DENCON and DENPAR, regardless of the number of variables or the choice of single-processor versus 64 processors. The most impressive NOMAD performance is for the most difficult problems, with $n = 30$ variables, running on 64 processors. Figure 1 has graphs showing the profiles, where the top two graphs display the data profiles; the x-axis represents the number of parallel simplex gradients κ, and the y-axis the data profile $d_a(\kappa)$ for $\tau = 10^{-3}$ and $\tau = 10^{-5}$. The lower two graphs show the performance profiles, with the x-axis representing the values of α, and the y-axis the performance profile $\rho_a(\alpha)$.

The solid blue curve is produced by NOMAD, the heavy-dash black curve by DENCON, and the dot-dash red curve by DENPAR. With each graph, the enclosed box has the percentages indicating relations between areas under the curves. For example, for the top left graph, the worst data profile area is produced by DENCON and thus is assigned 0%. DENPAR improves upon that value by 17.36%, while NOMAD has a much larger improvement of 200.81%. The four graphs of Fig. 1 indicate that, no matter which profile and τ value is used, NOMAD is far superior to DENCON and DENPAR when the problems with $n = 30$ variables are solved on 64 processors. Concerning the poor behavior of DENCON with respect to NOMAD, we remark that the choice we made for the tolerance in the stopping condition of DENCON (i.e., 10^{-4} as opposed to the value 10^{-6} used for `MIN_POLL_SIZE` and `MIN_MESH_SIZE` in NOMAD) has no significant impact. Indeed, running DENCON with a tolerance of 10^{-6} essentially does not change the results. The poor behavior of DENCON when compared to the results of CS-DFN, as reported in Fasano et al. (2014), is more reasonably due to the different pseudo-random generator used by the two codes. In fact, in DENCON the Halton sequence Halton (1960) is used, whereas the published version of CS-DFN uses the more efficient Sobol sequence Sobol (1977), Bratley and Fox (1988).

Space limitations preclude the display of graphs for the cases where either all problems or those with $n = 10$ or 20 variables are solved, or where a single processor instead of 64 processors is used. All graphs are available at http://www.iasi.cnr.it/~liuzzi/hybridDF/. But a summarizing display is possible, in the form of Table 1.

It covers solution of all subsets of problems on a single processor as well as on 64 processors. The table is subdivided into two major parts: the top part for the single-processor runs, and the bottom part for 64-processor runs. Within each part, row subsections cover executions for the entire problem set as well as for subsets with $n = 10$, 20, or 30 variables. In each column of each row subsection, the bold entry is the maximum value of that column.

For example, in the bottom case of Table 1, which concerns runs with 64 processors for the problem subset with $n = 30$ variables, the columns for $\tau = 10^{-3}$ contain the percentages listed in the boxes of Fig. 1. NOMAD exhibits best performance for that case, with percentages

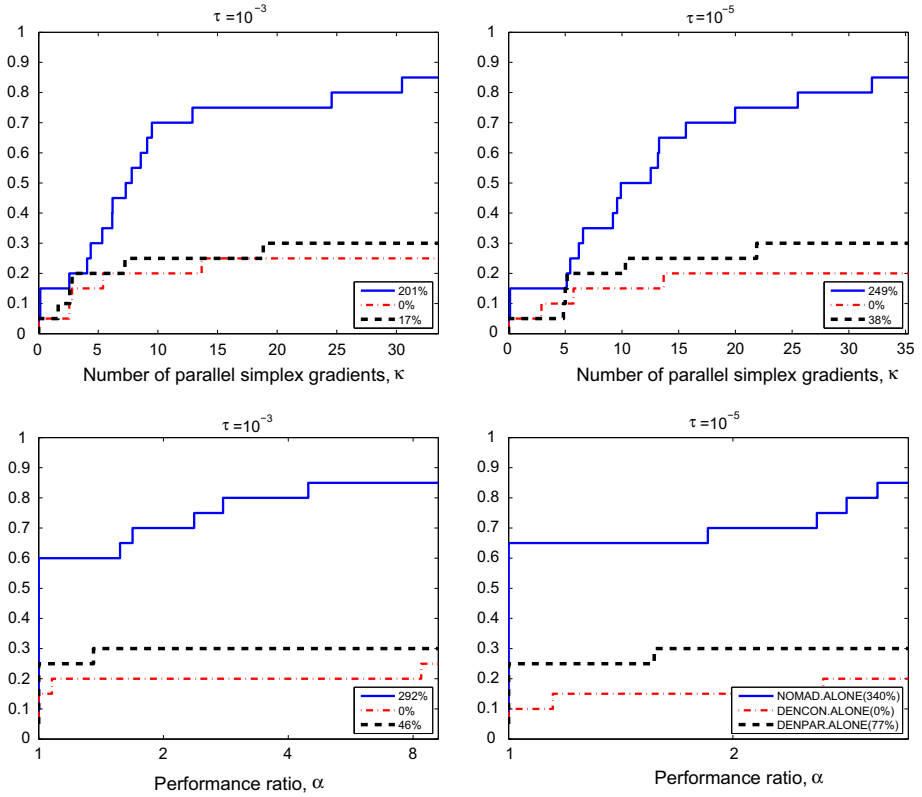

Fig. 1 Data (*top*) and performance (*bottom*) profiles for DENCON, DENPAR, and NOMAD for problems with $n = 30$ variables, running on 64 processors

ranging from 201 to 340%, as indicated in bold font in Table 1. All other bold percentages of Table 1 do not exceed 62%. Then for single processor execution we would choose NOMAD for $\tau = 10^{-3}$ and DENCON for $\tau = 10^{-5}$. For execution on 64 processors, we would choose NOMAD regardless of the τ value. Thus, Fig. 1 displays the strongest performance of NOMAD relative to DENCON and DENPAR.

We might use other selection rules based on the bold entries, but regardless of choice, the likely conclusion is that NOMAD dominates DENCON and DENPAR.

Given the dominance of NOMAD, one might have doubt that combining it with DENCON or DENPAR is useful. Before we check whether this conclusion is valid, let us look at the efficiency with which the three methods exploit parallel processors.

7 Effectiveness of parallelization for NOMAD, DENCON, and DENPAR

The analysis of effectiveness of the parallelization is complicated by the fact that for some problems and some methods, the computations are terminated due to the limit of 50,000 imposed on the total number of evaluations. It turns out that in the runs of the three methods, this limit is rarely reached during single-processor runs. But it is reached several times by both DENCON and DENPAR during runs using 64 processors. Details are included in Table 2.

Table 1 NOMAD, DENCON, and DENPAR comparison

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	**16**	**20**	4	21
DENCON	11	17	**10**	**30**
DENPAR	0	0	0	0
One processor, $n = 10$ variables				
NOMAD	0	0	0	11
DENCON	**27**	**52**	**25**	**30**
DENPAR	14	18	21	0
One processor, $n = 20$ variables				
NOMAD	**29**	14	3	25
DENCON	26	**21**	**15**	**48**
DENPAR	0	0	0	0
One processor, $n = 30$ variables				
NOMAD	**55**	**52**	**24**	**2**
DENCON	0	0	3	1
DENPAR	9	16	0	0
64 processors, all problems				
NOMAD	**48**	**60**	**41**	**59**
DENCON	0	0	0	4
DENPAR	1	0	0	0
64 processors, $n = 10$ variables				
NOMAD	0	11	0	4
DENCON	**22**	**29**	22	**28**
DENPAR	13	0	**30**	0
64 processors, $n = 20$ variables				
NOMAD	**50**	**62**	**39**	**41**
DENCON	1	0	13	0
DENPAR	0	13	0	5
64 processors, $n = 30$ variables				
NOMAD	**201**	**249**	**292**	**340**
DENCON	0	0	0	0
DENPAR	17	38	46	77

For example, for the case of DENCON and problems with $n = 30$ variables, 8 out of a total of 20 problems are solved within the limit of 50,000 evaluations, as shown in the column labeled "used." Thus, for $20 - 8 = 12$ problems, that limit triggers termination.

If the problems that terminated due to the evaluation limit are included in the analysis of effectiveness of parallelization, then biased cases would be admitted where the method converges on a single-processor run and fails to do so on 64 processors. Thus, we exclude these cases. A negative effect of this decision is that we cannot compare different methods. But on the positive side, we obtain a reasonable and unbiased comparison of parallelization

Table 2 NOMAD, DENCON, and DENPAR: efficiency for 64 processors

Vars.	Cases			Single proc.	Parallel proc.		Efficiency	
n	t total	u used	u/t (%)	s evals.	p evals.	c cycs.	$s/(64 \cdot c)$ (%)	$p/(64 \cdot c)$ (%)
NOMAD								
All	60	60	100	425, 000	411, 228	23, 722	28	27
10	20	20	100	40, 937	37, 972	4667	14	13
20	20	20	100	121, 678	137, 929	8801	22	24
30	20	20	100	262, 385	235, 327	10, 254	40	36
DENCON								
All	60	38	63	86, 520	441, 879	15, 155	9	46
10	20	19	95	33, 689	108, 447	5385	10	31
20	20	11	55	26, 713	161, 522	5563	8	45
30	20	8	40	26, 118	171, 910	4207	10	64
DENPAR								
All	60	44	73	161, 150	697, 529	11, 696	22	93
10	20	19	95	30, 000	204, 065	3436	14	93
20	20	15	75	62, 073	252, 914	4376	22	90
30	20	10	50	69, 077	240, 550	3884	28	97

effectiveness for each method. Table 2 has these results. We discuss the first line of the section devoted to NOMAD as example case. For that case, the number of variables n is not restricted, so $t = 60$ problems are available. Of these, $u = 60$ are solved within the limit of 50000 evaluations, resulting in a ratio $u/t = 100\%$. The runs on a single processor use a total of $s = 425{,}000$ evaluations. In contrast, the runs on 64 processors require a total of $p = 411{,}228$ evaluations carried out in $c = 23722$ cycles.

The table lists two efficiencies. The first one is the ratio $s/(64 \cdot c) = 28\%$, which measures not only how well the 64 processors are used, but also how convergence is affected by the parallelization. The second ratio is $p/(64 \cdot c) = 27\%$, which measures the effectiveness of the parallelization while disregarding changes in convergence and thus is of minor interest.

The first ratio, $s/(64 \cdot c)$, ranges for NOMAD from 14 to 40%, for DENCON from 8 to 10%, and for DENPAR from 14 to 28%. Thus, NOMAD is best in the use of the 64 processors. Combining this information with the results of Sect. 6, we see that NOMAD not only finds better solutions faster, but also makes more efficient use of 64 processors.

For NOMAD and the subcases of $n = 10, 20$, and 30, the ratio $s/(64 \cdot c)$ is 14, 22, and 40%, reflecting that NOMAD (version 3.6.2) can take advantage of at most $2n$ processors. Indeed, the poll steps of NOMAD (i.e., the steps which are essential to guarantee convergence) require the evaluation of (at most) $2n$ points. Then, one can ask NOMAD to output all these $2n$ points in one file for parallel execution. DENPAR shows similar behavior with ratios equal to 14, 22, and 28%. On the other hand, for DENCON the ratio is roughly constant, ranging from 8 to 10%. The behavior of the ratio for DENCON and DENPAR can be traced back to the way the parallelization is carried out when 64 processors are available. In DENCON, for up to 32 variables, one cycle is used to decide which direction to use, and a second cycle is needed to search for the good point of that direction. In DENPAR, all directions are evaluated in parallel. When the number of variables is small relative to the number of processors, that

parallel evaluation considers many candidate points beyond the good points of the directions, thus decreasing the effectiveness of the parallelization.

8 Hybrid methods

We only consider hybrid methods that iteratively execute sequences of NOMAD, DENCON, and DENPAR. We always compute a starting solution using NOMAD with a budget of 500 evaluations as described in Sect. 6. These evaluations are not counted in the overall budget.

The initial method in any hybrid sequence begins with that starting solution. Subsequently, each method receives the best solution obtained so far and tries to improve upon it while constrained by a certain budget in the number of function (for single-processor runs) or cycle (for multi-processor runs) evaluations called the *iteration budget*. Section 9 discusses selection of that budget. The method stops if the iteration budget is used up or the convergence condition of the method is satisfied. In either case, the method passes the best solution on hand to the next method.

The parameters assigned to each method are as defined in Sect. 6 except that in the kth iterative application of NOMAD, $k = 2, 3, \ldots$, the value 0.125 of `INITIAL_MESH_SIZE`

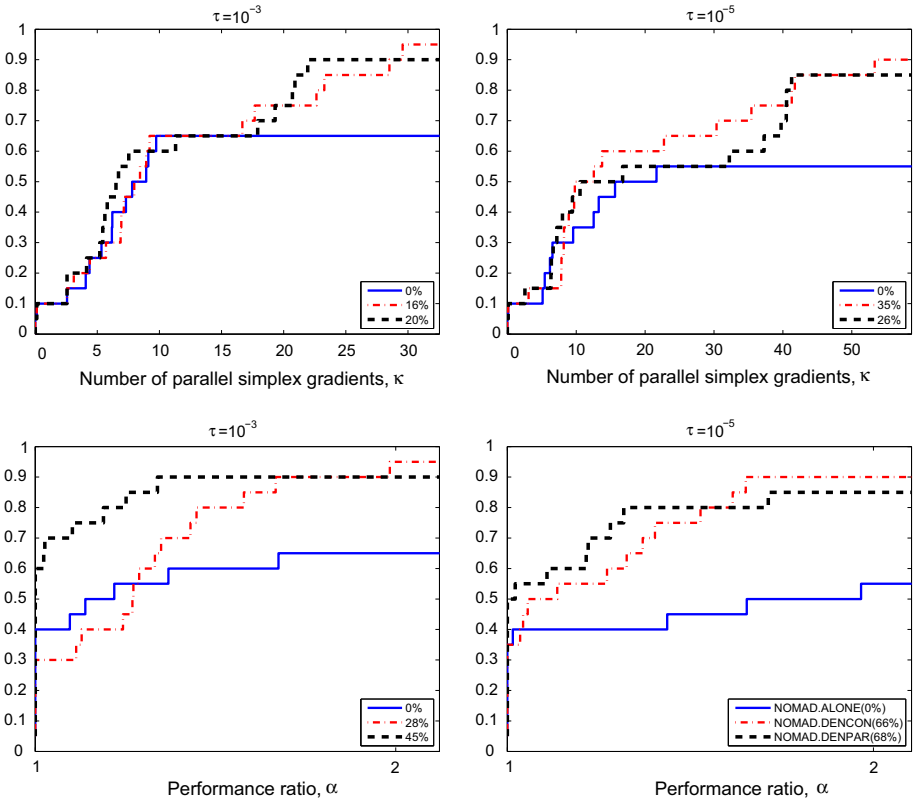

Fig. 2 Data (*top*) and performance (*bottom*) profiles for NOMAD, NOMAD–DENCON, and NOMAD–DENPAR on 64 processors, for problems with $n = 30$ variables, iter. budget = 2000

is reduced to $\max\{0.5^{k-1} \cdot 0.125, 100 \cdot \texttt{MIN\_MESH\_SIZE}\}$ to avoid big-step global searches that likely are fruitless. Here, k does not count the initial NOMAD application that finds a starting solution with a limit of 500 evaluations.

We only consider the sequences NOMAD–DENCON and NOMAD–DENPAR, each possibly iterated a number of times. The lead by NOMAD is motivated by the fact that NOMAD is more capable of searching out the entire solution space. The subsequent restriction to either DENCON or DENPAR is based on the fact that they are sufficiently similar to rule out their joint use in a sequence.

After each iteration through NOMAD–DENCON or NOMAD–DENPAR, we check whether the objective function has been reduced compared with the incumbent value. If

Table 3 NOMAD–DENCON efficiency for 64 processors

Vars.	Cases			Single proc.	Parallel proc.		Efficiency	
n	t total	u used	u/t (%)	s evals.	p evals.	c cycs.	$s/(64 \cdot c)$ (%)	$p/(64 \cdot c)$ (%)
Iteration budget = 500								
All	60	60	100	56,3365	58,533	34,966	25	26
10	20	20	100	85,645	133,328	12,358	11	17
20	20	20	100	174,575	184,977	10,096	27	29
30	20	20	100	303,145	264,228	12,512	38	33
Iteration budget = 1000								
All	60	60	100	563,365	798,283	43,377	20	29
10	20	20	100	85,645	168,148	14,679	9	18
20	20	20	100	174,575	283,096	14,358	19	31
30	20	20	100	303,145	347,039	14,340	33	38
Iteration budget = 2000								
All	60	60	100	600,504	1,111,756	53,019	18	33
10	20	20	100	94,071	187,726	14,762	10	20
20	20	20	100	172,274	377,125	18,088	15	33
30	20	20	100	334,159	546,905	20,169	26	42
Iteration budget = 3000								
All	60	60	100	645,439	1,174,454	53,171	19	35
10	20	20	100	104,492	186,117	14,199	11	20
20	20	20	100	205,396	390,614	18,302	18	33
30	20	20	100	335,551	597,723	20,670	25	45
Iteration budget = 4000								
All	60	60	100	662,459	1,178,341	52,766	20	35
10	20	20	100	108,435	197,696	14,585	12	21
20	20	20	100	216,063	401,562	18,486	18	34
30	20	20	100	337,961	579,083	19,695	27	46
Iteration budget = 5000								
All	60	59	98	694,899	1,158,894	50,767	21	36
10	20	20	100	106,742	185,751	13,759	12	21
20	20	20	100	243,675	419,199	18,657	20	35
30	20	19	95	344,482	553,944	18,351	29	47

the reduction is less than or equal to 10^{-6}, the computations are stopped. But we also limit the number of iterations directly to prevent excessive computations when that convergence condition is never satisfied. The iteration limit is computed as 50,000/(*iteration budget*).

9 Computational results

Before we begin the detailed comparison, let us look at a particular case where we compare NOMAD as stand-alone method with the two hybrid methods NOMAD–DENCON and NOMAD–DENPAR using an iteration budget of 2000 evaluations. In Appendix A, we shall

Table 4 NOMAD–DENPAR efficiency for 64 processors

Vars.	Cases			Single proc.	Parallel proc.		Efficiency	
n	t total	u used	u/t (%)	s evals.	p evals.	c cycs.	$s/(64 \cdot c)$ (%)	$p/(64 \cdot c)$ (%)
Iteration budget = 500								
All	60	60	100	563,865	644,179	32,342	27	31
10	20	20	100	85,407	139,036	10,066	13	22
20	20	20	100	189v905	213,312	9612	31	35
30	20	20	100	288,553	291,831	12,664	36	36
Iteration budget = 1000								
All	60	59	98	550,048	812,672	37,460	23	34
10	20	20	100	85,407	175,729	11,965	11	23
20	20	20	100	189,905	289,076	12,456	24	36
30	20	19	95	274,736	347,867	13,039	33	42
Iteration budget = 2000								
All	60	60	100	629,919	1,144,981	47,922	21	37
10	20	20	100	103,344	194,388	12,705	13	24
20	20	20	100	201,387	414,745	17,127	18	38
30	20	20	100	325,188	535,848	18,090	28	46
Iteration budget = 3000								
All	60	60	100	681,429	1,167,667	45,987	23	40
10	20	20	100	92,573	178,597	11,843	12	24
20	20	20	100	222,466	395,495	15,806	22	39
30	20	20	100	366,390	593,575	18,338	31	51
Iteration budget = 4000								
All	60	60	100	683,641	1,135,223	43,979	24	40
10	20	20	100	93,829	174,622	11,272	13	24
20	20	20	100	216,327	384,006	15,232	22	39
30	20	20	100	373,485	576,595	17,475	33	52
Iteration budget = 5000								
All	60	59	98	678,716	1,105,983	42,630	25	41
10	20	20	100	92,739	169,908	10,920	13	24
20	20	20	100	21,1230	397,199	15,824	21	39
30	20	19	95	374,747	538,876	15,886	37	53

prove that this iteration budget generally is a good choice. Figure 2 has the results when the problems with $n = 30$ variables are solved on 64 processors. Recall from Sect. 6 that this problem set produces the strongest demonstration that NOMAD is best when compared with DENCON and DENPAR. Yet, Fig. 2 shows that this best-case performance of NOMAD is significantly improved upon by either of the two hybrid methods regardless of the τ tolerance.

This strong result motivates the daring conjecture that both hybrid methods improve upon the performance of NOMAD for the entire problem set as well as for the three problem subsets defined by $n = 10$, 20, and 30, and also for a large range of iteration budgets. The conjecture is proved in Appendix A, except for two cases solved with an iteration budget of 500 that later is seen to be quite non-optimal.

Tables 3 and 4 show for an iteration budget of 2000 the following efficiency ratios $s/(64 \cdot c)$ when the entire set of problems is solved: for NOMAD–DENCON 18%, and for NOMAD–DENPAR, 21%. It is also interesting to examine the total number of evaluations s with single processor and the total number of cycles c with 64 processors, again when the entire set of problems is solved. For NOMAD–DENCON, the numbers are $s = 600{,}504$ and $c = 53{,}019$, and for NOMAD–DENPAR, $s = 629{,}919$ and $c = 47{,}922$. Comparing these numbers pairwise, we see that NOMAD–DENPAR uses 5% more evaluations than NOMAD–DENCON on a single processor, and that NOMAD–DENCON uses 11% more cycles than NOMAD–DENPAR on 64 processors.

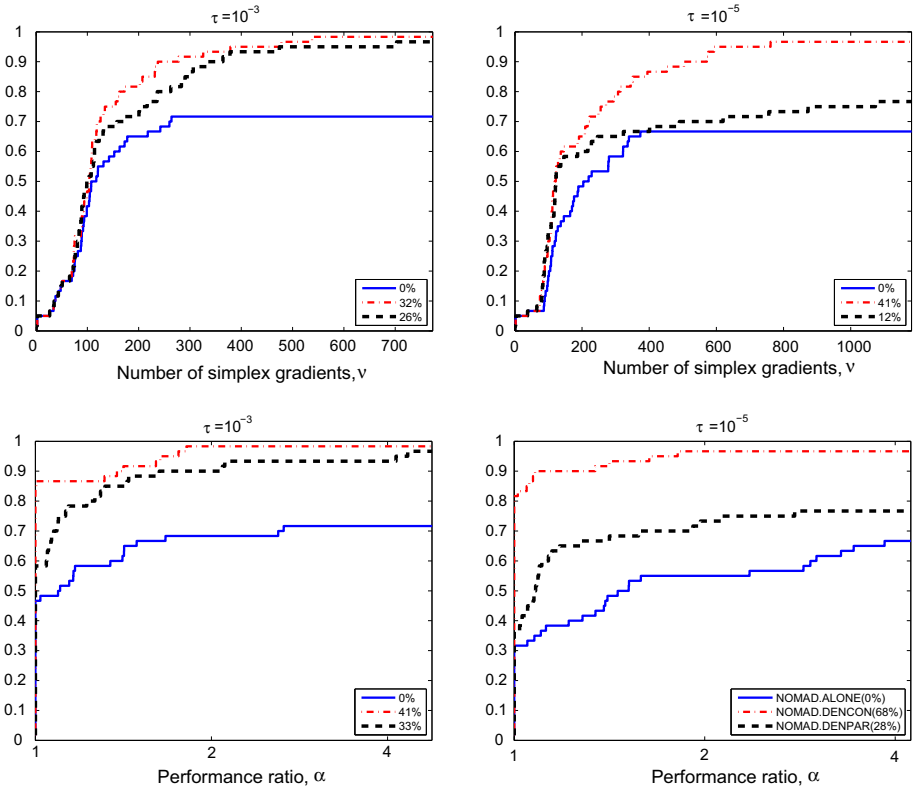

Fig. 3 Data (*top*) and performance (*bottom*) profiles for NOMAD, NOMAD–DENCON, and NOMAD–DENPAR on single processor, all problems, iter. budget = 2000

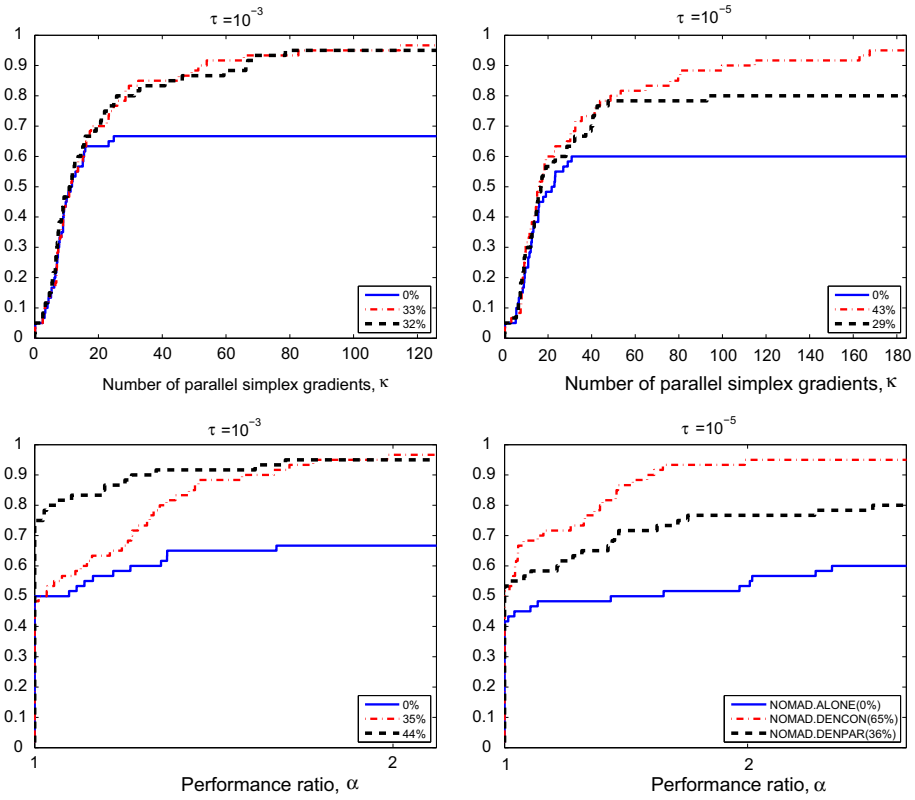

Fig. 4 Data (*top*) and performance (*bottom*) profiles for NOMAD, NOMAD–DENCON, and NOMAD–DENPAR on 64 processors, all problems, iter. budget = 2000

We back up the decision in favor of NOMAD–DENCON using Figs. 3 and 4. They compare NOMAD, NOMAD–DENCON, and NOMAD–DENPAR running the entire problem set on a single processor or 64 processors, with iteration budget equal to 2000. Evidently, NOMAD–DENCON is almost uniformly best, except for the performance profile for 64 processors and $\tau = 10^{-3}$. Thus, NOMAD–DENCON is the best choice for the problem set at hand, just as concluded earlier.

Furthermore, we put the results in perspective using an assumed time of 1 sec per evaluation of a solution vector and computing the average solution time per problem based on the total solution time for all problems. For NOMAD–DENCON, that average time per problem on a single processor is $600504/60 = 10008$ s or 2.7 h. On 64 processors, it is $53019/60 = 883$ s or 15 min, a significant reduction.

Finally, we include Table 5 showing performance details for a specific problem (namely, instance 1 of the 20 variables problem obtained by pairing the functions maxl and sin, see Sect. 5 for details). Each line of the table shows the objective function values produced after a given number of single processor evaluations (*s* evals.). For example, after 6000 evaluations, NOMAD, NOMAD–DENCON, and NOMAD–DENPAR achieve objective values 4.44, 0.00181, and 0.105, respectively. The three methods terminate with objective values 2.70, 0.00039, and 0.0058, respectively. Evidently, NOMAD–DENCON is the winner, NOMAD–DENPAR is a reasonably close second, while NOMAD has essentially failed.

Table 5 Progress of the best objective function value found by NOMAD, NOMAD–DENCON, NOMAD–DENPAR on single processor for problem maxl.sin (version 1, 20 variables)

s evals.	NOMAD	NOMAD–DENCON	NOMAD–DENPAR
1	2.3399100e+02	2.3399100e+02	2.3399100e+02
500	4.6491500e+01	4.6491500e+01	4.6491500e+01
1000	3.1135700e+01	3.1135700e+01	3.1135700e+01
1500	2.5305800e+01	2.5305800e+01	2.5305800e+01
2000	1.8197400e+01	1.8197400e+01	1.8197400e+01
2500	1.3587100e+01	5.1947500e+00	1.4306500e+01
3000	1.0928000e+01	1.5227300e+00	1.3936000e+01
3500	8.4306700e+00	6.1823200e−03	1.3936000e+01
4000	7.3690100e+00	1.8121800e−03	1.3929300e+01
4500	6.2703800e+00	1.8121800e−03	1.9749700e−01
5000	5.1442400e+00	1.8121800e−03	1.9749700e−01
5500	5.0707000e+00	1.8121800e−03	1.9749700e−01
6000	4.4412800e+00	1.8121800e−03	1.0594400e−01
6500	2.8666200e+00	1.8121800e−03	2.6513600e−02
7000	2.7070500e+00	5.4132200e−04	1.6792700e−02
7500	2.7070500e+00	3.9339600e−04	1.2553900e−02
8000	2.7070500e+00	3.9339600e−04	1.2553900e−02
8500	2.7070500e+00	3.9339600e−04	1.2553900e−02
9000	2.7070500e+00	3.9339600e−04	1.2553900e−02
9500	2.7070500e+00	3.9339600e−04	1.2496700e−02
10000	2.7070500e+00	3.9339600e−04	1.2496700e−02
10500	2.7070500e+00	3.9339600e−04	1.2496700e−02
11000	2.7070500e+00	3.9339600e−04	6.4966600e−03
11500	2.7070500e+00	3.9339600e−04	6.4966600e−03
12000	2.7070500e+00	3.9339600e−04	5.8789300e−03

Iteration budget is 2000

10 Conclusions

A straightforward combination of stand-alone methods for optimization of constrained optimization problems has been shown to produce hybrid methods that are significantly better than the best of the component methods. The approach amounts to a simple, iterative execution of the component methods. The implementation does not require modification of the component methods except for the input of the best solution obtained so far as starting solution. Thus, methods previously thought of moderate interest may acquire new importance in readily constructed hybrid schemes.

Acknowledgements We are thankful to an anonymous reviewer for helpful comments and suggestions.

Appendix

A More comparisons

We show that the iteration budget of 2000 evaluation generally is a good choice.

To start, we define six candidate iteration budgets, of size 500, 1000, 2000, 3000, 4000, and 5000. That choice is based on earlier runs involving hard engineering problems where 1000 and 2000 turned out to be good choices.

We construct the graphs showing the data and performance profiles for NOMAD, NOMAD–DENCON, and NOMAD–DENPAR for the selected six iteration budgets. Space restrictions prevent inclusion of the graphs; they are available at http://www.iasi.cnr.it/~liuzzi/hybridDF/.

From the graphs, we derive tables structured like Table 1, which summarize the relative size of areas under the profile curves of the various graphs. The tables are included in the appendix as Tables 6, 7, 8, 9, 10 and 11. The percentages in the tables show that NOMAD always produces the smallest area and thus has worst performance except for two cases in Table 6 where the problems with $n = 20$ variables are solved on 64 processors, and where NOMAD dominates NOMAD–DENCON but still is dominated by NOMAD–DENPAR.

The results of Tables 6, 7, 8, 9, 10 and 11 are reassuring in the sense that the dominance of NOMAD–DENCON and NOMAD–DENPAR over NOMAD does not depend on a critical choice of the iteration budget. There is an intuitive explanation for this results. During each iteration, the selected method may terminate due to convergence conditions and thus may not reach the iteration budget. In such a case, a larger value of the iteration budget will induce the same behavior. A corollary of this argument is that we always should prefer component methods that have well-justified convergence conditions, i.e., conditions based on a sound convergence analysis, as is the case for the three methods NOMAD, DENCON, and DENPAR selected here.

We interrupt the analysis of Tables 6, 7, 8, 9, 10 and 11 and look at the efficiency of the hybrid methods under parallelization, again considering the six iteration budgets. The relevant results are compiled in Table 3 for NOMAD–DENCON and in Table 4 for NOMAD–DENPAR. The interpretation is analogous to that for Table 2. For each problem subset, the efficiency ratios $s/(64 \cdot c)$ are very similar regardless of the iteration budget. For example, when the entire problem set is solved, then for NOMAD–DENCON the ratio ranges from 18 to 25%, and for NOMAD–DENPAR from 21 to 27%. These results indicate that the efficiency under parallelization is not very sensitive with respect to the iteration budget.

After this general investigation of the impact of the iteration budget, we turn to the problem of deciding a best iteration budget. For the selection, we compute the profile graphs and tables evaluating the performance of NOMAD–DENCON under the six iteration budgets. The graphs are omitted here; they are available at http://www.iasi.cnr.it/~liuzzi/hybridDF/. The graphs are summarized in Tables 12 and 13 for NOMAD–DENCON and Tables 14 and 15 for NOMAD–DENPAR.

We analyze the tables. Table 12 has the percentages for NOMAD–DENCON when running on a single processor. The bold numbers, indicating maximum as before, occur mostly in the rows for an iteration budget of 2000. Thus, that number is a good choice. Table 13, which covers NOMAD–DENCON and 64 processors, results in the same conclusion.

But Tables 14 and 15 do not lead to such clear-cut choices. Nevertheless, in Table 14 half of the bold entries occur in rows for the iteration budget of 2000. But Table 15 provides no significant insight. The reason becomes clear when we look at the corresponding graph,

Table 6 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 500

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	10	1	29	17
NOMAD–DENPAR	**26**	**25**	**47**	**47**
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	21	**8**	53	**47**
NOMAD–DENPAR	**30**	4	**54**	32
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	12	7	42	33
NOMAD–DENPAR	**15**	**27**	**47**	**67**
One processor, $n = 30$ variables				
NOMAD	7	16	0	0
NOMAD–DENCON	0	0	16	2
NOMAD–DENPAR	**37**	**59**	**56**	**81**
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	9	6	17	23
NOMAD–DENPAR	**22**	**24**	**34**	**30**
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	10	**7**	22	**39**
NOMAD–DENPAR	**21**	2	**41**	1
64 processors, $n = 20$ variables				
NOMAD	6	25	0	8
NOMAD–DENCON	0	0	12	0
NOMAD–DENPAR	**30**	**54**	**45**	**51**
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	7	22	13	58
NOMAD–DENPAR	**15**	**39**	**20**	**66**

provided at http://www.iasi.cnr.it/~liuzzi/hybridDF/. That graph shows the profile curves bunched together, and we can accept an iteration budget of 2000 as reasonable choice, in tune with the decisions deduced from the other tables.

We conclude that an iteration budget of 2000 is a reasonable choice for both NOMAD–DENCON and NOMAD–DENPAR. For that choice, Table 8 indicates improvement percentages of NOMAD–DENCON ranging from 32 to 68% over NOMAD when the entire problem set is solved on a single processor, and ranging from 33 to 65% for 64 processors. The corresponding percentages for NOMAD–DENPAR are 12 to 33% for single processor

Table 7 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 1000

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	19	**23**	39	**63**
NOMAD–DENPAR	**27**	20	**45**	40
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**34**	**30**	**47**	**56**
NOMAD–DENPAR	31	16	36	24
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	15	**22**	**59**	**75**
NOMAD–DENPAR	**20**	16	58	40
One processor, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	4	5	31	**75**
NOMAD–DENPAR	**27**	**26**	**51**	74
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	25	**36**	34	**56**
NOMAD–DENPAR	**34**	34	**48**	45
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	31	**43**	45	**74**
NOMAD–DENPAR	**48**	29	**63**	29
64 processors, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	3	5	21	32
NOMAD–DENPAR	**17**	**21**	**43**	**46**
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	9	20	29	81
NOMAD–DENPAR	**16**	**26**	**38**	**87**

and 29 to 44% for 64 processors. Thus, NOMAD–DENCON is clearly better than NOMAD–DENPAR on a single processor as well as on 64 processors.

B Convergence analysis for DENPAR

This section is devoted to the convergence analysis of Algorithm DENPAR. In this section, we consider problem (3) where the objective function is assumed to be quasi-convex Shetty and Bazaraa (1979), i.e., for each $x, y \in \mathbb{R}^n$,

Table 8 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 2000

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	**32**	**41**	**41**	**68**
NOMAD–DENPAR	26	12	33	28
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**22**	**30**	36	**45**
NOMAD–DENPAR	**22**	12	**40**	12
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**29**	**40**	**35**	**79**
NOMAD–DENPAR	24	19	29	45
One processor, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**31**	**46**	**47**	**100**
NOMAD–DENPAR	16	3	28	37
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	**33**	**43**	35	**65**
NOMAD–DENPAR	32	29	**44**	36
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	44	**53**	53	**80**
NOMAD–DENPAR	**47**	40	**61**	38
64 processors, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	10	**11**	19	**56**
NOMAD–DENPAR	**11**	5	**29**	14
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	16	**35**	28	66
NOMAD–DENPAR	**20**	26	**45**	**68**

$$f(\lambda x + (1-\lambda)y) \leq \max\{f(x), f(y)\}, \qquad \text{for each } \lambda \in [0,1].$$

Algorithm DENPAR is based on DFN$_{simple}$ from Fasano et al. (2014). We review the latter scheme, beginning with the definition of Clarke stationarity (see, e.g., Clarke 1983).

Definition B.1 *(Clarke Stationarity)* Given the unconstrained problem $\min_{x\in\mathbb{R}^n} f(x)$, a point $\bar{x}$ is a Clarke stationary point if $0 \in \partial f(\bar{x})$, where $\partial f(x) = \{s \in \mathbb{R}^n : f^{Cl}(x;d) \geq d^T s,\ \forall d \in \mathbb{R}^n\}$ is the generalized gradient of f at x, and

Table 9 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 3000

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	**26**	**34**	**36**	**58**
NOMAD–DENPAR	**26**	15	33	23
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	25	**28**	28	**37**
NOMAD–DENPAR	**33**	14	**42**	13
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**22**	**27**	**29**	**66**
NOMAD–DENPAR	21	14	24	34
One processor, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**30**	**41**	**41**	**83**
NOMAD–DENPAR	18	13	29	35
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	31	**46**	34	**73**
NOMAD–DENPAR	**37**	31	**45**	41
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	39	**53**	49	**77**
NOMAD–DENPAR	**52**	41	**67**	38
64 processors, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	9	12	16	55
NOMAD–DENPAR	**15**	**13**	**25**	**37**
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	34	**43**	41	**89**
NOMAD–DENPAR	**37**	30	**48**	47

$$f^{Cl}(x; d) = \limsup_{y \to x, t \downarrow 0} \frac{f(y + td) - f(y)}{t}. \tag{7}$$

We also need the definition of dense subsequences.

Definition B.2 *(Dense subsequence)* Let K be an infinite subset of indices (possibly $K = \{0, 1, \ldots\}$). The subsequence of normalized directions $\{d_k\}_K$ is said to be dense in the unit sphere $S(0, 1)$, if for any $\bar{D} \in S(0, 1)$ and for any $\epsilon > 0$ there exists an index $k \in K$ such that $\|d_k - \bar{D}\| \leq \epsilon$.

Table 10 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 4000

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	28	**37**	34	**54**
NOMAD–DENPAR	**29**	17	**35**	21
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	26	**26**	30	**37**
NOMAD–DENPAR	**33**	14	**42**	13
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	24	**36**	26	**71**
NOMAD–DENPAR	**25**	18	**28**	39
One processor, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**23**	**36**	**45**	**69**
NOMAD–DENPAR	15	7	31	24
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	33	**43**	35	**63**
NOMAD–DENPAR	**39**	31	**45**	42
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	42	**55**	47	**72**
NOMAD–DENPAR	**56**	42	**67**	39
64 processors, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	9	9	14	37
NOMAD–DENPAR	**15**	**12**	**26**	**42**
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	30	**40**	36	**87**
NOMAD–DENPAR	**33**	33	**46**	51

Table 11 NOMAD, NOMAD–DENCON, and NOMAD–DENPAR, iter. budget = 5000

τ	Data profiles		Perf. profiles	
	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
One processor, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	25	**32**	31	**49**
NOMAD–DENPAR	**29**	15	**36**	19
One processor, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	25	**26**	29	**37**
NOMAD–DENPAR	**32**	14	**42**	13
One processor, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	12	**25**	13	**52**
NOMAD–DENPAR	**21**	18	**33**	26
One processor, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	**32**	**40**	**40**	**62**
NOMAD–DENPAR	31	15	34	25
64 processors, all problems				
NOMAD	0	0	0	0
NOMAD–DENCON	35	**46**	37	**62**
NOMAD–DENPAR	**41**	33	**47**	42
64 processors, $n = 10$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	45	**58**	49	**77**
NOMAD–DENPAR	**56**	41	**67**	38
64 processors, $n = 20$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	14	**17**	28	**41**
NOMAD–DENPAR	**15**	16	**28**	39
64 processors, $n = 30$ variables				
NOMAD	0	0	0	0
NOMAD–DENCON	30	**36**	27	**83**
NOMAD–DENPAR	**36**	33	**48**	63

Table 12 NOMAD–DENCON on single processor, with various iter. budgets

Iter. budget	Data profiles		Perf. profiles	
τ	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
All problems				
500	0	0	0	0
1000	12	38	10	47
2000	**21**	**53**	**19**	**56**
3000	17	47	12	49
4000	16	48	10	50
5000	15	48	9	50
$n = 10$ variables				
500	0	0	8	0
1000	**14**	41	**16**	**33**
2000	12	**42**	12	28
3000	3	34	0	16
4000	3	34	0	16
5000	4	34	1	17
$n = 20$ variables				
500	0	0	7	0
1000	4	27	12	37
2000	**16**	**42**	**19**	**44**
3000	11	33	10	35
4000	9	39	8	39
5000	5	33	0	31
$n = 30$ variables				
500	0	0	0	0
1000	18	43	17	56
2000	**39**	81	**47**	**91**
3000	37	80	41	81
4000	35	77	37	71
5000	37	**82**	39	80

Table 13 NOMAD–DENCON on 64 processors, with various iter. budgets

Iter. budget	Data profiles		Perf. profiles	
τ	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
All problems				
500	0	0	0	0
1000	8	23	11	29
2000	17	**51**	17	**58**
3000	18	40	19	45
4000	19	35	19	39
5000	**21**	44	**22**	48
$n = 10$ variables				
500	0	0	0	0
1000	6	13	9	30
2000	25	34	25	48
3000	25	35	26	52
4000	23	35	23	51
5000	**26**	**39**	**27**	**54**
$n = 20$ variables				
500	0	0	0	0
1000	10	38	14	55
2000	13	69	15	**110**
3000	8	56	7	75
4000	6	58	5	61
5000	**16**	**74**	**16**	93
$n = 30$ variables				
500	0	0	0	0
1000	10	6	9	6
2000	11	**13**	10	**19**
3000	19	13	18	12
4000	**22**	8	**22**	12
5000	21	12	21	11

Table 14 NOMAD–DENPAR on single processor, with various iter. budgets

Iter. budget	Data profiles		Perf. profiles	
τ	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
All problems				
500	3	0	**5**	1
1000	**4**	**8**	4	8
2000	2	5	3	**11**
3000	0	6	0	6
4000	0	5	0	0
5000	2	6	1	1
$n = 10$ variables				
500	2	0	**15**	**27**
1000	**4**	11	10	13
2000	4	**14**	5	20
3000	1	14	1	9
4000	0	9	0	0
5000	0	9	0	0
$n = 20$ variables				
500	2	0	14	11
1000	2	8	**15**	22
2000	**4**	**11**	10	**33**
3000	0	4	0	17
4000	1	7	0	6
5000	1	5	0	0
$n = 30$ variables				
500	**9**	15	**13**	5
1000	9	**18**	9	**15**
2000	0	0	3	0
3000	0	12	0	5
4000	1	11	0	3
5000	5	17	5	9

Table 15 NOMAD–DENPAR on 64 processors, with various iter. budgets

Iter. budget	Data profiles		Perf. profiles	
τ	10^{-3} (%)	10^{-5} (%)	10^{-3} (%)	10^{-5} (%)
all problems				
500	0	0	0	0
1000	4	11	3	**11**
2000	3	**15**	1	4
3000	9	9	4	4
4000	11	7	5	1
5000	**12**	10	**7**	2
$n = 10$ variables				
500	0	0	1	0
1000	9	21	0	30
2000	15	26	10	26
3000	22	30	19	35
4000	**23**	**33**	**20**	**40**
5000	23	33	20	37
$n = 20$ variables				
500	**13**	0	**24**	14
1000	9	**7**	19	**27**
2000	2	3	5	10
3000	1	3	1	10
4000	0	0	0	3
5000	4	3	5	0
$n = 30$ variables				
500	0	**18**	0	**20**
1000	5	12	5	14
2000	2	0	1	7
3000	13	6	11	2
4000	15	6	14	0
5000	**16**	11	**15**	6

Here is a summary of DFN$_{simple}$ for the solution of Problem (3).

Algorithm 5: DFN$_{simple}$ from Fasano et al. (2014)

Input: Problem (3).
Output: Sequences $\{x_k\}$, $\{\alpha_k\}$ and $\{\tilde{\alpha}_k\}$.
Choose $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0 > 0$, a sequence $\{d_k \in \mathbb{R}^n : \|d_k\| = 1\}$, and set $k \leftarrow 0$
repeat
  if $f(x_k + \tilde{\alpha}_k d_k) > f(x_k) - \gamma(\tilde{\alpha}_k)^2$ and $f(x_k - \tilde{\alpha}_k d_k) > f(x_k) - \gamma(\tilde{\alpha}_k)^2$ **then**
    Set $\tilde{\alpha} \leftarrow \alpha_k \leftarrow 0$, $\tilde{\alpha}_{k+1} \leftarrow \tilde{\alpha}_k/2$, $d^+ \leftarrow d_k$
  else if $f(x_k + \tilde{\alpha}_k d_k) \le f(x_k) - \gamma(\tilde{\alpha}_k)^2$ **then** Set $d^+ \leftarrow d_k$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k$
  else Set $d^+ \leftarrow -d_k$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k$
  end if
  if $\tilde{\alpha} > 0$ **then**
    Compute $\hat{\alpha} =$ `Expansion Step`$(\tilde{\alpha}, x_k, d^+)$ and set $\alpha_k \leftarrow \hat{\alpha}$, $\tilde{\alpha}_{k+1} \leftarrow \hat{\alpha}$
  end if
  Set $\tilde{x}_k \leftarrow x_k + \alpha_k d^+$
  Find $x_{k+1} \in \mathbb{R}^n$ s.t. $f(x_{k+1}) \le f(\tilde{x}_k)$ or set $x_{k+1} \leftarrow \tilde{x}_k$, set $k \leftarrow k + 1$.
until convergence

In Algorithm DFN$_{simple}$, a predefined sequence of search directions $\{d_k\}$ is used. Then, the behavior of the function $f(x)$ along the direction d_k is investigated. If the direction (or its opposite) is deemed a good direction, in the sense that sufficient decrease can be obtained along it, then a sufficiently large step size is computed by means of the `Expansion Step` procedure. On the other hand, if neither d_k nor $-d_k$ are good direction, then the tentative step size is reduced by a constant factor.

Note that, in Algorithm DFN$_{simple}$, considerable freedom is left for the selection of the next iterate x_{k+1} once the new point $\tilde{x}_k$ has been computed. More specifically, the next iterate x_{k+1} is only required to satisfy inequality $f(x_{k+1}) \le f(\tilde{x}_k)$. This can trivially be satisfied by setting $x_{k+1} \leftarrow \tilde{x}_k$. However, more sophisticated selection strategies can be implemented. For instance, x_{k+1} might be defined by minimizing suitable approximating models of the objective function, thus possibly improving the efficiency of the overall scheme. As we shall see, this freedom offered by DFN$_{simple}$ is particularly useful for our purposes.

Next we describe a parallelized version of DFN$_{simple}$ called DEN$_{check}$.

B.1 Algorithm DEN$_{check}$

Here is a summary of the algorithm.

Algorithm 6: DEN$_{check}$

Input: Problem (3).
Output: Sequences $\{x_k\}$, $\{d_k\}$, $\{\alpha_k^i\}$ and $\{\tilde{\alpha}_k^i\}$, $i = 1, \ldots, n$
Choose $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, n$, a sequence $\{\hat{d}_k \in \mathbb{R}^n : \|\hat{d}_k\| = 1\}$, and set $k \leftarrow 0$
1: **repeat**
2: Given d_k, form an orthonormal basis $\{d_k^1, \ldots, d_k^n\}$ of $\mathbb{R}^n$, where $d_k^1 = d_k$
3: Evaluate in parallel points in $\mathcal{M}(x_k, D_k, \tilde{\alpha}_k) \cup_{i=1}^n \mathcal{E}(x_k, d_k^i, \tilde{\alpha}_k) \cup_{i=1}^n \mathcal{E}(x_k, -d_k^i, \tilde{\alpha}_k)$
4: **for** $i = 1, 2, \ldots, n$ **do**
5: **if** $f(x_k + \tilde{\alpha}_k^i d_k^i) > f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ and $f(x_k - \tilde{\alpha}_k^i d_k^i) > f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ **then**
6: Set $\tilde{\alpha} \leftarrow \alpha_k^i \leftarrow 0$, $\tilde{\alpha}_{k+1}^i \leftarrow \tilde{\alpha}_k^i/2$, $d^+ \leftarrow d_k^i$

else if $f(x_k + \tilde{\alpha}_k^i d_k^i) \leq f(x_k) - \gamma(\tilde{\alpha}_k^i)^2$ **then** Set $d^+ \leftarrow d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
else Set $d^+ \leftarrow -d_k^i$, $\tilde{\alpha} \leftarrow \tilde{\alpha}_k^i$
end if
if $\tilde{\alpha} > 0$ **then**
compute $\hat{\alpha} =$ `Expansion Step`$(\tilde{\alpha}, x_k, d^+)$
Set $\alpha_k^i \leftarrow \hat{\alpha}$, $\tilde{\alpha}_{k+1}^i \leftarrow \hat{\alpha}$, $\tilde{d}_k^i \leftarrow d^+$
end if
end for
if $\left(\sum_{i=1}^n \alpha_k^i\right) > 0$ **then**
Let j_M be such that $f(x_k + \alpha_k^{j_M} d_k^{j_M}) = \max_{\alpha_k^i > 0}\{f(x_k + \alpha_k^i d_k^i)\}$
Set $\tilde{x}_k \leftarrow x_k + \alpha_k^{j_M} d_k^{j_M}$
Compute $x_c = \frac{\sum_{i=1}^n x_k + \alpha_k^i d_k^i \alpha_k^i}{\sum_{i=1}^n \alpha_k^i}$ (*convex combination*)
Set $d_k = d_k^{j_M}$
if $f(x_c) \leq f(\tilde{x}_k)$ **then**
Set $x_{k+1} \leftarrow x_c$
else Set $x_{k+1} \leftarrow \tilde{x}_k$
end if
else Set $\tilde{x}_k \leftarrow x_k$, $x_{k+1} \leftarrow \tilde{x}_k$ and $d_k = \hat{d}_k$
end if
Set $k \leftarrow k + 1$.
until convergence

At every iteration of DEN$_{check}$, an orthonormal basis is formed starting from the given direction $\hat{d}_k$. First, the behavior of the objective function along directions $d_k^1, \ldots, d_k^n$ is investigated starting from the same point x_k. This produces step sizes $\alpha_k^i \geq 0$ and $\tilde{\alpha}_k^i > 0$, $i = 1, \ldots, n$. Provided that $\alpha_k^i > 0$ for at least an index i, the index j_M is computed and $\tilde{x}_k \leftarrow x_k + \alpha_k^{j_M} d_k^{j_M}$, that is $\tilde{x}_k$ is the point that produces the worst improvement for the objective function.

Additional computation is carried out if $\sum_{i=1}^n \alpha_k^i > 0$. In particular, the step sizes obtained by the n linesearches are combined together to define the convex combination point x_c. Then $f(x_c)$ is compared with $f(\tilde{x}_k)$. If $f(x_c)$ improves upon the latter value, then x_{k+1} is set equal to x_c, otherwise x_{k+1} is set equal to the previously computed $\tilde{x}_k$. The reader may wonder about the choice of $\tilde{x}_k$. We specify it here to get a theoretical scheme that can be readily converted to the more efficient DENPAR.

In the following proposition, we show that Algorithm DEN$_{check}$ inherits the convergence properties of the sequential code DFN$_{simple}$ by showing that DEN$_{check}$ is a particular case of the latter method.

Proposition B.3 *Let $\{x_k\}$ be the sequence produced by Algorithm DEN$_{check}$. Let $\bar{x}$ be any limit point of $\{x_k\}$ and K be the subset of indices such that*

$$\lim_{k \to \infty, k \in K} x_k = \bar{x}.$$

If the subsequence $\{d_k\}_K$ is dense in the unit sphere (see Definition B.2), then $\bar{x}$ is Clarke stationary for problem (3) (see Definition B.1).

Proof We prove the proposition by showing that DEN$_{check}$ is an instance of DFN$_{simple}$. To this aim, let us consider the last step of Algorithm DFN$_{simple}$, namely where the next

iterate x_{k+1} is defined. As it can be seen, in Algorithm DFN$_{simple}$, x_{k+1} is required to satisfy the condition $f(x_{k+1}) \leq f(\tilde{x}_k)$. Note that $\tilde{x}_k$ of DFN$_{simple}$ corresponds to the point $\tilde{x}_k$ of DEN$_{check}$. Indeed, if $\sum_{i=1}^{n} \alpha_k^i > 0$, then $\tilde{x}_k = x_k + \alpha_k^{j_M} d_k^{j_M}$ and $d_k = d_k^{j_M}$. Otherwise, $\tilde{x}_k = x_k$ and $d_k = \hat{d}_k$.

Now, let us consider an iteration k of Algorithm DEN$_{check}$. By the instructions of the algorithm, one of the following cases occurs.

(i) $\sum_{i=1}^{n} \alpha_k^i > 0$ and $f(x_c) \leq f(\tilde{x}_k)$;
(ii) $\sum_{i=1}^{n} \alpha_k^i > 0$ and $f(x_c) > f(\tilde{x}_k)$;
(iii) $\sum_{i=1}^{n} \alpha_k^i = 0$.

In case (i), $x_{k+1} \leftarrow x_c$ and the iteration of DEN$_{check}$ is like an iteration of DFN$_{simple}$ with $d_k = d_k^{j_M}$ and where x_{k+1} is chosen as point x_c.

In case (ii), $x_{k+1} \leftarrow \tilde{x}_k$ and the iteration of DEN$_{check}$ is like an iteration of DFN$_{simple}$ with $d_k = d_k^{j_M}$ and where x_{k+1} is set equal to $\tilde{x}_k$.

Finally, in case (iii), $x_{k+1} \leftarrow x_k$ and the iteration of DEN$_{check}$ is like an iteration of DFN$_{simple}$ with $d_k = \hat{d}_k$ and where sufficient improvement cannot be obtained both along d_k and $-d_k$.

Hence, any iteration of DEN$_{check}$ can be viewed as a particular iteration of DFN$_{simple}$. This establishes the proposition. □

DENPAR is derived from DEN$_{check}$ by replacing lines 20–23 by $x_{k+1} \leftarrow x_c$. Effectively, that replacement assumes that the inequality $f(x_c) \leq f(\tilde{x}_k)$ of line 20 is satisfied. This is indeed the case since $f(x)$ is assumed to be quasi-convex.

References

Abramson MA, Audet C, Couture G, Dennis Jr JE, Le Digabel S, Tribes C (2014) The NOMAD project. http://www.gerad.ca/nomad

Abramson MA, Audet C, Dennis JE Jr, Le Digabel S (2009) Orthomads: a deterministic mads instance with orthogonal directions. SIAM J Optim 20(2):948–966

Audet C, Dennis JE Jr (2006) Mesh adaptive direct search algorithms for constrained optimization. SIAM J Optim 17(1):188–217

Audet C, Dennis JE Jr, Le Digabel S (2008) Parallel space decomposition of the mesh adaptive direct search algorithm. SIAM J Optim 19(3):1150–1170

Bratley P, Fox B (1988) Algorithm 659: implementing Sobol's quasirandom sequence generator. ACM Trans Math Softw 14(1):88–100

Clarke FH (1983) Optimization and nonsmooth analysis. Wiley, New York

Dennis JE Jr, Torczon V (1991) Direct search methods on parallel machines. SIAM J Optim 1(4):448–474

Di Pillo G, Grippo L, Lucidi S (1993) A smooth method for the finite minimax problem. J Glob Optim 60:187–214

Fasano G, Liuzzi G, Lucidi S, Rinaldi F (2014) A linesearch-based derivative-free approach for nonsmooth constrained optimization. SIAM J Optim 24(3):959–992

García-Palomares UM, Rodríguez JF (2002) New sequential and parallel derivative-free algorithms for unconstrained minimization. SIAM J Optim 13(1):79–96

García-Palomares UM, García-Urrea IJ, Rodríguez-Hernández PS (2013) On sequential and parallel non-monotone derivative-free algorithms for box constrained optimization. Optim Methods Softw 28(6):1233–1261

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Boston

Gray GA, Kolda TG (2006) Algorithm 856: Appspack 4.0: asynchronous parallel pattern search for derivative-free optimization. ACM Trans Math Softw 32(3):485–507

Griffin JD, Kolda TG, Lewis RM (2008) Asynchronous parallel generating set search for linearly constrained optimization. SIAM J Sci Comput 30(4):1892–1924

Grippo L, Lampariello F, Lucidi S (1986) A nonmonotone line search technique for Newton's method. SIAM J Numer Anal 23(4):707–716

Halton JH (1960) On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer Math 2:84–90

Hough PD, Kolda TG, Torczon VJ (2001) Asynchronous parallel pattern search for nonlinear optimization. SIAM J Sci Comput 23(1):134–156

Kolda TG (2005) Revisiting asynchronous parallel pattern search for nonlinear optimization. SIAM J Optim 16(2):563–586

Kolda TG, Torczon V (2004) On the convergence of asynchronous parallel pattern search. SIAM J Optim 14(4):939–964

Laguna M, Molina J, Pérez F, Caballero R, Hernández-Díaz AG (2009) The challenge of optimizing expensive black boxes: a scatter search/rough set theory approach. J Oper Res Soc 61:53–67

Le Digabel S (2011) Algorithm 909: NOMAD: nonlinear optimization with the MADS algorithm. ACM Trans Math Softw 37(4):1–15

Meza JC, Oliva RA, Hough PD, Williams PJ (2007) Opt++: an object oriented toolkit for nonlinear optimization. ACM Trans Math Softw 33(2):12

Moré JJ, Wild SM (2009) Benchmarking derivative-free optimization algorithms. SIAM J Optim 20(1):172–191

Ponstein J (1967) Seven kinds of convexity. SIAM Rev 9(1):115–119

Shetty CM, Bazaraa MS (1979) Nonlinear programming: theory and algorithms. Wiley, Massachusetts

Sobol I (1977) Uniformly distributed sequences with an additional uniform property. USSR Comput Math Math Phys 16:236–242

Truemper K (in review) Simple seeding of evolutionary algorithms for hard multiobjective minimization problems