Efficient global unconstrained black box optimization

Morteza Kimiaei
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: kimiaeim83@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~kimiaei/`

Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~neum/`

March 29, 2019

Abstract. For the unconstrained global optimization of black box functions, this paper presents a new stochastic algorithm called *VSBBO*. In practice, *VSBBO* matches the quality of other state-of-the-art algorithms for finding, with reasonable accuracy, a global minimizer in small and large dimensions, or at least in the majority of cases a point of a quality comparable with the best competing algorithms.

For smooth, everywhere defined functions, it is proved that, with probability arbitrarily close to 1, one finds with $O(n\epsilon^{-2})$ function evaluations a point with gradient 2-norm $\leq \epsilon$. In the smooth convex case, this number improves to $O(n\epsilon^{-1})$ and in the smooth strongly convex case to $O(n \log \epsilon^{-1})$. This matches known recent complexity results for reaching a slightly different goal, namely the expected gradient 2-norm $\leq \epsilon$.

Keywords: Derivative-free optimization, complexity bounds, global optimization, sufficient decrease, line search.

2010 MSC Classification: primary 90C56

Contents

1 Introduction

We consider the unconstrained optimization problem of minimizing a function $f : \mathbb{R}^n \to \mathbb{R}$, assuming the avaialbility of an oracle that returns for a given $x \in \mathbb{R}^n$ the function value

$f(x)$. Neither gradients nor Lipschitz constants nor structural information about f are assumed to be available, though for convergence and/or complexity analysis one needs to make further assumptions.

This problem (see, e.g., [7, 40]) is usually called black box optimization (BBO) or derivative-free optimization (DFO). A huge literature exists about the problem, and we only mention a few pointers to the literature. Past software of our group on BBO includes the deterministic algorithms *GRID* [17, 18] and *MCS* [25] and the stochastic algorithms *SnobFit* [26] and *VXQR* [35]. Software by many others is mentioned in our extensive numerical comparison; see Section 5.2.

The techniques for solving BBO problems fall into two classes, deterministic and stochastic methods. We mainly discuss the stochastic case; for deterministic methods see, e.g., the book by Conn et al. [7] and its many references. Stochastic methods for BBO go back to [42] and were later discussed especially in the framework of evolutionary optimization [2, 23, 41].

Our goal in this paper is to describe a new, practically very efficient stochastic method, called *VSBBO*, for which good complexity results can be proved.

In theory, the complexity of *VSBBO* for reaching a given accuracy with probability arbitrarily close to 1 matches the known recent complexity results for reaching a slightly different goal, namely the expected gradient 2-norm $\leq \epsilon$.

In practice, *VSBBO* matches the quality of other state-of-the-art algorithms for finding, with reasonable accuracy, a global minimizer in small and large dimensions, or at least in the majority of cases a point of a quality comparable with the best competing algorithms.

An algorithm loosely related to *VSBBO* (but without complexity guarantees) is the *Hit-and-Run* algorithm by Bélisle [3].

1.1 Complexity

The goal is to find an efficient algorithm that, starting from a point x^0, finds with high probability and at most $N(\varepsilon)$ function evaluations a point $x_{\rm best}$ satisfying[1]

$$f(x_{\rm best}) \leq \sup\{f(x) \leq f(x^0) \mid \|g(x)\|_* \leq \varepsilon\}. \tag{1}$$

Here we use a scaled 2-norm $\|p\|$ and its dual norm $\|g\|_*$ of $p, g \in \mathbb{R}^n$, defined by

$$\|p\| := \sqrt{\sum p_i^2/s_i^2}, \quad \|g\|_* := \sqrt{\sum s_i^2 g_i^2} \quad (\text{all } s_i > 0) \tag{2}$$

in terms of a positive scaling vector $s \in \mathbb{R}^n$.

[1] To see the meaning of the condition (1) we consider the example of a strictly convex quadratic function $f(x) = \xi + c^T x + \frac{1}{2} x^T G x$ with symmetric, positive definite G. If $\widehat{x}$ denotes the minimizer then (1) is equivalent with $f(x) - f(\widehat{x}) \leq \varepsilon/2\lambda_{\min}$, where $\lambda_{\min}$ denotes the smallest eigenvalue of G. Indeed, the maximum is attained at $x = \widehat{x} + p$, where p is an eigenvector of G of length $\lambda_{\min}^{-1}\sqrt{\varepsilon}$ corresponding to the smallest eigenvalue. Thus the flattest direction on a level set determines the quality of the resulting bound.

Complexity bounds limit the size of $N(\varepsilon)$. The appropriate asymptotic form for the expression $N(\varepsilon)$, found by Vicente [44], Dodangeh & Vicente [14], Dodangeh, Vicente & Zhang [15], Gratton et al. [20], Bergou, Gorbunov & Richtárik [4], and Nesterov & Spokoiny [33, 34], depends on the properties (smooth, smooth convex, or smooth strongly convex) of f; cf. Subsection 2.1 below. Standard assumptions for the complexity analysis of BBO algorithms are:

(A1) The function f is continuously differentiable on $\mathbb{R}^n$, and its gradient is Lipschitz continuous with Lipschitz constant L.

(A2) On the level set

$$\mathcal{L}(x_0) := \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$$

of x_0, the objective function f is bounded from below.

case	goal	complexity
nonconvex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}\left(n\varepsilon^{-2}\right)$
convex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}\left(n\varepsilon^{-1}\right)$
convex	$\mathbf{E}(f - \widehat{f}) \leq \varepsilon$	$\mathcal{O}\left(n\varepsilon^{-1}\right)$
strongly convex	$\mathbf{E}(\|g\|_*) \leq \varepsilon$	$\mathcal{O}\left(n \log \varepsilon^{-1}\right)$
strongly convex	$\mathbf{E}(f - \widehat{f}) \leq \varepsilon$	$\mathcal{O}\left(n \log \varepsilon^{-1}\right)$

Table 1: Complexity results for stochastic BBO in expectation (Gratton et al. [20] for the nonconvex case, Bergou et al. [4] for all cases)

Gratton et al. [20] studied direct search with probabilistic descent. If, in each poll step, one chooses a fixed number of directions uniformly independently distributed on the unit sphere, they proved that with overwhelmingly high probability a complexity bound $\mathcal{O}(n\varepsilon^{-2})$ holds. Bergou et al. [4] and Nesterov & Spokoiny [34] generalized this result to give algorithms with complexity results for the nonconvex, convex and strongly convex case shown in Table 1. In each case, the bounds are better by a factor of n than the best known complexity results for deterministic algorithms (by Dodangeh & Vicente [14], Vicente [44] and Konečný & Richtárik [30]) given in Table 2. Of course, being a stochastic algorithm, the performance guarantee obtained by Bergou et al. is slightly weaker, only valid in expectation.

case	goal	complexity
nonconvex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}\left(n^2\varepsilon^{-2}\right)$
convex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}\left(n^2\varepsilon^{-1}\right)$
convex	$f - \widehat{f} \leq \varepsilon$	$\mathcal{O}\left(n^2\varepsilon^{-1}\right)$
σ-strongly convex	$\|g\|_* \leq \varepsilon$	$\mathcal{O}\left(n^2 \log \varepsilon^{-1}\right)$
σ-strongly convex	$f - \widehat{f} \leq \varepsilon$	$\mathcal{O}\left(n^2 \log \varepsilon^{-1}\right)$

Table 2: Complexity results for deterministic BBO (Vicente [44] for the nonconvex case, Dodangeh & Vicente [14] for the convex and the strongly convex cases, Konečný & Richtárik [30] for all cases)

case	goal	complexity
nonconvex	$\Pr(\|g\|_* \le \varepsilon) \ge 1-\eta$	$\mathcal{O}\left(n\varepsilon^{-2}\right)$
convex	$\Pr(\|g\|_* \le \varepsilon) \ge 1-\eta$	$\mathcal{O}\left(n\varepsilon^{-1}\right)$
convex	$\Pr(f-\widehat{f} \le \varepsilon) \ge 1-\eta$	$\mathcal{O}\left(n\varepsilon^{-1}\right)$
σ-strongly convex	$\Pr(\|g\|_* \le \varepsilon) \ge 1-\eta$	$\mathcal{O}\left(n\log\varepsilon^{-1}\right)$
σ-strongly convex	$\Pr(f-\widehat{f} \le \varepsilon) \ge 1-\eta$	$\mathcal{O}\left(n\log\varepsilon^{-1}\right)$

Table 3: Complexity results for stochastic BBO with probability $1-\eta$, for fixed $0<\eta<1$ (present paper)

1.2 Algorithms and data structures

In the present paper we describe and analyze a new stochastic method, the *Vienna stochastic black box optimization algorithm* (*VSBBO*). It gives the same order of complexity as the one by Bergou et al. but with a guarantee that holds with probability arbitrarily close to 1; see Table 3. Numerical results in Section 5 show that, in practice, *VSBBO* matches the quality of other state-of-the-art algorithms for BBO, including those with good heuristics but without a complexity guarantee.

To be competitive with the state of the art – which means finding, with reasonable accuracy, a global minimizer in small and large dimensions, or at least in the majority of cases a point as good as competing algorithms – requires our algorithm to be quite complex. As a consequence, the algorithm manipulates many quantities, which for convenience are grouped into a number of separate data structures to which the subalgorithms may have access if needed. On the other hand, to be able to prove complexity results requires a detailed description of all steps. To present the algorithms in a concise way, we compiled the variables of all data structures in Table 5, and the in-out dependence of the algorithms (described later in full detail) on the data structure in Table 4. Flow charts for *setScale*, *MLS*, *FDS*, and *VSBBO* can be found in Figures 12 and 11 of Appendix C.

Algorithm 2.2	**function** [`step`] = **updateCum**(`point`, `step`, `tune`)
Algorithm 2.3	**function** [`step`] = **lbfgsDir**(`point`, `step`);
Algorithm 2.4	**function** [`step`] = **enforceAngle**(`point`, `step`, `tune`);
Algorithm 2.5	**function** [`step`] = **direction**(`point`, `step`, `par`, `tune`)
Algorithm 2.6	**function** [`point`] = **updateSY**(`point`, `tune`)
Algorithm 2.7	**function** [`point`] = **updateXF**(`point`, `tune`)
Algorithm 2.9	**function** [`point`, `par`] = **MLS**(`fun`, `point`, `step`, `par`, `tune`)
Algorithm 3.1	**function** [`point`, `step`, `par`] = **setScales** (`fun`, `point`, `step`, `par`, `tune`)
Algorithm 3.2	**function** [`point`] = **FDS**(`fun`, `point`, `step`, `par`, `tune`)
Algorithm 3.4	**function** [x,f] = **VSBBO**(`fun`, x, `tune`)

Table 4: List of algorithms defined in present paper. The main algorithm *VSBBO* solves a BBO problem; the others are called within *VSBBO*.

VSBBO initally calls the algorithm *setScales* to estimate a good scaling of norms, step lengths, and related control parameters. Then it uses in each iteration the fixed decrease search algorithm *FDS*, aimed at repeatedly reducing the function value by an amount of at least Δ to update the best point. If no progress is made in a call to *FDS*, Δ is reduced by a factor of Q. Once Δ is below a minimum threshold, the algorithm stops.

Both *setScales* and *FDS* work by making repeated calls to the multi-line search *MLS*. *MLS* polls in a number of suitable chosen directions (defined by *direction*) in a line search fashion a few objective function values each in the hope of reducing the objective by more than Δ.

direction generates 5 kinds of direction vectors: coordinate directions, limited memory quasi-Newton directions, random subspace directions, random directions, and cumulative directions, explained in more detail in Subsections 2.2 and 2.4. Finally, *updateSY*, *updateXF* and *updateCum* are auxiliary routines for updating the data needed for calculating, limited memory quasi-Newton steps, random subspace steps and cumulative steps, respectively.

`fun` (function handle for the objective function f)
`point` (structure with information about points and function values)
m, X, F (list of the m best points so far and their function values) b, $x = X_{:b}$, $f = F_b$ (best point and its function value) x_{init}, f_{init} (initial point and its function value), `xr`, `fr` (newest point and its function value) `xm`, `fm` (point `xr` $- p$ and its function value), `xl`, `fl` (point at `xr` $- 2p$ and its function value) `dxg` (parameter for quasi-Newton), $\tilde{g}$ and $\tilde{g}_{\text{old}}$ (newest/oldest approximate gradient) S (a list of the previous m search directions) Y (a list of the previous m approximated gradient differences)
`step` (structure with information about the step management)
s (scaling vector), p (random search direction), `dp` (scaled length of p) $\delta_{\min}$, $\delta_{\max}$ (minimum/maximum norm of trial steps) δ (norm of trial steps), Δ (threshold for good improvement) $\Delta_{\min}$, $\Delta_{\max}$ (minimal/maximal threshold for good improvement) q (cumulative step), r (cumulative gain)
`par` (structure with parameters modified during the search)
T (maximal number of directions in *MLS*), λ (approximate Lipschitz constant) `good` (sufficiently improved function value?), `ss` (are we in *setScale*?) `dir` (direction type), `state` (state of cumulative step) α_E (step-size for extrapolation), $\mathcal{A}$ (list of step-size)
`tune` (structure with fixed parameters for tuning the performance)
$m_{\max} \geq 3$ (maximum number of best points kept) $T_0 \geq m_{\max}$ (maximal number of multi-line searches in *setScale*) $C \geq n$ (maximal number of coordinate directions in *MLS*) $S \geq 1$ (maximal number of random subspace directions in *MLS*) $R \geq 1$ (number of random direction per random subspace direction in *MLS*) $1 \leq E \leq +\infty$ (maximal number of extrapolations of each extrapolation stage in *MLS*) `scSub` (scale random subspace direction?), `scCum` (scale cumulative direction?) `cum` (cumulative step type, 0=none, 1, or 2) $a > 0$ (bound for cumulative step size), $Q > 1$ (factor for adjusting Δ) $0 \leq \Delta_{\min} < 1$, $\Delta_{\max} \geq 0$ (minimal/initial maximal threshold for good improvement)
Continued on next page

$\gamma_\delta > 0$ (factor for finding δ), $\gamma_{\max} > 0$ (factor for adjusting $\Delta_{\max}$)
$\gamma_E > 1$ (factor for extrapolation test), $\gamma_\lambda > 0$ (factor for finding initial λ)
$0 < \gamma_w < 1$, $0 < \gamma_{\text{angle}} < 1$ (tiny parameters for angle condition)
$\delta_{\min} > 0$, $\delta_{\max} > 0$ (minimum/maximum norm of trial steps)

Table 5: Global data structure for the algorithms of the present paper

In the detailed descriptions of *VSBBO*, given below, we use a pseudo-Matlab notation. In particular, the notation $\circ$ denotes componentwise multiplication, == is the comparison operator for equality, `length`(V) computes the length of the vector V, ones$(n, 1)$ generates a $n \times 1$ vector whose entires are 1, zeros$(n, 1)$ generates a $n \times 1$ vector whose entries are zero, `all` determines if all array elements are all nonzero or true, $\sim$ (or `not`) finds logical NOT and $A_{:k}$ denotes the kth column of a matrix A.

In recent years, there has been an increasing interest in finding the best tuning parameters configuration for derivative-free solvers with respect to a benchmark problem set; see, e.g., [1, 37, 38]. In Table 5, there are 6 integral, 2 binary, 1 ternary, and 12 continuous tuning parameters, giving a total of 21 parameters for tuning our algorithm. A small amount of tuning was done by hand. Automatic tuning of *VSBBO* will be considered elsewhere.

2 Line search techniques for BBO

In this section, we describe methods that try to achieve a good decrease in the function value using line searches along specially chosen directions.

Coordinate directions are the coordinate axes e_i, $i = 1, \cdots, n$, in a cyclic fashion.

Limited memory quasi-Newton directions using a finite-differences estimate of the gradient are used to generate useful quadratic models.

Random subspace directions point into the low-dimensional affine subspace spanned by a number of good points kept from previous iterations.

Random directions are used to exploit the fact that stochastic black box optimization methods have a worst case complexity superior to those of deterministic algorithms.

Cumulative directions are based on a simple separable model assumption.

A line search then polls one or more points along the lines in each chosen direction starting at the currently best point. Several such line searches are packaged together into a multi-line search, for which strong probabilistic results can be proved.

The details are chosen in such a way that failure to achieve the desired descent implies that, with high probability, a good bound on the gradient is obtained.

2.1 Probing a direction

First we give a theoretical test that either results in a gain of Δ or more in function value, or gives a small upper bound for the norm of at least one of the gradient encountered.

Assumption (A1) implies that for every $x, p \in \mathbb{R}^n$, we have

$$f(x+p) - f(x) = g(x)^T p + \frac{1}{2}\gamma\|p\|^2, \tag{3}$$

where γ depends on x and p and satisfies one of

$$|\gamma| \le L, \quad \text{(general case)} \tag{4}$$

$$0 \le \gamma \le L, \quad \text{(convex case)} \tag{5}$$

$$0 < \sigma \le \gamma \le L. \quad \text{(strongly convex case)} \tag{6}$$

In all three cases,

$$g(x)^T p - \frac{1}{2}L\|p\|^2 \le f(x+p) - f(x) \le g(x)^T p + \frac{1}{2}L\|p\|^2. \tag{7}$$

Continuity and condition (A2) imply that a minimizer $\widehat{x}$ exists and

$$r_0 := \sup\left\{\|x - \widehat{x}\| \mid f(x) \le f(x_0)\right\} < \infty. \tag{8}$$

(It is enough that this holds with x_0 replaced by some point found during the iteration, which is then taken as x_0).

2.1 Proposition. *Let $x, p \in \mathbb{R}^n$ and $\Delta \ge 0$. Then (A1) implies that*

$$L \ge \frac{|f(x+p) + f(x-p) - 2f(x)|}{\|p\|^2}, \tag{9}$$

and one of the following holds:

(i) $f(x+p) < f(x) - \Delta$,

(ii) $f(x+p) > f(x) + \Delta$ *and* $f(x-p) < f(x) - \Delta$,

(iii) $|g^T p| \le \Delta + \frac{1}{2}L\|p\|^2$.

Proof. Taking the sum of (7) and the formula obtained from it by replacing p with $-p$ gives (9).

Assume that (iii) is violated, so that $\pm g^T p = |g^T p| > \Delta + \frac{1}{2}L\|p\|^2$ for an appropriate choice of the sign. Then by (3) with $\mp p$ in place of p,

$$f(x \mp p) - f(x) \le \mp g(x)^T p + \frac{1}{2}L\|p\|^2 < -\Delta.$$

For the lower sign we conclude that (i) holds. For the upper sign we get the second half of (ii), and the first half follows from $f(x+p) - f(x) \geq g(x)^T p - \frac{1}{2}L\|p\|^2 > \Delta$. □

Proposition 2.1 will play a key role in the construction of our multi-line search *MLS* detailed in Subsection 2.5:

- It presents a lower bound for the Lipschitz constant L which can be used to find reasonable estimates for L.
- If (i) holds, then the step p gives a gain of at least Δ.
- If (ii) holds, then the step $-p$ gives a gain of at least Δ.
- If (iii) holds, possibly none of the steps $\pm p$ give a gain of Δ or more. Instead we have a useful upper bound for the directional derivative.

Care must be taken to ensure that the book-keeping needed for the evaluation of the lower bound for the Lipschitz constant comes out correctly. To ensure this during a line search, we always use x for the best point found, and rescale p such that the next evaluation is always at $x+p$ and a former third evaluation point is at $x-p$. The function values immediately after the next evaluation are then

$$\texttt{fl} := f(x-p), \quad \texttt{fm} := f(x), \quad \texttt{fr} := f(x+p). \tag{10}$$

At this stage, we can compute the lower bound

$$\underline{L} := |\texttt{fl} + \texttt{fr} - 2\texttt{fm}|/\|p\|^2$$

for the Lipschitz constant, valid by (9). Afterwards, whenever $\texttt{fr} < \texttt{fm}$, the best point is updated by overwriting $x+p$ over x, with the consequence that then

$$\texttt{fl} := f(x-2p), \quad \texttt{fm} := f(x-p), \quad \texttt{fr} := f(x). \tag{11}$$

This is used in the cumulative contributions discussed now.

2.2 A cumulative direction

We consider two possibilities to accumulate past directional information into a cumulative search direction:

(i) The first cumulative direction is model independent, computed by $p = x - x_{\text{init}}$, where x is the best point and x_{init} the initial point of the current multi-line search. Here the idea is that many small improvement steps accumulate to a direction pointing from the starting point into a valley, so that more progress can be expected by going further into this cumulative direction.

(ii) The second cumulative direction assumes a separable quadratic model of the form

$$f\Big(x + \sum_{i\in I} \alpha_i p_i\Big) \approx f(x) - \sum_{i\in I} e_i(\alpha_i) \tag{12}$$

with quadratic univariate functions $e_i(\alpha)$ vanishing at $\alpha = 0$. Here I is the set of directions polled at least twice, and p_i is the corresponding direction as rescaled by *MLS*.

By construction, we have for any $i \in I$ three function values at equispaced arguments. We write the quadratic interpolant as

$$f(x + \alpha p) = f - \frac{\alpha}{2} d + \frac{\alpha^2}{2} h = f - e(\alpha),$$

where $e(\alpha) := \dfrac{\alpha}{2}(d - \alpha h)$. If $\texttt{fr} < \texttt{fm}$, the last evaluated point was the best one, so $\texttt{fr} \leq \min(\texttt{fl}, \texttt{fm})$. In this case, (11) holds and we have

$$d := 4\texttt{fm} - 3\texttt{fr} - \texttt{fl}, \quad h := \texttt{fr} + \texttt{fl} - 2\texttt{fm}.$$

Otherwise, the last evaluated point was not the best one, so $\texttt{fm} \leq \min(\texttt{fl}, \texttt{fr})$. In this case, (10) holds and we have

$$d := \texttt{fl} - \texttt{fr}, \quad h := \texttt{fr} + \texttt{fl} - 2\texttt{fm}.$$

The minimizer of the quadratic interpolant restricted to the interval $[-a, a]$ is

$$\alpha = \begin{cases} a & \text{if } d \geq 0, \\ -a & \text{if } d < 0 \end{cases}$$

in case $h \leq 0$. Otherwise, we have

$$\alpha = \begin{cases} \min(a, d/2h) & \text{if } d \geq 0, \\ \max(-a, d/2h) & \text{if } d < 0. \end{cases}$$

Assuming the validity of the quadratic model (12), we find the model optimizer by additively accumulating the estimated steps αp and gains e into a cumulative step q with anticipated gain r, by the following algorithm:

2.2 Algorithm. cumulative update (updateCum)

Purpose: Update the cumulative direction
function [step] = **updateCum**(point, step, tune);

if ($\texttt{fr} < \texttt{fm}$), $d = 4\texttt{fm} - 3\texttt{fr} - \texttt{fl}$; **else** $d = \texttt{fr} - \texttt{fl}$; **end**;
$h = \texttt{fr} + \texttt{fl} - 2\texttt{fm}$;
if ($h \leq 0$),
  if ($d \geq 0$), $\alpha = a$; **else** $\alpha = -a$; **end**;
else
  if ($d \geq 0$), $\alpha = \min(a, d/2h)$; **else** $\alpha = \max(-a, d/2h)$; **end**;
end;
$q = q + \alpha p$; $r = r + 0.5\alpha(d - \alpha h)$; % update q and r

2.3 Finite difference L-BFGS direction

Finite difference quasi-Newton methods estimate the gradient with components

$$\widehat{g}_i := \frac{f(x+\alpha e_i) - f(x)}{\alpha},$$

where e_i is the ith coordinate vector. The most popular choice for α is

$$\alpha := \max\{1, \|x\|_\infty\}\sqrt{\varepsilon_m}$$

where ε_m is the machine precision; another choice for α is made in Section 2.5. Then the Hessian approximation is computed by quasi-Newton methods, cf. [36].

Here is given how to compute the L-BFGS direction [36] by the following algorithm:

2.3 Algorithm. L-BFGS direction (lbfgsDir)

Purpose: Generate L-BFGS direction
function [step] = **lbfgsDir**(point, step);
for $(i = 1:m)$, $\rho_i = 1/(Y_{:i}^T S_{:i})$; **end**; $\overline{q} = \text{zeros}(n, m+1)$; $\overline{r} = \text{zeros}(n,1)$; $\overline{\alpha} = \text{zeros}(m,1)$; $\beta = \text{zeros}(m,1)$; $\overline{q}_{:m+1} = \widehat{g}$; **for** $(i = m:-1:1)$, $\overline{\alpha}_i = \rho_i S_{:i}^T \overline{q}_{:i+1}$; $\overline{q}_{:i} = \overline{q}_{:i+1} - \overline{\alpha}_i Y_{:i}$; **end**; $\overline{r} = \texttt{dxg} * \overline{q}_{:1}$; **for** $(i = 1:m)$, $\beta_i = \rho_i Y_{:i}^T \overline{r}$; $\overline{r} = \overline{r} + S_{:i}(\overline{\alpha}_i - \beta_i)$; **end**; $p = -\overline{r}$;

Due to rounding error, the computed direction by *lbfgsDir* may not satisfy the angle condition. From [29], we call *enforceAngle* to enforce the angle condition.

2.4 Algorithm. (**enforceAngle**)

Purpose: Enforce the angle condition
function [step] = **enforceAngle**(point, step, tune);

$\kappa = \widehat{g}^T p$;
if $(\kappa > 0)$, act $= \{i \mid \widehat{g}_i p_i > 0\}$; $p_{\texttt{act}} = -p_{\texttt{act}}$; $\kappa = -\kappa$; **end**;
$\kappa_1 = \widehat{g}^T \widehat{g}$; $\kappa_2 = p^T p$; $\kappa_3 = \kappa_1 \kappa_2$; $\overline{\kappa} = \kappa/\sqrt{\kappa_3}$;
if $(\overline{\kappa} \geq -\gamma_{\text{angle}})$
  $w = (\kappa_3 \max(\gamma_w, 1 - \overline{\kappa}^2))/(1 - \gamma_{\text{angle}}^2)$; $t = (\kappa + \gamma_{\text{angle}}\sqrt{w})/\kappa_1$;
  if $(w > 0$ & $t \neq \pm\infty)$, $p = p - t\widehat{g}$; **else**, $p = -\widehat{g}$; **end**;
end;

2.4 Choice of search direction

The following algorithm generates 5 kinds of direction vectors: coordinate directions (`dir` = 1), limited memory quasi-Newton directions (`dir` = 2), random subspace directions (`dir` = 3), random directions (`dir` = 4), and cumulative directions (`dir` = 5).

The scaling of the search directions to norm δ may be done with the intention to approximately minimize the final bound $\sqrt{cn}\Gamma(\delta)$ for the gradient norm in (16) below. For fixed Δ, the scale-dependent factor $\Gamma(\delta) = L\delta + 2\Delta/\delta$ (see (17) below) is smallest for the choice

$$\widehat{\delta} = \sqrt{2\Delta/L}. \tag{13}$$

However, in practice, L is unknown and we replace it by the approximation λ from *MLS*. Moreover, we safeguard δ by enforcing sensible positive lower and upper bounds.

2.5 Algorithm. Direction generator (direction)

Purpose: Create random subspace, coordinate, quasi-Newton, random, or cumulative direction

function [`step`, `par`] = **direction**(`point`, `step`, `par`, `tune`);

switch `dir`
case 1 % coordinate direction p
  $p = 1$; `scale` $= 0$;
case 2 % scaling and limited memory quasi Newton direction p
  if $(m == 0)$, $p = -\widehat{g}/\|\widehat{g}\|$; **else**, *lbfgsDir*; *enforceAngle*; **end**;
  `scale`=0;
case 3 % random subspace direction p
  $\alpha = \text{rand}(m-1, 1) - 0.5$; $\alpha = \alpha/\|\alpha\|$; $p = \sum_{i=1, i\neq b}^{m} \alpha_i (X_{:i} - X_{:b})$; `scale`=`ScSub`;
case 4 % random direction p
  $p = \text{rand}(n, 1) - 0.5$; `scale`=1;
case 5 % cumulative direction p
  $p = q$; `scale`=`scCum`;
end;
if `scale` % scale to $\|p\| = \delta$
  $\delta = \max\left(\delta_{\min}, \min\left(\sqrt{\gamma_\delta \Delta/\lambda}, \delta_{\max}\right)\right)$; $p = s \circ p(\delta/\|p\|)$; `dp`=$\delta$;
else % evaluate $\|p\|$
  `dp`=$\|p\|$;
end;

The coordinate direction values enhances the global search properties, decreasing on average with the number of function evaluations used.

The quasi-Newton direction to be computed needs the matrix S whose columns are the previous m search directions and the matrix Y whose columns are the previous m estimated gradient differences. At first, $m = 0$ and then increases up to a maximum of $m_{\max}$. Here is

given how to update both S and Y as follows:

2.6 Algorithm. quasi Newton update (updateSY)

Purpose: Update quasi-Newton information
function [point] = **updateSY**(point, tune);
dx = xm − $\texttt{xm}_{\text{old}}$; dg = $\widehat{g} - \widehat{g}_{\text{old}}$; dxg = $\texttt{dx}^T\texttt{dg}/\texttt{dg}^T\texttt{dg}$; **if** ($m <$ mmax), $m = m + 1$; **else**, $m = 1$; **end**; $S_{:m} = \texttt{dx}$; $Y_{:m} = \texttt{dg}$;

The subspace direction generated by *direction* with par.dir = 3 requires to keep a matrix X whose columns are the m best points and a vector F with their function values. Initially $m = 1$, to be increased up to a maximum of $m_{\max}$; later we overwrite each time the worst point. The following algorithm updates both X and F:

2.7 Algorithm. subspace update (updateXF)

Purpose: Update subspace information
function [point] = **updateXF**(point, tune);
if ($m == m_{\max}$), $[f_{\mathbf{w}}, i_{\mathbf{w}}] = \max(F)$; % find worst point **else** $m = m + 1$; $i_{\mathbf{w}} = m$; **end** $X_{:i_{\mathbf{w}}} = \texttt{xm}$; $F_{i_{\mathbf{w}}} = \texttt{fm}$; $b = i_{\mathbf{w}}$; % select the index of best point

2.8 Proposition. *For the random search direction generated by direction with* dir = 4, *the output p of direction satisfies $\|p\| = \delta$ and, with probability $\geq \frac{1}{2}$,*

$$\|g(x)\|_*\|p\| \leq 2\sqrt{cn}|g(x)^T p| \tag{14}$$

with a positive constant $c \approx 4/7$.

Proof. Define $\overline{p}_i := p_i/s_i$ and $\overline{g}_i := s_i g_i$. Then by (2), $g^T p = \overline{g}^T\overline{p}$ and $\|g\|_* = \|\overline{g}\|_2$ and $\|p\| = \|\overline{p}\|_2$; so the results of Appendix A apply with $4c = c_0 \approx 16/7$. □

2.5 A multi-line search

The following multi-line search algorithm *MLS* polls in T suitable directions (defined by *direction*) in a line search fashion a few objective function values each in the hope of reducing the objective by more than Δ.

Schematically, *MLS* works as follows:

(i) At first, at most C iterations with coordinate directions are used.

(ii) Then, the finite differences L-BFGS direction is used only once.

(iii) Next, except in the final iteration, at most S iterations with subspace directions are used.

(iv) After generating $T-1$ directions without sufficient improvement of the function value, a cumulative direction is used as final, Tth direction in the hope of finding a model-based gain.

(v) For each direction generated, a line search is performed where the following happens:

- A step in the current direction is tried.
- If a large gain is found, a sequence of extrapolations is tried.
- If sufficient negative gain was found, a step in the opposite direction is tried.
- If a large gain is found in the opposite direction, a sequence of extrapolations is tried.

(iv) Once the algorithm has found an improvement of the function value of more than Δ it ends after completion of the current line search.

MLS also updates an approximation λ for the Lipschitz constant L of the objective function. Proposition 2.1 implies that

$$\lambda_0 \leq \lambda \leq \max(\lambda_0, L) \leq \lambda_0 + L, \tag{15}$$

where λ_0 is the initial value of λ. Moreover, after generating each coordinate search direction by *direction* it estimates each component of gradient in a way that is little different than the forward finite difference approach.

2.9 Algorithm. A multi-line search (MLS)

Purpose: Improve function value along multiple directions

function [point, par] = **MLS**(fun, point, step, par, tune);

$\mathtt{xm} = X_{:b}$; $\mathtt{fm} = F(b)$; $x_{\text{init}} = \mathtt{xm}$; $f_{\text{init}} = \mathtt{fm}$; $r = 0$; $q = \text{zeros}(n, 1)$; $t = 0$; $\mathtt{state} = -1$; $\mathtt{good} = 0$; $\mathtt{nf} = 0$;
while $(t < T)$,
  switch state
  case -1 % new direction
    $t = t + 1$; $\mathtt{nE} = 0$; $\alpha_E = \mathcal{A}_t$;
    if $(t \le C)$, $\mathtt{dir} = 1$; % pick coordinate step
    elseif $(t == C + 1)$, $\mathtt{dir} = 2$; % pick finite difference L-BFGS step
    elseif $(t \le C + S + 1)$, $\mathtt{dir} = 3$; % pick random subspace step
    elseif $(t < T)$, $\mathtt{dir} = 4$; % pick random step
    else % check for cumulative step
      if $(\mathtt{cum} == 1)$, $q = x - x_{\text{init}}$; % first cumulative step
      elseif $(\mathtt{cum} == 2$ & $r \ge \Delta)$, % second cumulative step
      end;
      if $(\mathtt{cum} > 0$ & $\mathtt{all}(q == 0))$, $\mathtt{dir} = 4$; % pick random step
      else, $\mathtt{dir} = 5$; % pick cumulative step
        if $(\mathtt{cum} == 1)$, $\mathtt{state} = 1$; $\mathtt{fl} = f_{\text{init}}$; **end**; % use initial value
      end;
    end;
    direction; % generate direction
  case 0, % extrapolate stage
    $\mathtt{fe} = \mathtt{fr}$; $\mathtt{nE} = \mathtt{nE} + 1$;
    if $(\mathtt{nE} == 1)$, $\mathtt{fl} = \mathtt{fm}$; **else**, $\alpha_E = \gamma_E \alpha_E$; **end**;
  case 1, % opposite direction
    $p = -p$; $\mathtt{fl} = \mathtt{fr}$; $\alpha_E = \mathcal{A}_t$;
  end;
  if $(\mathtt{dir} == 1)$, $(\mathtt{xr})_t = (\mathtt{xm})_t + \alpha_E p$; **else**, $\mathtt{xr} = \mathtt{xm} + \alpha_E p$; **end**;
  $\mathtt{fr} = \mathtt{fun}(\mathtt{xr})$; $\mathtt{nf} = \mathtt{nf} + 1$; $\mathtt{df} = \mathtt{fm} - \mathtt{fr}$;
  if $(\mathtt{dir} == 1$ & $\mathtt{state} == -1)$, $\widehat{g}_{t:} = (\mathtt{fr} - f_{\text{init}})/\alpha_E$; **end**;
  if $(\mathtt{nE} == 1)$, $\lambda = \max(\lambda, |\mathtt{fl} + \mathtt{fr} - 2\mathtt{fm}|/\mathtt{dp}^2)$; **end**;
  if $(\mathtt{cum} == 2)$, *updateCum*; **end**;
  $\mathtt{ext} = (\mathtt{df} > \alpha_E \Delta)$; % large gain; extrapolate
  $\mathtt{sext} = (\mathtt{nE} < E$ & $\mathtt{ext})$; % sequence large gain; sequence extrapolate
  if sext, $\mathtt{state} = 0$; **continue**; % with extrapolation stage
  else
    opp=$(\mathtt{state} < 0)$; % opposite gain expected
    if opp, $\mathtt{state} = 1$; **continue**;
    if $(\mathtt{state}==0)$, % the end of extrapolation stage
      $\mathtt{good} = 1$; $\alpha_E = \alpha_E/\gamma_E$; $\mathcal{A}_t = \alpha_E$; $\mathtt{fm} = \mathtt{fe}$;
      if $(\mathtt{dir} == 1)$, $(\mathtt{xm})_t = (\mathtt{xm})_t + \alpha_E p$; $(\mathtt{xr})_t = (\mathtt{xm})_t$; **else**, $\mathtt{xm} = \mathtt{xm} + \alpha_E p$; **end**;
    else
      $\mathcal{A}_t = \mathcal{A}_t/\gamma_E$;
      if $(\mathtt{dir} == 1)$, $(\mathtt{xr})_t = (\mathtt{xm})_t$; **end**;
    end
    $\mathtt{state} = -1$; % line search completed
  end;
end;

We now prove that one obtains either a gain of Δ or, with high probability, an upper bound of $\|g\|_*$ for at least one of the gradients encountered.

2.10 Theorem. *Assume that (A1) holds and let* `nf` *be the counter for the number of function evaluations and* Δ_f *be the improvement on the function value in MLS.*

(i) f *decreases by at least*

$$\Delta \max(\texttt{nf} - 2T - 1, 0).$$

(ii) Suppose that $0 < \eta < 1$ *and* $T \geq 2 + C + S + \log_2 \eta$. *If* f *did not decrease by more than* Δ *then, with probability* $\geq 1 - \eta$, *the original point or one of the points evaluated with better function values has a gradient* g *with*

$$\|g\|_* \leq \sqrt{cn}\Gamma(\delta), \tag{16}$$

where c *is the constant in Proposition 2.8 and*

$$\Gamma(\delta) := L\delta + \frac{2\Delta}{\delta}. \tag{17}$$

Proof. (i) In the while loop of *MLS*, a direction p is generated and at most two function values are computed, unless an extrapolation step is performed. In this case, at most $\texttt{nf} - 2T - 1$ additional function values are computed during the extrapolation stage, and the loop is ended.

As a result, if $\Delta_f < \Delta$, then f did not decrease by more than Δ. Otherwise, $\Delta_f \geq (\texttt{nf} - 2T - 1)\Delta$; *MLS* uses at least $2T + 1$ function evaluations. Clearly, the function value of the best point does not increase.

(ii) Assume that f did not decrease by more Δ. For $t = 1, \ldots, T$, let p_t be the tth search direction generated by *MLS*, and let x_t be the best point obtained before searching in direction p_t. Then, Proposition 2.1 gives

$$|g(x_t)^T p_t| \leq \Delta + \frac{L}{2}\|p_t\|^2 = \Delta + \frac{L}{2}\delta^2 = \frac{\delta}{2}\Gamma(\delta)$$

for $t = 1, \ldots, T$. Let $\mathcal{R} := \{C + S + 1 < t < T\}$. For $t \in \mathcal{R}$, the search direction was generated by *direction* as a random direction, hence Proposition 2.8 implies that for $t \in \mathcal{R}$,

$$\|g(x_t)\|_* = \|g(x_t)\|_*\|p_t\|/\delta \leq 2\sqrt{cn}|g(x_t)^T p_t| \leq \sqrt{cn}\Gamma(\delta)$$

holds with probability $\frac{1}{2}$ or more. Therefore $\|g(x_t)\|_* \leq \sqrt{cn}\Gamma(\delta)$ fails with a probability $p_t < \frac{1}{2}$ for any fixed $t \in \mathcal{R}$. Therefore, the probability p that (16) holds for at least one of the gradients $g = g(x_t)$ $(t \in \mathcal{R})$ is $p = 1 - \prod_{t=1}^{T-1} p_t \geq 1 - 2^{2+C+S-T}$. □

3 A stochastic descent algorithm for BBO

3.1 Setting the scales

Our stochastic algorithm is sensitive to the choice of the maximal desired gain $\Delta_{\max}$ and the initial choice of the approximate Lipschitz constant λ. The following algorithm gives

practically useful heuristics for choosing maximal desired gain $\Delta_{\max}$ and for initializing λ. At the same time, it is used to estimate a sensible scaling vector s. The algorithm performs T_0 iterations of *MLS*, updating both X and F by *updateXF* and S and Y by *updateSY*. It estimates the scaling vector s from information stored in X and $\Delta_{\max}$ and λ by from information stored in X and F.

3.1 Algorithm. Setting the scales (setScales)

Purpose: Estimate $\Delta_{\max}$, initial λ and s

function [`point`, `step`, `par`] = **setScales** (`fun`, `point`, `step`, `par`, `tune`);

% initialization

% get point

$n = \text{length}(x)$; $f = \texttt{fun}(x)$; $X = x$; $F = f$; $b = 1$; $m = 1$;

$m = 0$; $S = \text{zeros}(n, m)$; $Y = \text{zeros}(n, m)$;

% get step

$\Delta = \Delta_{\max}$; $\delta = \delta_{\max}$; $s = \text{ones}(n, 1)$;

% get par

$\lambda = 0$; $T = C + S + R + 2$; $\texttt{ss} = 0$; $\mathcal{A} = \text{ones}(T, 1)$;

% stage 1: no scaling vector

for $(t = 1 : T_0)$,

  if $(t \geq 2)$, $\widehat{g}_{\text{old}} = \widehat{g}$; $\texttt{xm}_{\text{old}} = \texttt{xm}$; **end**;

  MLS; *updateXF*;

  if $(t \geq 2)$, *updateSY*; **end**;

end;

% stage 2: estimate s, $\Delta_{\max}$ and initial λ

% get s

$\texttt{ss} = 1$; % use scaling vector for $\texttt{dir} = 4$

for $(i = 1 : m)$, $\texttt{dX}_{:i} = X_{:i} - X_{:b}$; **end**;

$s = \sup_{i=1:m} (\texttt{dX}_{:i})$; $s(s == 0) = 1$;

% get maximal desired gain $\Delta_{\max}$ and λ

$\texttt{dF} = \text{median}_{i=1:m} |F_i - F_b|$;

if $(\texttt{dF} == 0)$

  $\Delta_{\max} = 0$; $\Delta = \Delta_{\max}$;

  if $(\lambda == 0)$, $\lambda = \gamma_\lambda / \sqrt{n}$; **end**;

else,

  $\Delta_{\max} = \gamma_{\max} \min(\texttt{dF}, 1)$; $\Delta = \Delta_{\max}$;

  if $(\lambda == 0)$, $\lambda = \gamma_\lambda \sqrt{\texttt{dF}/n}$; **end**;

end;

3.2 Probing for fixed decrease

Based on the preceding results, we present a fixed decrease search algorithm whose goal is to use repeated calls to the multi-line search *MLS* in order to improve the function value each time by a fixed amount Δ.

3.2 Algorithm. Fixed Decrease Search (FDS)

Purpose: Repeatedly improve function value by $> \Delta$
function [point] = **FDS**(fun, point, step, par, tune);
while 1,
$\widehat{g}_{\text{old}} = \widehat{g}$; $\texttt{xm}_{\text{old}} = \texttt{xm}$; *MLS*;
if ($\sim$ good), **break**; **end**;
updateXF; *updateSY*;
end;

3.3 Theorem. *Assume that (A1) and (A2) hold and let Δ_f, N, f_0 and $\widehat{f}$ be the improvement on the function value, the number of iteations of FDS, the initial value of f and the minimal function value, repectively. Then:*

(i) The number of function evaluations of FDS is bounded by

$$(2T+1)\min(\frac{\Delta_f}{\Delta},1)N \le (2T+1)N \le (2T+1)\frac{f_0-\widehat{f}}{\Delta}.$$

(ii) Let $\eta_1 := 2^{-(T-C-S-2)}$. Then, with probability $\ge 1-\eta_1$,

$$\|g\|_* \le \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta_{\max}}\Big) \tag{18}$$

for at least one of the gradients g encountered. Here c is the constant from Proposition 2.8 and, if λ_0 denotes the value of λ before the first execution of FDS,

$$L' := \frac{L^2\gamma_\delta}{\lambda_0} + 4L + 4\frac{\lambda_0+L}{\gamma_\delta}. \tag{19}$$

Proof. Because of (A2), $\widehat{f}$ is finite. Denote by f_k the result of the kth execution of *FDS*. By the definition of good, $f_k \le f_{k-1} - \Delta$ for all but the last value of k. We conclude that $\widehat{f} \le f_N \le f_0 - N\Delta$, so that $N \le (f_0 - \widehat{f})/\Delta$. If $\Delta_f < \Delta$, then each iteration uses at most $2T+1$ function evaluations. Otherwise, it uses at most $(2T+1)\Delta_f/\Delta$ function evaluations since $\Delta_f \ge (\texttt{nf} - 2T - 1)\Delta$, where nf is the number of calls of *MLS*; hence (i) follows.

(ii) By Theorem 2.10, $\|g\|_* \le \sqrt{cn}\Gamma(\delta)$ holds with probability $\ge 1-\eta_1$. Thus it is sufficient to show that

$$\Gamma(\delta) \le L\delta_{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta_{\max}}. \tag{20}$$

By the definition of δ in *direction*, we have one of the following three cases:

CASE 1: $\delta = \sqrt{\frac{\gamma_\delta \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta + \frac{2\Delta}{\delta} = L\sqrt{\frac{\gamma_\delta \Delta}{\lambda}} + 2\sqrt{\frac{\lambda\Delta}{\gamma_\delta}} = \Lambda\sqrt{\Delta},$$

where

$$\Lambda := L\sqrt{\frac{\gamma_\delta}{\lambda}} + 2\sqrt{\frac{\lambda}{\gamma_\delta}}. \tag{21}$$

CASE 2: $\delta = \delta_{\min} \geq \sqrt{\frac{\gamma_\delta \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta_{\min} + \frac{2\Delta}{\delta_{\min}} \leq L\delta_{\min} + 2\sqrt{\frac{\lambda\Delta}{\gamma_\delta}} \leq L\delta_{\min} + \Lambda\sqrt{\Delta}.$$

CASE 3: $\delta = \delta_{\max} \leq \sqrt{\frac{\gamma_\delta \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta_{\max} + \frac{2\Delta}{\delta_{\max}} \leq L\sqrt{\frac{\gamma_\delta \Delta}{\lambda}} + \frac{2\Delta}{\delta_{\max}} \leq \Lambda\sqrt{\Delta} + \frac{2\Delta}{\delta_{\max}}.$$

Thus in each case,

$$\Gamma(\delta) \leq L\delta_{\min} + \Lambda\sqrt{\Delta} + \frac{2\Delta}{\delta_{\max}}.$$

Now (20) follows since by (15) and (21), $\Lambda^2 = \frac{L^2\gamma_\delta}{\lambda} + 4L + \frac{4\lambda}{\gamma_\delta} \leq L'$. □

3.3 The *VSBBO* algorithm

We now have all ingredients to formulate the *Vienna stochastic black box optimization* algorithm *VSBBO*. It uses in each iteration the fixed decrease search algorithm to update the best point. If no progress is made in the corresponding *FDS* call, Δ is reduced by a factor of Q. Once Δ is below a minimum threshold, the algorithm stops.

3.4 Algorithm. Vienna stochastic black box optimization (VSBBO)

Purpose: Solve stochastic black box optimization
function $[x,f]$ = **VSBBO**(`fun`, x, `tune`)
setScales; **while** 1,
Continued on next page

FDS;
% gain of Δ unlikely with step of length δ
if $(\Delta \le \Delta_{\min})$, **break**; **end**;
$\Delta = \Delta/Q$; % reduce Δ
end;

In Section 4, upper bounds are obtained for the total number of function evaluations of *VSBBO* for the nonconvex, convex, and strictly convex case.

Appendix 5 provides numerical results, comparing *VSBBO* with the best previous black box algorithms on a large benchmark of problems in low and high dimensions.

4 Complexity analysis of VSBBO

We now prove the complexity results for nonconvex, convex and strongly convex objective function.

4.1 The general (nonconvex) case

4.1 Proposition. *Assume that (A1) and (A2) hold and let f_0 be the initial and $\widehat{f}$ the optimal function value. If $\Delta_{\min} > 0$ then the number of function evaluations needed up to iteration k by VSBBO, started at x^0, is*

$$n_k < (2T+1)\Big(T_0 + \frac{Q(f_0 - \widehat{f})}{(Q-1)\Delta_{\min}}\Big).$$

Proof. In *setScale*, we need $n_0 < (2T+1)T_0$ function evaluations. By construction, the kth call to *FDS* uses $\Delta = Q^{1-k}\Delta_{\max}$, hence uses by Theorem 3.3(i) at most

$$(2T+1)\frac{f_0 - \widehat{f}}{Q^{1-k}\Delta_{\max}}$$

function evalutions. Then, the total number of function evaluations up to iteration k is

$$\begin{aligned} n_k &\le (2T+1)T_0 + \sum_{j=1}^{k}(2T+1)\frac{f_0 - \widehat{f}}{Q^{1-k}\Delta_{\max}} \\ &= (2T+1)\Big(T_0 + \frac{Q(f_0 - \widehat{f})}{(Q-1)\Delta_{\min}}\Big) \end{aligned}$$

since $\Delta_{\max} \ge Q^{k-1}\Delta_{\min}$. □

4.2 Theorem. *Assume that (A1) and (A2) hold. With c from Proposition 2.8, let $\zeta \geq 9c$, $0 < \eta < 1$, and let K be a positive integer. Then, for any $\varepsilon > 0$, if*

$$T \geq C + S + 2 + \log_2 \frac{K+1}{\eta}, \tag{22}$$

$$\max\left(Q^{1-K}\Delta_{\max}, \frac{\varepsilon^2}{\zeta L' n}\right) \leq \Delta_{\min} \leq \frac{\varepsilon^2}{9cL'n}, \tag{23}$$

$$\delta_{\min} \leq \frac{\varepsilon}{3L\sqrt{cn}}, \quad \delta_{\max} \geq \frac{2\varepsilon}{3L'\sqrt{cn}}, \tag{24}$$

Algorithm 3.4 finds after at most $\mathcal{O}(n\varepsilon^{-2})$ function evaluations with probability $\geq 1-\eta$ a point x with

$$\|g(x)\|_* \leq \varepsilon. \tag{25}$$

Proof. By the rule of updating Δ in *VSBBO*, $\Delta_k = Q^{1-k}\Delta_{\max} \leq \Delta_{\min}$ for $k \geq K$. Hence at most K steps of *FDS* are performed. By (22), we have $\eta_1 = 2^{-(T-C-S-2)} \leq \eta/(K+1)$. Thus by Theorem 3.3(ii), we have, with probability $\geq 1-(K+1)\eta_1 \geq 1-\eta$, for at least one of the gradients g encountered,

$$\|g\|_* \leq \min_{j=0:K} \Gamma(\delta_j) \leq \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\Delta_{\min}} + \frac{2\Delta_{\min}}{\delta_{\max}}\Big) \leq \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} = \varepsilon. \tag{26}$$

Here we used (23) and (24). Moreover, by Proposition 4.1 and (23),

$$\begin{aligned} n_K &\leq \Big(T_0 + \frac{Q(f_0 - \widehat{f})}{(Q-1)\Delta_{\min}}\Big) \\ &\leq (2T+1)\Big(T_0 + \frac{\zeta L' n Q}{(Q-1)\varepsilon^2}(f_0 - \widehat{f})\Big) = \mathcal{O}\Big(n\varepsilon^{-2}\Big). \end{aligned}$$

□

In the limiting case $\Delta_{\min} = \delta_{\min} = 0$, we obtain the following convergence result:

4.3 Corollary. *Suppose that (A1) holds. Let f_k be the function value at the end of the kth iteration of VSBBO with $\Delta_{\min} = \delta_{\min} = 0$. Then*

$$f_k \to -\infty \quad \textit{or} \quad \inf_k \|g_k\|_* = 0.$$

Proof. Let $\mathcal{S}$ and $\mathcal{U}$ be the set of the successful and unsuccessful iterations generated by Algorithm 3.4, respectively. We may assume that f_k is bounded below, and have to show that $\lim_{k\to\infty} \Delta_k = 0$.

If the number of iterations is finite then $\mathcal{S}$ is finite. Let $k_0 \in \mathcal{S}$ be maximal. Then the updating rule for Δ_k implies that $\Delta_k = \Delta_{k_0}/Q^{k-k_0} \to 0$ as $k \to \infty$. If the number of iterations is not finite then $\mathcal{S}$ and $\mathcal{U}$ are infinite, and again $\Delta_k \to 0$ as $k \to \infty$. Theorem 3.3(ii) now implies that $\inf_k \|g_k\|_* = 0$. □

4.2 The convex case

4.4 Theorem. *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. With c from Proposition 2.8, let $\zeta \geq 9c$, $0 < \eta < 1$, and let K be a positive integer. For any $\varepsilon > 0$, if (22)–(24) hold then VSBBO finds after at most $\mathcal{O}\big(n\varepsilon^{-1}\big)$ function evaluations with probability $\geq 1-\eta$ a point x with*

$$\|g\|_* \leq \varepsilon, \quad f - \widehat{f} \leq \varepsilon r_0, \tag{27}$$

where r_0 is given by (8).

Proof. By (A2), f has a minimizer $\widehat{x}$ and $r_0 < \infty$. By Theorem 4.2, at most K steps of *FDS* are performed. Let f_{k-1} be the results of the $(k-1)$th execution of *VSBBO* satisfying (25); hence $k-1 \leq K$. The convex case is characterized by (5), so that

$$\widehat{f} \geq f_{k-1} + g_{k-1}^T(\widehat{x} - x_{k-1}).$$

We know from Theorem 4.2 that, with probability $\geq 1-\eta$, (25) holds and hence

$$f_{k-1} - \widehat{f} \leq g_{k-1}^T(x_{k-1} - \widehat{x}) \leq \|g_{k-1}\|_* \|x_{k-1} - \widehat{x}\| \leq \varepsilon r_0 \tag{28}$$

by (8). (27) now follows from (25) and (28). Let f_k^j $(j = 0, \ldots, N_k + 1)$ be the finite sequence of function values generated by performing $N_k + 1$ steps of *MLS* at kth execution of *FDS*, so that

$$f_k \leq f_k^{N_k} - \Delta_k \leq f_k^{N_k - 1} - 2\Delta_k \leq \ldots \leq f_{k-1} - N_k\Delta_k, \tag{29}$$

where $f_{k-1} := f_k^0$ and $f_k := f_k^{N_k+1}$. From (28), (29) and the rule of updating Δ in *VSBBO*, we find

$$0 \leq f_k - \widehat{f} \leq f_{k-1} - \widehat{f} - N_k\Delta_k \leq \varepsilon r_0 - N_k\Delta_k = \varepsilon r_0 - N_k Q^{k-1}\Delta_{\min},$$

leading to

$$N_k \leq Q^{1-k}\frac{\varepsilon r_0}{\Delta_{\min}} \leq Q^{1-k}\frac{r_0 c\zeta L' n}{\varepsilon} \tag{30}$$

by (23). The bound for the number of function evaluations is now obtained from Theorem 3.3 and (30):

$$\begin{aligned} n_K &\leq (2T+1)\Big(T_0 + \sum_{j=1}^{K} N_j\Big) \leq (2T+1)\Big(T_0 + \frac{r_0 c\zeta L' n}{\varepsilon}\sum_{j=1}^{K} Q^{1-j}\Big) \\ &\leq (2T+1)\Big(T_0 + \frac{Q r_0 c\zeta L' n}{\varepsilon(Q-1)}\Big) = \mathcal{O}\big(n\varepsilon^{-1}\big). \end{aligned}$$

□

4.3 The strongly convex case

4.5 Theorem. *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. Under the assumptions of Theorem 4.2, VSBBO finds after at most $\mathcal{O}\big(n\log\varepsilon^{-1}\big)$ function evaluations with probability $\geq 1-\eta$ a point x with*

$$\|g\|_* \leq \varepsilon, \quad f-\widehat{f} \leq \frac{\varepsilon^2}{2\sigma}, \quad \|x-\widehat{x}\| \leq \frac{\varepsilon}{\sigma^2}. \tag{31}$$

Proof. By Theorem 4.3, at most K steps of *FDS* are performed. Let f_{k-1} be the results of the $(k-1)$th execution of *VSBBO* satisfying (25); hence $k-1 \leq K$. The strongly convex case is characterized by (6), so that f has a minimizer $\widehat{x}$ and

$$f(y) \geq f(x) + g(x)^T(y-x) + \frac{1}{2}\sigma\|y-x\|^2$$

for any x and y in $\mathcal{L}(x_0)$. For fixed x, the right-hand side of this inequality is a convex quadratic function of y, minimal when its gradient vanishes. By (2), this is the case iff y_i takes the value $x_i - \dfrac{s_i}{\sigma}g_i(x)$ for $i=1,\cdots,n$, and we conclude that $f(y) \geq f(x) - \dfrac{1}{2\sigma}\|g(x)\|_*^2$ for $y \in \mathcal{L}(x_0)$. Therefore

$$\widehat{f} \geq f(x) - \frac{1}{2\sigma}\|g(x)\|_*^2. \tag{32}$$

The replacement of x by x_{k-1} in (32) and (25) gives, with probability $\geq 1-\eta$,

$$f_{k-1} - \widehat{f} \leq \frac{\|g_{k-1}\|_*^2}{2\sigma} \leq \frac{\varepsilon^2}{2\sigma}. \tag{33}$$

Since the gradient vanishes at the optimal point, we get from Theorem 4.2 and (33)

$$\|\widehat{x} - x_{k-1}\|^2 \leq \frac{2}{\sigma}(f_{k-1}-\widehat{f}) \leq \frac{\varepsilon}{\sigma^2} \tag{34}$$

with probability $\geq 1-\eta$. By the rule of updating Δ in *VSBBO*, we may use (29), where N_k+1 is the number of steps performed by *MLS* at the kth execution of *FDS*. By (33),

$$0 \leq f_k - \widehat{f} \leq f_{k-1} - \widehat{f} - N_k\Delta \leq \frac{\varepsilon^2}{2\sigma} - N_k\Delta \leq \frac{\varepsilon^2}{2\sigma} - N_k\Delta_{\min},$$

so that

$$N_k \leq \frac{\varepsilon^2}{2\sigma\Delta_{\min}} \leq \frac{c\zeta L'n}{2\sigma} \tag{35}$$

by (23). Now Theorem 3.3 implies

$$\begin{aligned} n_K &\leq (2T+1)\Big(T_0 + \sum_{j=1}^{K} N_j\Big) \leq (2T+1)\Big(T_0 + \frac{c\zeta L'n}{2\sigma}K\Big) \\ &= (2T+1)\Big(T_0 + \frac{c\zeta L'n}{2\sigma}\Big(1+\log_Q\frac{\Delta_{\max}}{\Delta_{\min}}\Big)\Big) \\ &= (2T+1)\Big(T_0 + \frac{c\zeta L'n}{2\sigma}\Big(1+\log_Q\frac{c\zeta L'n\Delta_{\max}}{\varepsilon^2}\Big)\Big) = \mathcal{O}\big(n\log\varepsilon^{-1}\big). \end{aligned}$$

□

5 Numerical results

In this section we compare our new solver with other state-of-the-art solvers on a large public benchmark.

5.1 Test problems used

VSBBO was compared with other codes from the literature on all 549 unconstrained problems from the `CUTEst` [19] collection of test problems for optimization with up to 5000 variables, in case of variable dimension problems for all allowed dimensions in this range. Figure 1 shows that a total of 525 such test problems were solved by at least one solver. To avoid guessing the solution of toy problems with a simple solution (such as all zero or all one), we shifted the arguments, for all $i = 1, \ldots, n$, by

$$x_i = (-1)^{i-1} \frac{m}{m+i},$$

where $m = 2$.

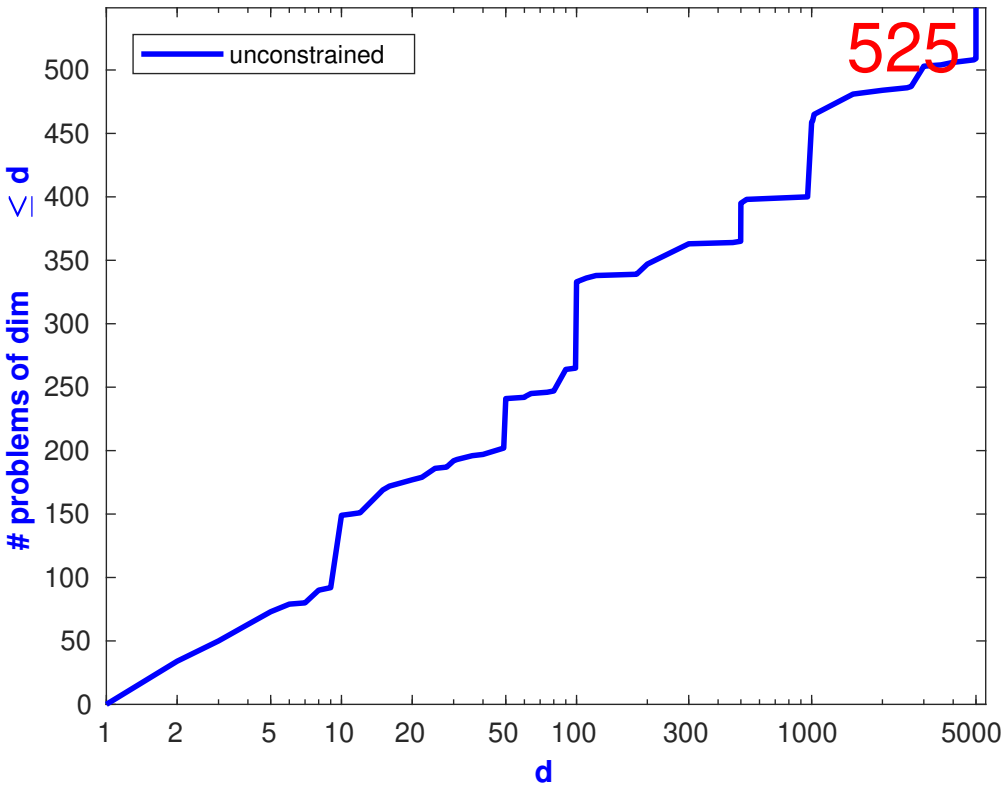


Figure 1: The number of problems with at most `d` variables solved by at least one solver: There were a total of 525 such problems

`nf` and `msec` denote the number of function evaluations and the time in milliseconds, respectively. We limited the budget available for each solver by allowing at most

$$\begin{cases} 180 & \text{if } 1 \le n \le 100, \\ 700 & \text{if } 101 \le n \le 1000 \\ 1500 & \text{if } 1001 \le n \le 5000 \end{cases}$$

seconds of run time and at most $2n^2 + 1000n + 5000$ function evaluations for a problem with n variables. A problem with small dimension is considered solved if the target accuracy satisfies

$$q_f := (f - f_{\text{best}})/(f_{\text{init}} - f_{\text{best}}) = \begin{cases} 10^{-4} & \text{if } 1 \le n \le 100, \\ 10^{-3} & \text{if } 101 \le n \le 10000 \end{cases}$$

where f_{init} is the function value of the starting point (common to all solvers) and f_{best} is the best point known to us.

Note that this amounts to testing for finding the *global minimizer* to some reasonable accuracy. We did not check which of the test problems were multimodal, so that descent algorithms might end up in a local minimum only.

There have been numerous attempts for finding the best local minimizer or global minimizer for all used test problems by calling several gradient-based solvers such as *LMBFG-DDOGL*, *LMBFG-EIG-MS* and *LMBFG-EIG-curve-inf* presented by Burdakov et al. [5], *ASACG* presented by Hagar & Zhang [22] and *LMBOPT* implemented by Kimiaei & Neumaier [29]. The condition

$$\|g_k\|_\infty \le 10^{-5}$$

holds for most test problems but not for them given in Appendix B.

5.2 Default parameters for *VSBBO*

VSBBO was implemented in Matlab; the source code is obtainable from

`http://www.mat.univie.ac.at/~neum/software/VSBBO`.

For our tests we used in `tune` the following parameter choices:

$\texttt{mmax} = 5$; $T_0 = 50n$; $C = n$; $S = \min(\texttt{fix}(n/10) + 1, 5)$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $E = +\infty$; $\texttt{scCum} = 0$; $\texttt{scCor} = 0$; $\texttt{scSub} = 0$; $\texttt{cum} = 1$; $\delta_{\min} = 0.01$; $\delta_{\max} = 1$; $\Delta_{\min} = 0$; $\Delta_{\max} = 10^{-6}$; $\gamma_\delta = 10^6$; $\gamma_{\max} = 10^{-6}$; $\gamma_E = 4$; $\gamma_\lambda = 10^{-6}$; $Q = 2$;

5.3 Codes compared

We compare *VSBBO* with the following solvers for unconstrained black box optimization. For some of the solvers we had to choose options different from the default to make them competitive.

- *RDSFS, RDSVS* and *PRDS*, obtained from the authors of Bergou et al. [4], are three versions of a stochastic direct search method with good complexity guarantees.

- *BFO*, obtained from

  `https://sites.google.com/site/bfocode/file`,

  is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli & Toint [38].

- *CMAES*, obtained from

  `http://cma.gforge.inria.fr/count-cmaes-m.php?Down=cmaes.m`,

  is the stochastic covariance matrix adaptation evolution strategy by Auger & Hansen [2]. We used *CMAES* with the following parameters:

  `oCMAES.MaxFunEvals` $=$ `nfmax`; `oCMAES.DispFinal` $=$ 0; `oCMAES.DispModulo` $=$ 0; `oCMAES.LogModulo` $=$ 0; `oCMAES.SaveVariables` $=$ 0; `oCMAES.MaxIter` $=$ `nfmax`; `oCMAES.Restarts` $=$ 7;

- *GLOBAL*, obtained from

  `http://www.mat.univie.ac.at/~neum/glopt/contrib/global.f90`,

  is a stochastic multistart clustering global optimization method by Csendes et al. [8]. We used *GLOBAL* with the following parameters:

  `oGLOBAL.MAXFNALL` $=$ `nfmax`; `oGLOBAL.MAXFN` $=$ `nfmax`/5; `oGLOBAL.DISPLAY` $=$'`off`'; `oGLOBAL.N100` $=$ 300; `oGLOBAL.METHOD` $=$'`unirandi`'; `oGLOBAL.NG0` $=$ 2

- *DE*, obtained from

  `http://www.icsi.berkeley.edu/~storn/code.html`,

  is the stochastic differential evolution algorithm by Storn & Price [41].

- *MCS*, obtained from

  `https://www.mat.univie.ac.at/~neum/software/mcs/`,

  is the deterministic global optimization by multilevel coordinate search by Huyer & Neumaier [25]. We used *MCS* with the following parameters:

  `iinit` $= 1$; `nfMCS` $=$ `nfmax`; `smax` $= 5n + 10$; `stop` $= 3n$; `local` $= 50$; `gamma` $=$ `eps`; `hess` $=$ `ones`(n, n); `prt` $= 0$.

- *BCDFO*, obtained from Anke Troeltzsch (personal communication), is a deterministic model-based trust-region algorithm for derivative-free bound-constrained minimization by Gratton et al. [21].

- *PSM*, obtained from

  `http://ferrari.dmat.fct.unl.pt/personal/alcustodio`,

  is a deterministic pattern search method guided by simplex derivatives for use in derivative-free optimization proposed by Custódio & Vicente [10, 11].

- *FMINUNC*, obtained from the Matlab Optimization Toolbox at

  `https://ch.mathworks.com/help/optim/ug/fminunc.html`,

  is a deterministic quasi-Newton or trust-region algorithm. We use *FMINUNC* with the options set by `optimoptions` as follows:

```
opts = optimoptions(@fminunc),'Algorithm','quasi-newton'
'Display', 'Iter','MaxIter',Inf,'MaxFunEvals', limits.nfmax
'TolX', 0,'TolFun',0,'ObjectiveLimit',-1e-50);
```

- *FMINSEARCH*, obtained from the Matlab Optimization Toolbox at

  `https://ch.mathworks.com/help/matlab/ref/fminsearch.html`,

  is the deterministic Nelder-Mead simplex algorithm by Lagarias et al. [31]. We use *fminseach* with the options set by `optimset` as follows:

```
opts = optimset('Display','Iter', 'MaxIter', Inf,'MaxFunEvals', ...
       limits.nfmax,'TolX', 0, 'TolFun',0,'ObjectiveLimit',-1e-50);
```

- *GCES* is a globally convergence evolution strategy presented by Diouane et al. [14, 15].

- *PSWARM*, obtained from

  `http://www.norg.uminho.pt/aivaz`

  is Particle swarm pattern search algorithm for global optimization presented by Vaz & Vicente [43].

- *MDS*, *NELDER* and *HOOKE*, obtained from

  `https://ctk.math.ncsu.edu/matlab_darts.html`

  are Multidirectional search, Nelder-Mead and Hooke-Jeeves algorithms, respectively, presented by Kelley [27].

- *MDSMAX*, *NMSMAX*, and *ADSMAX*, obtained from

  `http://www.ma.man.ac.uk/~higham/mctoolbox/`

  are Multidirectional search, Nelder-Mead simplex and alternating directions method for direct search optimization algorithms, respectively, presented by Higham [24].

- *GLODS*, obtained from

  `http://ferrari.dmat.fct.unl.pt/personal/alcustodio/`

  is Global and Local Optimization using Direct Search, presented by Custódio & Madeira [9].

- *ACRS*, obtained from

  `http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`

  is a global optimization algorithm presented by Brachetti et al. [6]

- *SDBOX*, obtained from

  `http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`

  is a derivative-Free algorithm for bound constrained optimization problems presented by [32].

VSBBO and the other stochastic algorithms use random numbers, hence give slightly different results when run repeatedly. Each solver was run only once for each problem. However, we checked in preliminary tests that the summarized results reported were quite similar when another run was done.

Some of the other solvers have additional capabilities that were not used in our tests; e.g., allowing for bound constraints or integer constraints, or for noisy function values). Hence our conclusions are silent about the performance of these solvers outside the task of *global unconstrained* black box optimization with noiseless function values (apart from rounding errors).

5.4 Results for small dimensions ($n \leq 20$)

We tested all 24 solvers on low dimensional test problems ($n \leq 20$). The problems unsolved by all solvers are HATFLDFL, FLETCBV3 and FLETCHBV.

Performance plots [16] for two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds) are shown in Figure 2.

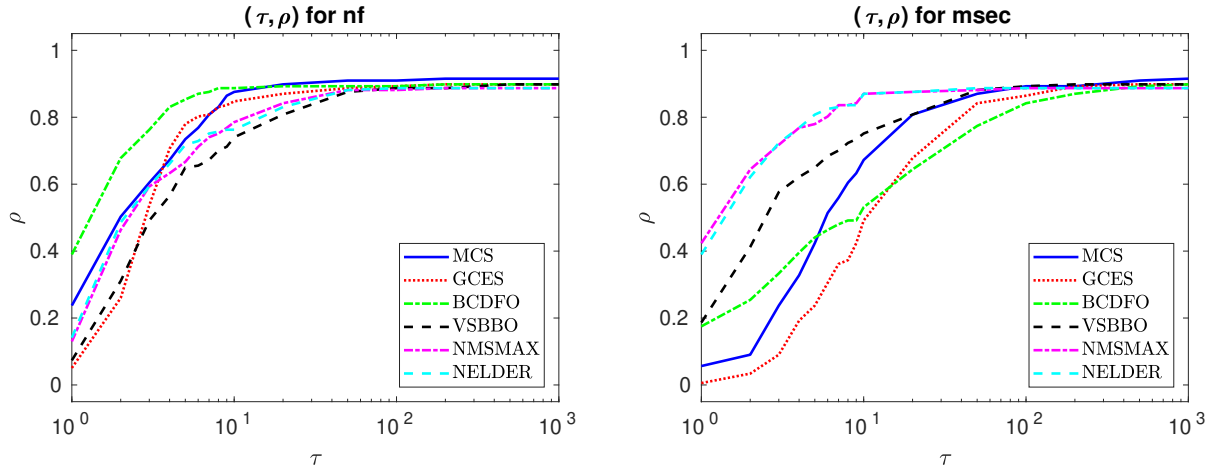


Figure 2: Small dimensions $2 - 20$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ denotes the fraction of problems solved within a factor τ of the best solver.

For a more refined statistics, we use our test environment (Kimiaei & Neumaier [28]) for comparing optimization routines on the `CUTEst` test problem collection by Gould et al. [19]. For a given collection S of solvers, the strength of a solver $so \in S$ – relative to an

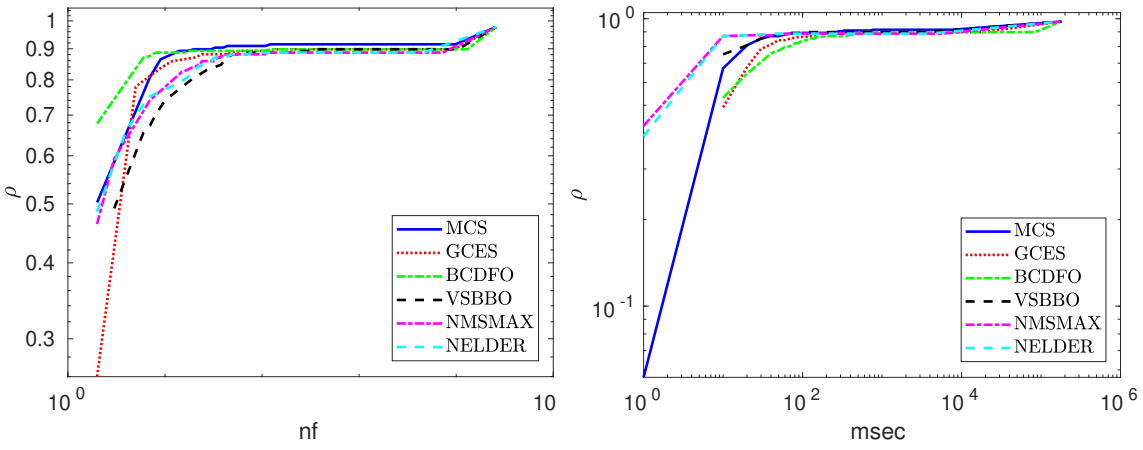


Figure 3: Small dimensions $2-20$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

ideal solver that matches on each problem the best solver – is measured, for any given cost measure c_s by the number, q_{so} defined by

$$q_{so} := \begin{cases} (\min_{s\in S} c_s)/c_{so}, & \text{if } so \text{ solved by the problem,} \\ 0, & \text{otherwise,} \end{cases}$$

called the **efficiency** of the solver *so* with respect to this cost measure. In the tables, efficiencies are given in percent. Larger efficiencies in the table imply a better average behaviour; a zero efficiency indicates failure. All values are rounded (towards zero) to integers. Mean efficiencies are taken over the 174 problems tried by all solvers and solved by at least one of them, from a total of 177 problems. In the following tables, #100 and !100 count the number of times we have `nf` efficiency 100% or unique `nf` efficiency 100%. T_{mean} is defined by

$$T_{\text{mean}} := \frac{\sum \text{ solved}}{\#\text{ solved}}.$$

Failure reasons were reported in the anomaly columns:

- n indicates that `nf` $\geq 2n^2 + 1000n + 5000$ `was reached`.
- t indicates that `sec` ≥ 180 `was reached`.
- f indicates that the `algorithm failed` for other reasons.

In the times, the (for some problems significant) setup time for `CUTEst` is not included. Although running times are reported, the comparison of times is not very reliable for several reasons:
(i) The times were obtained under different conditions (solver source code Fortran, C and Matlab).
(ii) In unsuccessful runs, the actual running time depends a lot on when and why the solver

Table 12: The summary results for small dimensions $n \leq 20$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq 2n^2 + 1000n + 5000$										
174 of 177 problems without bounds solved									mean efficiency in %	
dim∈[1,20]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
MCS	*mcs*	162	21	20	428	3	0	12	41	13
GCES	*gecs*	159	5	3	1684	1	0	17	29	9
BCDFO	*bcd*	159	42	38	3796	0	1	17	53	21
VSBBO	*vsbb*	159	2	2	241	18	0	0	25	23
NMSMAX	*nmsm*	157	11	10	193	12	0	8	36	41
NELDER	*neld*	157	16	13	190	0	0	20	37	42
PSM	*psm*	156	7	6	1222	10	2	9	38	14
SDBOX	*sdb*	144	7	7	205	33	0	0	25	43
FMINUNC	*func*	142	54	51	84	0	0	35	53	53
CMAES	*cma*	142	3	2	457	35	0	0	9	10
BFO	*bfo*	140	1	1	269	0	0	37	19	19
ADSMAX	*adsm*	126	1	1	662	37	0	14	14	10
MDSMAX	*mdsm*	123	3	2	318	53	0	1	12	18
GLODS	*glods*	123	9	8	1085	51	0	3	13	5
PSWARM	*psw*	122	2	0	491	33	0	22	7	6
HOOKE	*hook*	120	0	0	196	13	0	44	18	25
FMINSEARCH	*fmin*	97	0	0	575	20	0	60	7	6
GLOBAL	*glo*	93	1	1	166	21	0	63	6	12
DE	*de*	84	0	0	769	68	0	25	0	2
RDSvs	*rvs*	57	1	0	258	120	0	0	3	8
PRDS	*prd*	55	0	0	913	122	0	0	1	2
RDSfs	*rfs*	52	0	0	308	125	0	0	2	5
ACRS	*acr*	50	1	0	330	83	0	44	3	4
MDS	*mds*	13	3	1	133	96	0	68	2	3

was stopped. Table entries use the maximal allowed time (180 sec) for each unsuccessful run.

In Table 12, we see that on problems with few variables, *VSBBO* has above average performance and can compete in robustness with the best solvers *MCS*, *GCES*, *NELDER* and *BCDFO*.

5.5 Results for medium dimensions ($21 \leq n \leq 100$)

We tested all 24 solvers on medium dimensional test problems ($21 \leq n \leq 100$). The problems unsolved by all solvers are NONMSQRT:49, CURLY10:100, NONMSQRT:100 and OSCIGRAD:100.

Performance plots [16] for two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds) are shown in Figure 4. The other performance plots for two mentioned cost measures are shown in Figure 5.

In Table 13, we see that on problems with a medium number of variables, *VSBBO* is outstanding in robustness, but *FMINUNC* is the best in terms of `nf` and `sec` efficiency.

Table 13: The summary results for medium dimensions $21 - 100$

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq 2n^2 + 1000n + 5000$										
152 of 156 problems without bounds solved									mean efficiency in %	
dim∈[21,100]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	*vsbb*	140	30	30	1785	16	0	0	46	38
MCS	*mcs*	133	3	3	3852	3	0	20	22	14
FMINUNC	*func*	128	74	72	911	0	0	28	62	68
SDBOX	*sdb*	126	10	9	1615	30	0	0	37	45
ADSMAX	*adsm*	115	9	9	5021	36	0	5	26	15
CMAES	*cma*	101	1	0	14852	52	3	0	7	2
PSWARM	*psw*	100	0	0	11142	51	0	5	3	2
NMSMAX	*nmsm*	94	3	3	5214	60	2	0	13	10
MDSMAX	*mdsm*	90	1	0	4172	66	0	0	4	7
NELDER	*neld*	90	3	3	25525	5	29	32	13	4
BFO	*bfo*	83	0	0	4781	0	4	69	5	5
HOOKE	*hook*	76	0	0	4616	11	0	69	10	7
DE	*de*	73	0	0	3670	48	0	35	1	4
GLOBAL	*glo*	36	1	1	1925	31	0	89	4	5
PSM	*psm*	31	4	4	28431	0	125	0	8	1
GLODS	*glods*	28	5	5	2225	0	0	128	4	3
GCES	*gecs*	27	0	0	42836	0	124	5	4	0
BCDFO	*bcd*	20	11	10	29203	0	135	1	7	1
RDSvs	*rvs*	18	0	0	6906	137	1	0	0	0
PRDS	*prd*	17	0	0	4408	138	1	0	0	1
RDSfs	*rfs*	15	0	0	7302	140	1	0	0	1
FMINSEARCH	*fmin*	14	0	0	6433	136	2	4	0	0
MDS	*mds*	7	0	0	4691	146	2	1	0	0
ACRS	*acr*	6	0	0	27262	84	66	0	0	0

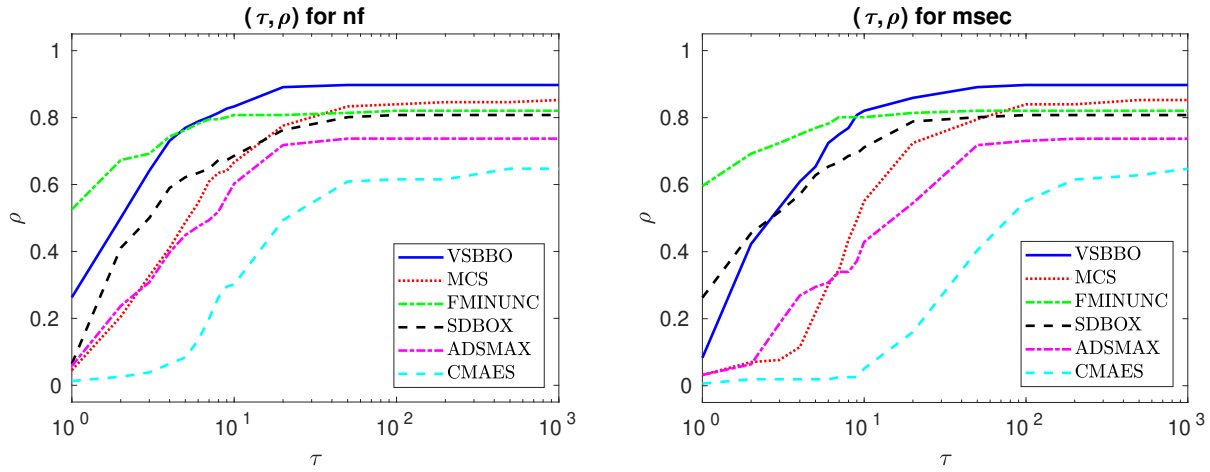


Figure 4: Medium dimensions $21 - 100$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

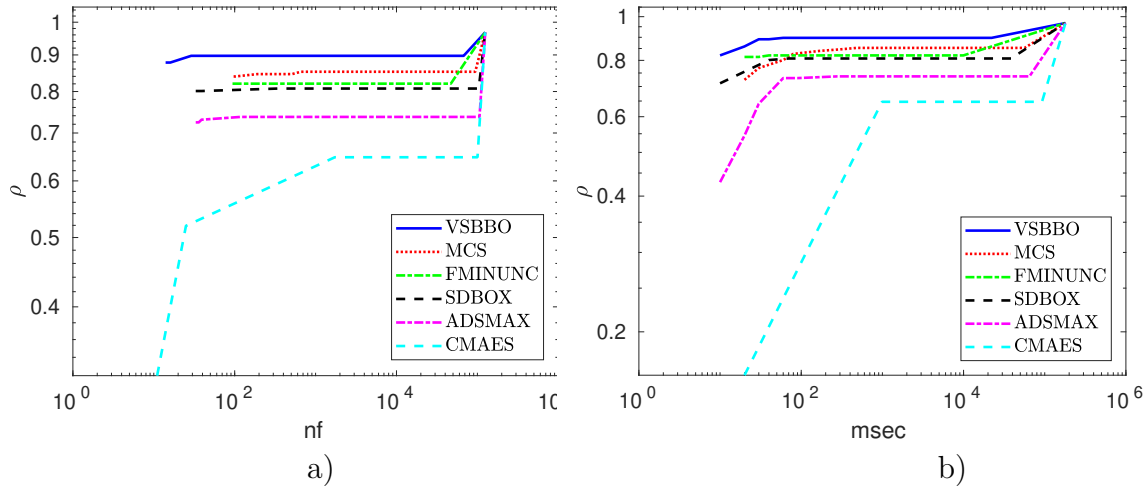
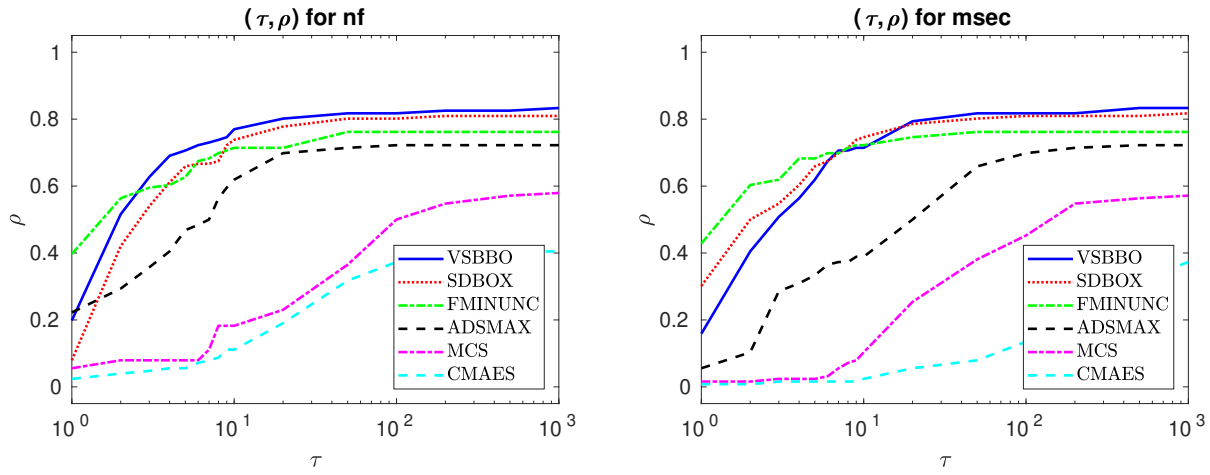


Figure 5: Medium dimensions $21 - 100$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

5.6 Results for large dimensions ($101 \leq n \leq 1000$)

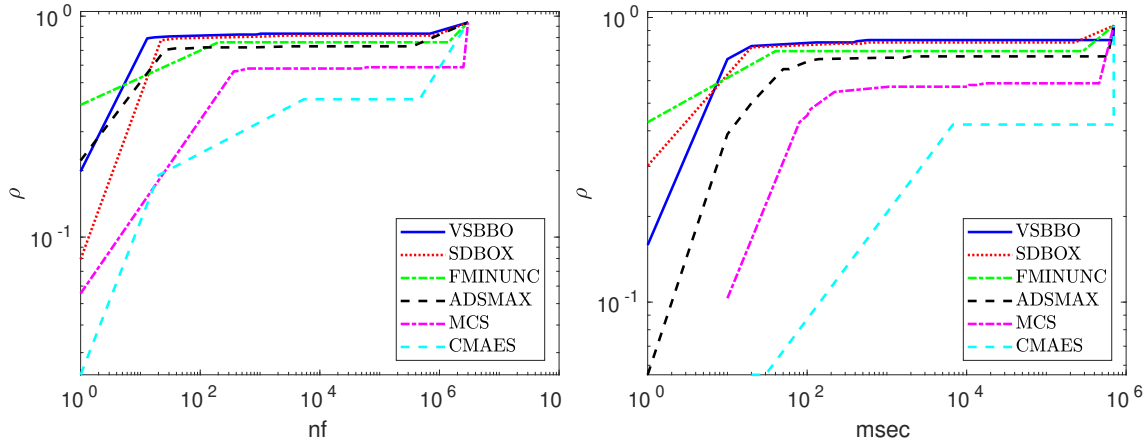
We tested the 6 most robust solvers from Table 13 on large dimensional test problems ($101 \leq n \leq 1000$). The problems unsolved by the 6 solvers are GENROSE:500, NONMSQRT:529, CURLY20, CURLY30, FLETCBV3:1000, FLETCHCR:1000, SENSORS:1000 and SPMSRTLS:1000.

Performance plots [16] for two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds) are shown in Figure 6. The other performance plots for two mentioned cost measures are shown in Figure 7.



a) b)

Figure 6: Large dimensions $101 - 1000$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.



a) b)

Figure 7: Large dimensions $101 - 1000$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 14: The summary results for for large dimensions $101 - 1000$

stopping test: $q_f \le$ 0.001, sec $\le$ 700, nf $\le 2n^2 + 1000n + 5000$										
118 of 126 problems solved									mean efficiency in %	
dim∈[101,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	*vsbb*	105	25	24	17780	2	19	0	53	41
SDBOX	*sdb*	103	10	9	10544	10	13	0	42	52
FMINUNC	*func*	96	52	49	9831	2	1	27	55	59
ADSMAX	*adsm*	92	28	27	42157	2	26	6	34	18
MCS	*mcs*	74	7	4	83711	0	6	46	9	4
CMAES	*cma*	53	3	2	214364	4	68	1	6	1

A comparison of five solvers in Table 14 shows that *VSBBO* is the most robust solver. It is better than *SDBOX* in terms of the `nf` efficiency and than *FMINUNC* in terms of the number of solved problems.

5.7 Results for large dimensions ($1001 \le n \le 5000$)

We tested the 3 most robust solvers from Table 14 on large dimensional test problems ($101 \le n \le 1000$). The problems unsolved by all solvers are NONMSQRT:1024, EIGENALS:2550, EIGENBLS:2550, MSQRTALS:4900, MSQRTBLS:4900, SPMSRTLS:4999, FLETCBV2:5000 and NONCVXU2:5000.

Performance plots [16] for two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds) are shown in Figure 8. The other performance plots for two mentioned cost measures are shown in Figure 9.

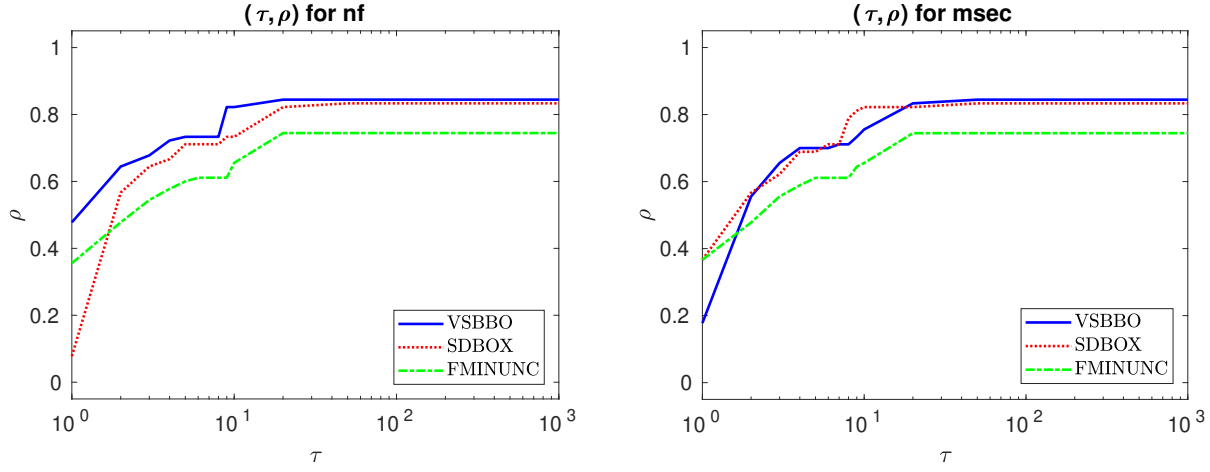
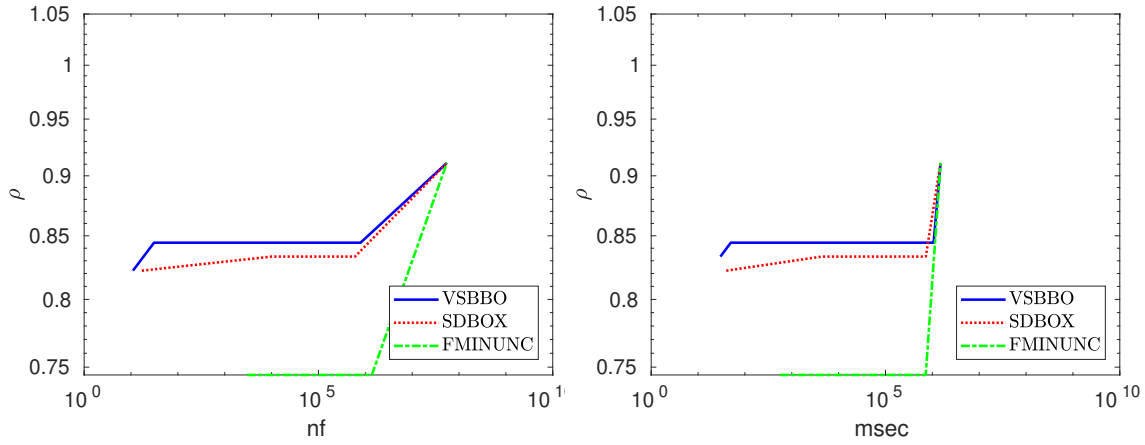


Figure 8: Large dimensions $1001 - 5000$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

In Table 15, we have a comparison of best solvers *SDBOX*, *VSBBO* and *FMINUNC* for large dimensions 1001 up 5000. The high quality of *VSBBO* is clearly visible. *VSBBO* is



a) b)

Figure 9: Large dimensions $1001 - 5000$: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

Table 15: The summary results for large dimensions $1001 - 5000$

stopping test: $q_f \leq$ `0.001`, `sec` $\leq$ `1500`, `nf` $\leq 2n^2 + 1000n + 5000$										
82 of 90 problems without bounds solved									mean efficiency in %	
dim∈[1001,5000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	vsbb	76	44	43	70724	0	14	0	64	53
SDBOX	sdb	75	8	7	61381	0	15	0	49	59
FMINUNC	func	67	32	30	92763	0	10	13	49	50

the best in terms of the `nf` efficiency, #100 and !100.

5.8 Results for all dimensions

For dimensions 1 up 5000, *VSBBO*, *SDBOX* and *FMINUNC* have solved 477, 448 and 433 test problems, respectively. Hence *VSBBO* is robust solver.

Acknowledgement The first author acknowledges the financial support of the Doctoral Program "Vienna Graduate School on Computational Optimization" funded by Austrian Science Foundation under Project No W1260-N35.

References

[1] C. Audet and D. Orban, Finding optimal algorithmic parameters using derivative free optimization, SIAM J. Optim 17 (2006), 642–664.

[2] A. Auger and N. Hansen, A restart CMA evolution strategy with increasing population size, In: The 2005 IEEE congress on evolutionary computation 2 (2005), 1769–1776.

[3] C.J. Bélisle, H.E. Romeijn, and R.L. Smith, Hit-and-run algorithms for generating multivariate distributions, Math. Oper. Res. 18 (1993), 255–266.

[4] E.H. Bergou, E. Gorbunov and P. Richtárik, Random direct search method for minimizing nonconvex, convex and strongly convex functions, Manuscript (2018).

[5] Burdakov, Oleg and Gong, Lujin and Zikrin, Spartak and Yuan, Ya-xiang, On efficiently combining limited-memory and trust-region techniques, Math. Program. Comput. 9(1) (2017), 101–134.

[6] P. Brachetti, G. Di Pillo, M. De Felice Ciccoli, S. Lucidi, A New Version of the Prices Algorithm for Global Optimization, J. Glob. Optim. 10 (1997) 165–184.

[7] A.R. Conn, K. Scheinberg, and L.N. Vicente, Introduction to derivative-free optimization, SIAM, Philadelphia, PA, 2009.

[8] T. Csendes, L. Pál, J.O.H. Sendin and J.R. Banga, The GLOBAL optimization method revisited, Optim. Lett. 2 (2008), 445–454.

[9] A.L. Custódio and J.F.A. Madeira, GLODS: Global and Local Optimization using Direct Search, J. Glob. Optim., 62 (2015), 1–28.

[10] A.L. Custódio, L.N. Vicente, Using sampling and simplex derivatives in pattern search methods, SIAM J. Optim 18 (2007), 537–555.

[11] A.L. Custódio, H. Rocha, L.N. Vicente, Incorporating minimum Frobenius norm models in direct search, Comput. Optim. Appl. 46 (2010), 265–278.

[12] Y. Diouane, S. Gratton, and L.N. Vicente, Globally convergent evolution strategies, Math. Program. 152 (2015) 467–490.

[13] Y. Diouane, S. Gratton, and L.N. Vicente, Globally convergent evolution strategies for constrained optimization, Comput. Optim. Appl. 62 (2015) 323–346.

[14] M. Dodangeh, L.N. Vicente, Worst case complexity of direct search under convexity, Math. Program. 155(1–2) (2016), 307–332.

[15] M. Dodangeh, L.N. Vicente, Z. Zhang, On the optimal order of worst case complexity of direct search, Optim. Lett. 10(4) (2016), 699–708.

[16] E. Dolan and J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2002), 201–213.

[17] C. Elster and A. Neumaier, A grid algorithm for bound constrained optimization of noisy functions, IMA J. Numer. Anal. 15 (1995), 585–608.

[18] C. Elster and A. Neumaier, A trust region method for the optimization of noisy functions, Computing 58 (1997), 31–46.

[19] N.I.M. Gould, D. Orban, Ph.L. Toint, `CUTEst`: a constrained and unconstrained testing environment with safe threads for mathematical optimization, Comput. Optim. Appl. 60(3) (2015), 545–557.

[20] S. Gratton, C.W. Royer, L. N. Vicente and Z. Zhang, Direct search based on probablistic descent, SIAM J. Optim 25 (2015), 1515–1541.

[21] S. Gratton, Ph. L. Toint, and A. Tröeltzsch, An active-set trust-region method for derivative-free nonlinear bound-constrained optimization, Optim. Methods Softw. 26(4–5) (2011) 875–896.

[22] W.W. Hager and H. Zhang, A New Active Set Algorithm for Box Constrained Optimization, SIAM J. Optim 17(2) (2006), 526–557.

[23] N. Hansen, The CMA evolution strategy: a comparing review, pp. 75–102 in: Towards a new evolutionary computation, Advances on estimation of distribution algorithms (J.A. Lozano, ed.), Springer, Berlin 2006.

[24] N.J. Higham, Optimization by direct search in matrix computations, SIAM J. Matrix Anal. Appl. 14(2)(1993) 317–333.

[25] W. Huyer and A. Neumaier, Global optimization by multilevel coordinate search, J. Glob. Optim. 14 (1999), 331–355.

[26] W. Huyer and A. Neumaier, SNOBFIT–stable noisy optimization by branch and Fit, ACM. Trans. Math. Software 35, Article 9 (2008).

[27] C.T. Kelley, Iterative methods for optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[28] M. Kimiaei, A. Neumaier, Testing and tuning optimization algorithm, in preparation (2019).

[29] M. Kimiaei, A. Neumaier, LMBOPT–a limited memory method for bound-constrained optimization, in preparation (2019).

[30] J. Konečný and P. Richtárik, Simple complexity analysis of simplified direct search, Manuscript (2014), `https://arxiv.org/abs/1410.0390`.

[31] J.C. Lagarias, J.A. Reeds, M.H. Wright, P.E. Wright, Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions, SIAM J. Optim, 9(1) (1998), 112–147.

[32] S. Lucidi, M. Sciandrone, A Derivative-Free Algorithm for Bound Constrained Optimization, Comput. Optim. Appl. 21(2) (2002), 119–142.

[33] Y. Nesterov, Random gradient-free minimization of convex functions, CORE discussion paper #2011/1, unpublished (2011).

[34] Y. Nesterov and V. Spokoiny, Random gradient-free minimization of convex functions, Found. Comput. Math. 17 (2017), 527–566.

[35] A. Neumaier, H. Fendl, H. Schilly and T. Leitner, Derivative-free unconstrained optimization based on QR factorizations, Soft Computing 15 (2011), 2287–2298.

[36] J. Nocedal, S. Wright, Numerical Optimization. Springer New York, 2 edition, 1999.

[37] M. Porcelli, Ph. L. Toint, A note on using performance and data profiles for training algorithms, (2017), `https://arxiv.org/abs/1711.09407`

[38] M. Porcelli, Ph. L. Toint, BFO, a trainable derivative–free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables, ACM. Trans. Math. Software 44–1 (2017), Article 6, 25 pages.

[39] I. Pinelis, A probabilistic angle inequality, MathOverflow (2018). `https://mathoverflow.net/q/298590`

[40] L.M. Rios and N.V. Sahinidis, Derivative-free optimization: A review of algorithms and comparison of software implementations, Manuscript (2009).

[41] R. Storn and K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (1997), 341–359.

[42] P.J.M. Van Laarhoven and E.H.L. Aarts, Simulated annealing: theory and applications, Kluwer, Dordrecht (1987).

[43] A.I.F. Vaz and L.N.Vicente, A particle swarm pattern search method for bound constrained global optimization, J. Glob. Optim. 39 (2007), 197–219.

[44] L.N. Vicente, Worst case complexity of direct search, EURO J. Comput. Optim. 1(1–2) (2013), 143–153.

A Appendix: Estimation of c

The following theorem was recently proved by Pinelis [39].

A.1 Theorem. *There is a universal constant c_0 such that for any fixed nonzero real vector q of any dimension n and any random vector p of the same dimension n with independent components uniformly distributed in $[-1, 1]$, we have*

$$(p^T p)(q^T q) \leq c_0 n (p^T q)^2 \tag{36}$$

with probability $\geq 1/2$.

More specifically, Pinelis proved the bounds $0.73 < c_0 < 50$ for the optimal value of the constant c_0. The true optimal value seems to be approximately $16/7$. This is suggested by numerical simulation. To estimate c_0, we executed three times the Matlab commands

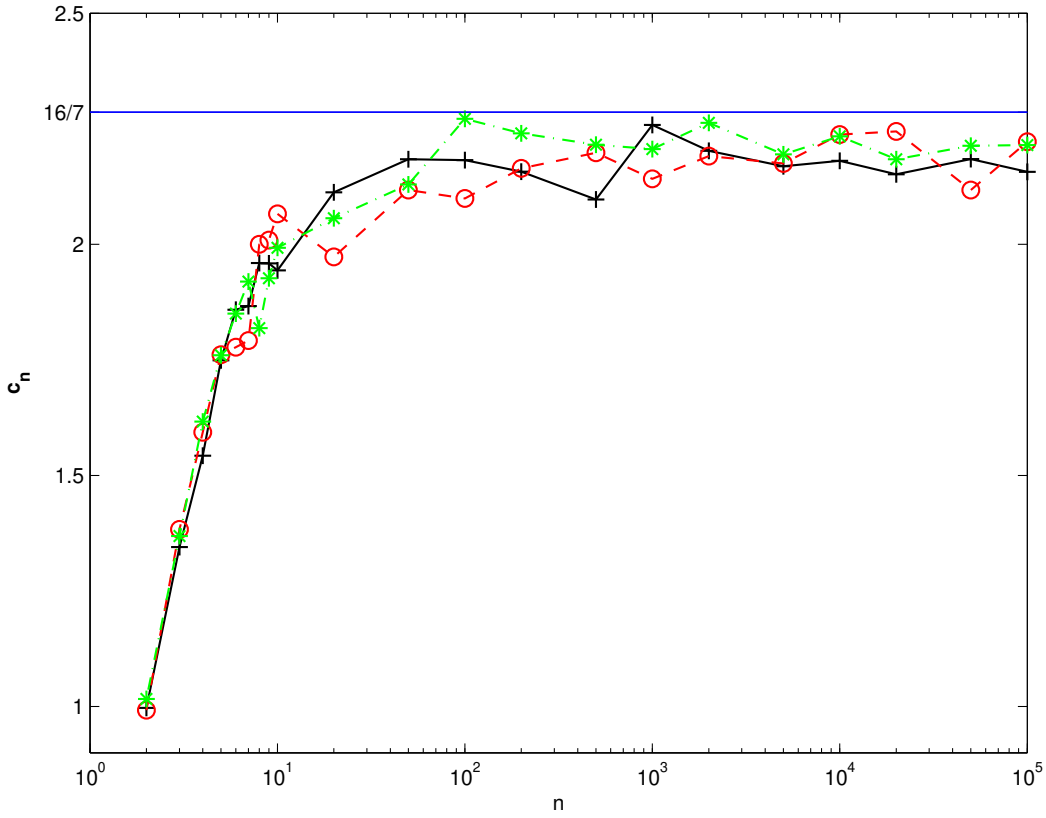


Figure 10: The plot of c_n versus the dimension n suggests that $c_0 \approx 16/7$.

```
% run PinConst
N=10000;
nlist=[2:10,20,50,100,200,500,1000,2000,5000,10000,20000,50000,100000];
c0=PinConst(N,nlist);
```

using the algorithm *PinConst* below. All three outputs,

$$c_0 = 2.2582, c_0 = 2.2444, c_0 = 2.2714$$

are slightly smaller than $16/7 = 2.2857....$

A.2 Algorithm. (Estimating the Pinelis constant)

Purpose: Estimate c_0 satisfying (36) with probability $\geq 1/2$
Input: N (the total number of gradient evaluations) D (vector of dimensions used)
Output: c_0
$[c_0]$=**PinConst**(N,D);

$M = \|D\|$;
for $i = 1, \cdots, M$,
  for $k = 1, \cdots, N$,
  generate random g_k and p_k with length D_i;
  $\texttt{gain}(k) = \dfrac{\|g_k\|_2 \|p_k\|_2}{\|g_k^T p_k\|}$;
  end;
  $\texttt{medgain}(i) = \texttt{median}(\texttt{gain})$; % obtain median of gain
  $c(i) = \texttt{medgain}(i))^2/D_i$;
end;
$c_0 = \max(c)$;

B Appendix: A list of test problems with f_{best}

Here is given a list of some test problems for which the condition

$$\|g_k\|_\infty \leq 10^{-5}$$

don't hold.

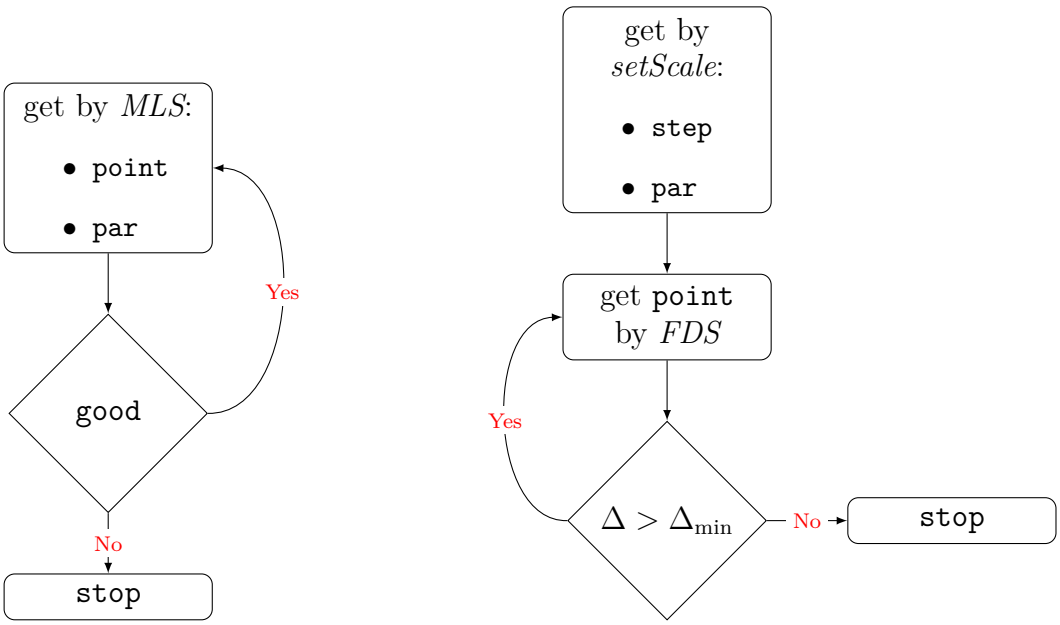
poblem	dim	f_{best}	$\|g\|_\infty$	$\|g\|_2$
BROWNBS	2	$-2.80e+00$	$2.08e-05$	$2.08e-05$
DJTL	2	$-8.95e+03$	$1.44e-04$	$1.44e-04$
STRATEC	10	$2.22e+03$	$8.10e-05$	$1.42e-04$
SCURLY10:10	10	$-1.00e+03$	$5.34e-04$	$5.50e-04$
OSBORNEB	11	$2.40e-01$	$3.47e-02$	$3.47e-02$
ERRINRSM:50	50	$3.77e+01$	$1.89e-05$	$1.89e-05$
ARGLINC:50	50	$1.01e+02$	$1.29e-05$	$5.28e-05$
HYDC20LS	99	$1.12e+01$	$5.54e-01$	$8.79e-01$
PENALTY3:100	100	$9.87e+03$	$2.01e-03$	$4.68e-03$
SCOSINE:100	100	$-9.30e+01$	$1.95e-02$	$3.58e-02$
SCURLY10:100	100	$-1.00e+04$	$5.74e-02$	$1.56e-01$
NONMSQRT:100	100	$1.81e+01$	$3.42e-05$	$6.51e-05$

Continued on next page

PENALTY2:200	200	$4.71e+13$	$3.85e-04$	$1.07e-03$
ARGLINB:200	200	$9.96e+01$	$3.27e-04$	$2.68e-03$
SPMSRTLS:499	499	$1.69e+01$	$1.08e-05$	$3.59e-05$
PENALTY2:500	500	$1.14e+39$	$1.97e+26$	$4.08e+26$
MSQRTBLS:529	529	$1.13e-02$	$1.44e-05$	$1.03e-04$
NONMSQRT:529	529	$6.13e+01$	$2.17e-05$	$1.76e-04$
SCOSINE	1000	$-9.21e+02$	$3.38e-03$	$9.32e-03$
SCURLY10	1000	$-1.00e+05$	$5.49e+01$	$3.37e+02$
COSINE	1000	$-9.99e+02$	$5.00e-05$	$6.34e-05$
PENALTY2:1000	1000	$1.13e+83$	$2.53e+77$	$3.41e+77$
SINQUAD:1000	1000	$-2.94e+05$	$1.21e-05$	$1.52e-05$
SPMSRTLS:1000	1000	$3.19e+01$	$9.75e-05$	$2.26e-04$
NONMSQRT:1024	1024	$9.01e+01$	$1.73e-04$	$1.28e-03$
MSQRTALS:4900	4900	$7.60e-01$	$1.88e-03$	$3.56e-02$
SPMSRTLS:4999	4999	$2.05e+02$	$2.36e-03$	$9.27e-03$
INDEFM:5000	5000	$-5.02e+05$	$1.43e-05$	$2.00e-05$
SBRYBND:5000	5000	$2.58e-10$	$3.73e-04$	$3.50e-03$
SCOSINE:5000	5000	$-4.60e+03$	$6.32e-03$	$2.72e-02$
NONCVXUN:5000	5000	$1.16e+04$	$3.94e-05$	$7.19e-04$

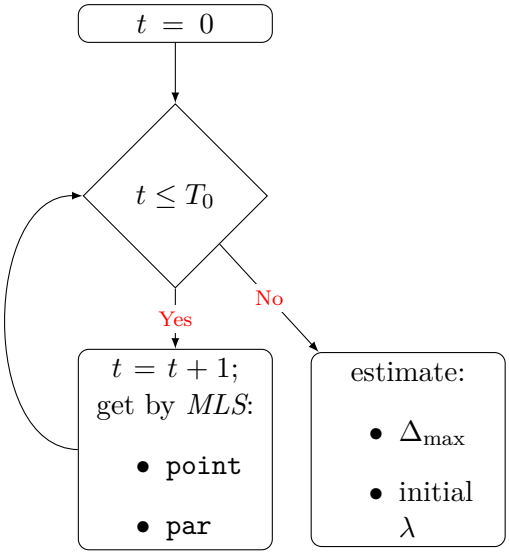
C Appendix: Flow charts

Here we give flow charts for the algorithms *setScale*, *MLS*, *FDS*, and *VSBBO*.

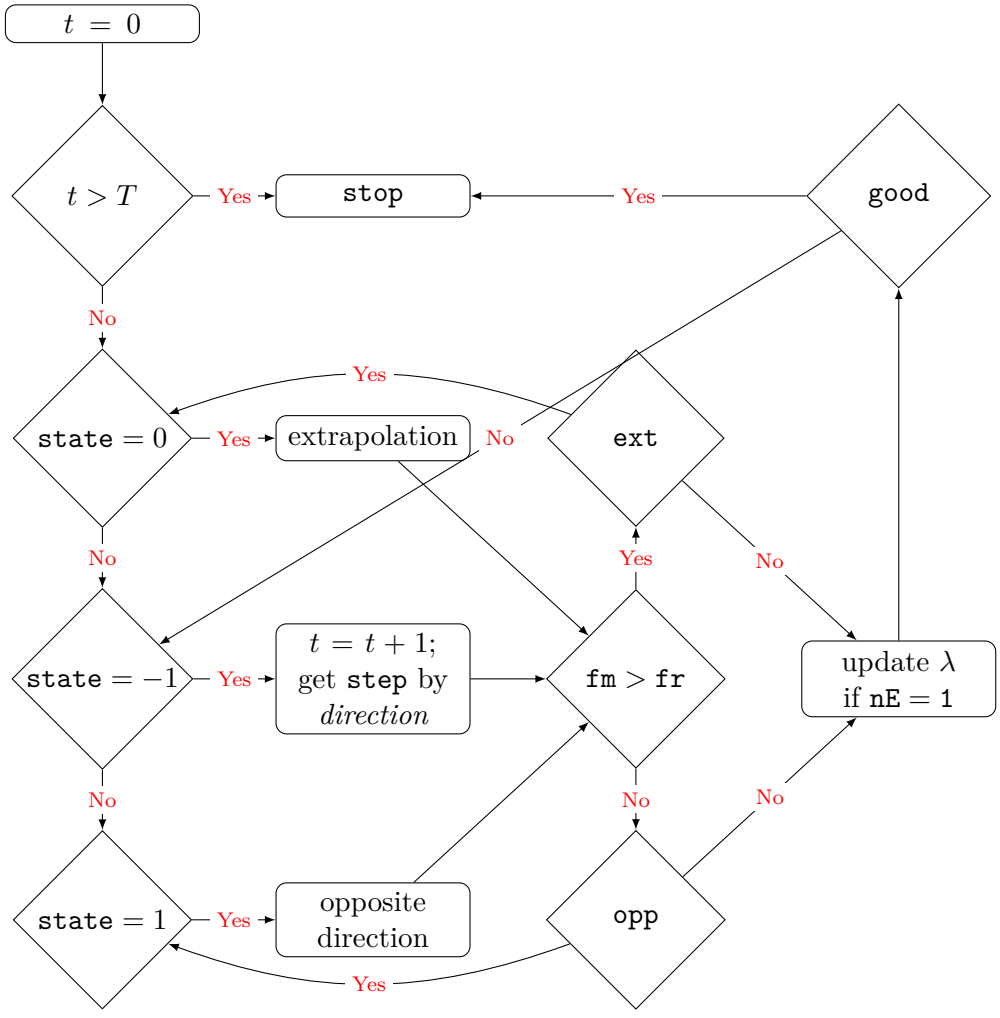


(a) (b)

Figure 11: Flow charts for (a) *FDS*, and (b) *VSBBO*.



(a)



(b)

Figure 12: Flow charts for (a) *setScale*, (b) *MLS*.