MPC manuscript No.
(will be inserted by the editor)

Efficient global unconstrained black box optimization

Morteza Kimiaei · Arnold Neumaier

the date of receipt and acceptance should be inserted later

Abstract For the unconstrained 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, in small and large dimensions, a local minimizer with reasonable accuracy. Although our theory guarantees only local minimizers our heuristic techniques turn **VSBBO** into an efficient global solver. In very thorough numerical experiments, we found in most cases either a global minimizer, or where this could not be checked, at least a point of similar quality with the best competitive global solvers.

For smooth, everywhere defined functions, it is proved that, with probability arbitrarily close to 1, the basic version of our algorithm finds with $O(nR\epsilon^{-2})$ function evaluations a point with gradient 2-norm $\leq \epsilon$, where n is the dimension and R is the number of random directions used in each iteration. In the smooth convex case, this number improves to $O(nR\epsilon^{-1})$ and in the smooth (strongly) convex case to $O(nR\log\epsilon^{-1})$. 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

2000 AMS Subject Classification: primary 90C56.

July 20, 2020

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.

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

This problem (see, e.g., [9,48]) 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. A thorough survey for defivative-free optimization methods was given by Larson et al. [38]. Another useful paper suggested by Rios & Sahinidis [48] discusses the practical behaviour of derivative-free optimization softwares. 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. [9] and its many references. Stochastic methods for BBO go back to [51] and were later discussed especially in the framework of evolutionary optimization [2,28,50]. There is also a randomized version of deterministic algorithm, e.g., Bandeira et al. [3].

Past BBO software of our group includes the deterministic algorithms **GRID** [20,21] and **MCS** [30] and the stochastic algorithms **SnobFit** [31] and **VXQR** [42]. Software by many others is mentioned in Section 7.1, where an extensive numerical comparison is discussed.

1.1 Our contribution

Our goal in this paper is to describe a new, practically very efficient stochastic method, called **VSBBO**, for which good local complexity results can be proved, and which is competitive with the state of the art solvers for global BBO.

An algorithm loosely related to **VSBBO** (but without complexity guarantees) is the **Hit-and-Run** algorithm by Bélisle [4].

VSBBO contains many new useful techniques, in particular:

- Several kinds of search ensure the good practical performance.
- Adapive heuristic estimations for the Lipschitz constant are used.
- A sensible scaling vector is estimated.
- The gradient vector is estimated by a stochastic finite approach.

In Section 2-4, we discuss a basic version of **VSBBO**, only using some random directions with a guarantee for finding complexity bounds for the nonconvex, convex, and strongly convex cases. Then we introduce heuristics that makes **VSBBO** the best stochastic algorithm in comparison with other known algorithms. However, even the basic version of our algorithm is more efficient than algorithms suggested by Gratton et al. [25] and Bergou et al. [6].

As will be shown in Section 4, 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 $\leq \epsilon$, given in Table 3 below.

The numerical results of Section 7 show that **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.

We want to find an efficient algorithm that, starting from a point x_0, finds with high probability a point $x_{\rm best}$ satisfying

$$f(x_{\rm best}) \leq \sup\{f(x) \leq f(x_0) \mid \|g(x)\|_* \leq \varepsilon\}. \tag{1}$$

Here $g(x)$ denotes the gradient of f at x. 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 $s \in \mathbb{R}^n$.

Clearly, any local minimizer with function value $\leq f(x_0)$ satisfies condition (1) for every $\varepsilon > 0$. However, for fixed $\varepsilon > 0$, this may also be satisfied far away from a local minimizer that lies on very flat path of graph of f. 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.

As a guarantee for our complexity results, we need that sufficiently many search directions p satisfy an **angle condition** of the form

$$\sup \frac{g^T p}{\|g\|_* \|p\|} \leq -\omega < 0. \tag{3}$$

In Section 2.2, it is shown that random directions generated by (11) and scaled by (15) satisfies the angle condition (3) with probability arbitrarily close to 1. This is the key property needed later in order to obtain our complexity results for the nonconvex, convex, and strongly convex cases.

This paper is organized as follows. In Section 2, a stochastic line search, called **SLS**, is constructed. Section 3 first presents a fixed decrease search algorithm, called **FDS** to hopefully get a decrease in the function value. It has repeated calls to **SLS** until the function value is decreased. Then the basic version of our algorithm is presented while having repeated calls to **FDS** until the function value is decreased. In Section 4, the complexity bounds are proven, with probability arbitrarily close to 1, for the nonconvex, convex, and strongly convex cases. In Section 5, we introduce heuristic techniques to improve the performance in practice, leading to the **VSBBO** implemented documentation in Section 6. The improved version of our algorithm including all heuristic techniques and directions is presented in Section 6. Section 7 provides numerical results, comparing **VSBBO** with the best previous black box algorithms on a large benchmark of problems in low, medium, and high dimensions.

1.2 Complexity

Complexity bounds limit the size of the number $N(\varepsilon)$ of function evaluations needed to reach the goal (1) with a given probability (or a related goal). The appropriate asymptotic form for the expression $N(\varepsilon)$, found by Vicente [53], Dodangeh & Vicente [16], Dodangeh, Vicente & Zhang [17], Gratton et al. [25], Bergou, Gorbunov & Richtárik [6], and Nesterov & Spokoiny [40,41], 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) The level set $\mathcal{L}(x_0) := \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$ of x_0 is compact.

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

Table 1: Complexity results for stochastic BBO in expectation (Bergou et al. [6] for all cases)

Bergou et al. [6] and Nesterov & Spokoiny [41] generalized this result to give algorithms with complexity results for the nonconvex, convex and strongly convex cases 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 [16], Vicente [53] and Konečný & Richtárik [36]) 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. Moreover, they generated step sizes without testing whether the function value is decreased or not. This is the reason why the algorithms proposed by Bergou et al. [6] are numerically poor, see Section 7.

The best complexity bound for a direct search with probabilistic (rather than expectation) guarantees has been found by Gratton et al. [25], only for nonconvex case. They used the Chernoff bounds to prove that a

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

Table 2: Complexity results for deterministic BBO (Vicente [53] for the nonconvex case, Dodangeh & Vicente [16] for the convex and the strongly convex cases, Konečný & Richtárik [36] for all cases)

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

Table 3: Complexity results for stochastic BBO with probability $1 - \eta$, for fixed $0 < \eta < 1$ (Gratton et al. [25] for the nonconvex case, present paper for all cases). Here R is the number of random directions used in each iteration.

complexity bound $\mathcal{O}(nR\varepsilon^{-2})$ holds, R is the number of random directions, uniformly independently distributed on the unit sphere, used in each iteration.

Our complexity bound for nonconvex (obtained by a numerical simulation; see Appendix A) is the same as that of found by Gratton et al. [25] using the Chernoff bounds. Both complexity results are better by the factor of R/n than those of given in Table 1 and are more reasonable than those given in Table 2.

In Sections 4.2 and 4.3, we find complexity bounds for the convex and strongly convex cases with probability arbitrarily close to 1, which are new results and are more reasonable than those given in Table 2, only valid in expectation.

Table 3 summarizes our complexity results for all cases, matching Gratton et al. [25] for the nonconvex case. Gratton et al.'s results for the nonconvex case allow $R = 2$, while **VSBBO** requires

$$R = \mathcal{O}\Big(\log\log \frac{n}{\varepsilon^2} + \log \eta^{-1}\Big).$$

But $\log\log \frac{n}{\varepsilon^2}$ and $\log \eta^{-1}$ cannot be large for reasonable values of n, ε, and η.

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 some random directions. 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.

A line search then polls one or more points along the lines in each chosen direction starting at the current best point. Several such line searches are packaged together into a stochastic line search (multi-line search later), 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 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 gradients encountered.

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

$$f(x+p) - f(x) = g(x)^T p + \frac{1}{2}\gamma \|p\|^2, \tag{4}$$

where γ depends on x and p and satisfies one of

$$|\gamma| \leq L, \quad \text{(general case)} \tag{5}$$

$$0 \leq \gamma \leq L, \quad \text{(convex case)} \tag{6}$$

$$0 < \sigma \leq \gamma \leq L. \quad \text{(strongly convex case)} \tag{7}$$

In all three cases,

$$g(x)^T p - \frac{1}{2}L\|p\|^2 \leq f(x+p) - f(x) \leq g(x)^T p + \frac{1}{2}L\|p\|^2. \tag{8}$$

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

$$r_0 := \sup\left\{\|x - \widehat{x}\| \mid f(x) \leq f(x_0)\right\} < \infty. \tag{9}$$

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

Proposition 1 *Let $x, p \in \mathbb{R}^n$ and $\Delta \geq 0$. Then (A1) implies that*

$$L \geq \frac{|f(x+p) + f(x-p) - 2f(x)|}{\|p\|^2}, \tag{10}$$

and at least 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) $|g^T p| \leq \Delta + \frac{1}{2}L\|p\|^2$.

Proof Taking the sum of (8) and the formula obtained from it by replacing p with $-p$ gives (10).

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 (4) with $\mp p$ in place of p,

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

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 1 will play a key role in the construction of our stochastic line search **SLS** detailed in Subsection 2.3:

• It exploits the well-known (Evtushenko [19], Pintér [45], Kvasov & Sergeyev [33]) lower bound (10) 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 neither (i) nor (ii) holds then (iii) holds, giving a useful upper bound for the directional derivative.

In particular, this allows us to prove statements about the true gradient even though our algorithm never calculates one.

2.2 Random search directions

Random directions are independent components uniformly distributed in $[-\frac{1}{2}, \frac{1}{2}]$, computed by

$$p = \text{rand}(n, 1) - 0.5, \tag{11}$$

where rand generates a uniformly distributed random vector. The scaling of these 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 (22) (defined in Theorem 1(ii) below). For fixed Δ, the scale-dependent factor $\Gamma(\delta) = L\delta + 2\Delta/\delta$ (defined by (23) in Theorem 1 below) is smallest for the choice

$$\widehat{\delta} = \sqrt{2\Delta/L}. \tag{12}$$

However, in practice, L is unknown and we replace later it by the approximation λ form an extrapolation step (called **extrapolationStep** below). Proposition 1 implies that

$$\lambda_0 \le \lambda \le \max(\lambda_0, L) \le \lambda_0 + L, \tag{13}$$

where λ_0 is the initial value of λ. Moreover, we safeguard δ by enforcing sensible positive lower and upper bounds by

$$\delta = \max\left(\delta_{\min}, \min\left(\sqrt{\alpha_e \gamma_\delta \Delta/\lambda}, \delta_{\max}\right)\right), \tag{14}$$

where $\gamma_\delta > 0$, $0 < \delta_{\min} < \delta_{\max} < +\infty$, and α_e is the step size generated by extrapolation steps (discussed later). Then we are interested in scaling the random directions by

$$p = p(\delta/\|p\|). \tag{15}$$

The following variant of the angle condition (16) plays a key role to get our complexity bounds.

Proposition 2 *For random search directions generated by* (11) *and scaled by* (15) *satisfies* $\|p\| = \delta$ *and, with probability* $\ge \frac{1}{2}$,

$$\|g(x)\|_* \|p\| \le 2\sqrt{cn}|g(x)^T p| \tag{16}$$

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 after scaling and give $c = c_0/4 \approx 4/7$. □

2.3 A stochastic line search

In this section, we construct a stochastic line search algorithm, called **SLS**. It polls in some random directions (satisfying (16), with probability $\ge \frac{1}{2}$, generated by (11), and scaled by (15)) in a line search fashion a few objective function values each in the hope of reducing the objective by more than a multiple of Δ.

2.3.1 An extrapolation step

We discuss how to construct an extrapolation step, called **extrapolationStep**, trying to find a good decrease in the function value.

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 an extrapolation step, we always use `xm` for the best point found, and rescale p such that the next evaluation is always at $\texttt{xm} + p$ and a former third evaluation point is at $\texttt{xm} - p$. The function values immediately after the next evaluation are then

$$\texttt{fl} := f(\texttt{xm} - p), \quad \texttt{fm} := f(\texttt{xm}), \quad \texttt{fr} := f(\texttt{xm} + p). \tag{17}$$

At this stage, we can compute the lower bound

$$\lambda := \max(\lambda_{\text{old}}, |\texttt{fl} + \texttt{fr} - 2\texttt{fm}|/\delta^2) \tag{18}$$

for the Lipschitz constant, valid by (10). Note that the initial λ_{old} is the tuning parameter $\lambda_{\max}$, however, it is updated by **extrapolationStep** and estimated by a heuristic formula (see Section 5.4).

Afterwards, whenever $\texttt{fr} < \texttt{fm}$, the best point is updated by overwriting $\texttt{xm} + p$ over $\texttt{xm}$, with the consequence that in this case

$$\texttt{fl} := f(\texttt{xm} - 2p), \quad \texttt{fm} := f(\texttt{xm} - p), \quad \texttt{fr} := f(\texttt{xm}). \tag{19}$$

Before describing an extrapolation step, we need to define some variables. As defined earlier, let R be the number of the random search directions used in **SLS** and **a** be a list of R *extrapolation step sizes*. All components of the initial **a** are one; each of its components is expanded or reduced according to whether the function value is reduced or not. Let `nE` be the number of iterations generated by **extrapolationStep** to exceed a decrease in the function value. If the counter `nE` stays zero, **extrapolationStep** cannot improve the function value; in this case it is called *unsuccessful*. Otherwise, the function value is decreased and **extrapolationStep** is called *successful*. There are some boolean variables which need to be defined as follows:

• `ext` (a large gain found) and `sext` (a sequence large gain found) are used to identify whether there is a considerable decrease in the function value ($\texttt{ext} = 1$) or not ($\texttt{ext} = 0$).

• `opp` determines whether an opposite direction is tried ($\texttt{opp} = 1$) or not ($\texttt{opp} = 0$).

• `good` takes 0 (the best point cannot be updated) and 1 (the best point is updated). In other words, if `good` is zero, **extrapolationStep** is unsuccessful; otherwise, **extrapolationStep** is successful.

• `done` takes 0 if none of maximum time in seconds and maximum number of function evaluations is reached; otherwise, it takes 1.

t is a counter for R taking $0, \cdots, R$. It does not change inside **extrapolationStep**, but it is updated later outside **extrapolationStep** (inside **SLS**).

We must be careful to make sure that the estimation of the Lipschitz constant is correct in (S.2) of the next algorithm, especially when an extrapolation step – improving the function value – is tried. This estimation is obtained when either the extrapolation step is performed (the first iteration was tried and the second iteration may be tried) or the opposite direction is tried. Let `fe` be the function value improved by an extrapolation step in each iteration. Instead of the previous best function value `fm`, `fe` must be used to estimate the Lipschitz constant λ by (18). In this case, the previous best function value `fm` is restored in $f_{\texttt{temp}}$. Then, after we estimate λ by (18), `fm` is replaced by $f_{\texttt{temp}}$. Finally, whenever the extrapolation step ends up, `fm` is replaced by `fe` and hence the best point and its function value are updated.

Denote the step sizes used in **extrapolationStep** by α_e. **extrapolationStep** first takes the initial step size $\alpha_e = 1$, which is necessary to estimate the Lipschitz constant and get an upper bound in Theorem 1(ii), and then takes the second step size from $\mathbf{a}_t$ while expanding it until the function value is decreased. We consider three cases: (i) the function value is improved; in this case, the extrapolation step ends up, $\mathbf{a}_t$ is replaced by the step size of the best point and the best point is updated ($\texttt{good} = 1$), (ii) the function value cannot decrease in the first iteration; an opposite direction is tried. Then either an extrapolation along an opposite direction is tried ($\texttt{good} = 1$) or **extrapolationStep** ends up without any improvement on the function value ($\texttt{good} = 0$). In this case, $\mathbf{a}_t$ does not update, but it is reduced later inside **SLS**, (iii) either maximum time in seconds (`secmax`) or maximum number function evaluations (`nfmax`) is reached, resulting in $\texttt{done} = 1$. In this case, **extrapolationStep** ends up.

Throughout the paper, == is the comparison operator for equality, $\texttt{length}(V)$ computes the length of the vector V, $\text{ones}(n, 1)$ generates a $n \times 1$ vector whose entries are 1, $\text{zeros}(n, 1)$ generates a $n \times 1$ vector whose entries are zero, `cputime` computes time in seconds and $A_{:k}$ denotes the kth column of a matrix A.

Algorithm 1 *(extrapolation step (***extrapolationStep***))*

Input*:* **fun** *(the function handle),* `xm` *and* `fm` *(the old best point and its function value), p (the search direction), $\Delta > 0$ (the threshold for good improvement), $\lambda \geq 0$ (approximation for the Lipschitz constant),* $\mathbf{a}_t$ *(the tth extrapolation step size),* `nf` *(the number of function evaluations),* `secinit` *(initial time in seconds),* `nfmax`

(maximum number of function evaluations), and `secmax` *(maximum time is seconds).*
Tuning parameters*:* $\gamma_e > 1$ *(the factor for extrapolation test) and* $E \geq 1$ *(maximum number of extrapolations).*
Output*:* `xm` *and* `fm` *(the best point and its function value),* λ *(the estimated Lipschitz constant),* `good` *(sufficiently improved function value?),* `nf` *(the number of function evaluations),* `done` *(*`secmax` *or* `nfmax` *reached?), and* $\mathbf{a}_t$ *(the* t*th extrapolation step size).*
(S.0) Set $\texttt{nE} = 0$, $\alpha_e = 1$, *and* $\texttt{opp} = 0$.
(S.1) Set $\texttt{xr} = \texttt{xm} + \alpha_e p$ *and compute* $\texttt{fr} = \mathbf{fun}(\texttt{xr})$, $\texttt{nf} = \texttt{nf} + 1$, $\texttt{df} = \texttt{fm} - \texttt{fr}$, *and*
$\texttt{done} = (\texttt{nf} == \texttt{nfmax} \;\; or \;\; \texttt{cputime} - \texttt{secinit} \geq \texttt{secmax})$. *Then*
if `done`
if $\texttt{df} > 0$, $\texttt{xm} = \texttt{xr}$; $\texttt{fm} = \texttt{fr}$; **end**;
break;
end;
(S.2) Estimate λ *either* `opp` *is true or* $\texttt{nE} = 1$*:*
if `opp`, *estimate* λ *by* (18); **end**;
if $\texttt{nE} == 1$, *set* $f_{\texttt{temp}} = \texttt{fm}$ *and* $\texttt{fm} = \texttt{fe}$, *estimate* λ *by* (18), *and set* $\texttt{fm} = f_{\texttt{temp}}$; **end**;
(S.3) Check whether a large gain is found or not:
identify $\texttt{ext} = (\texttt{df} > \min(\alpha_e, 1)\Delta)$ *and* $\texttt{sext} = (\texttt{nE} < E \;\&\; \texttt{ext})$ *and then:*
if `sext` *% a large gain or a sequence large gain*
update $\texttt{nE} = \texttt{nE} + 1$;
if $\texttt{nE} == 1$, *set* $\texttt{fl} = \texttt{fm}$ *and* $\alpha_e = \mathbf{a}_t$; **else**, *expand the step size by* $\alpha_e = \gamma_e \alpha_e$; **end**;
set $\texttt{fe} = \texttt{fr}$ *and then go to (S.1).*
elseif *(*$\texttt{nE} == 0$ *&* $\texttt{opp} == 0$*) % extrapolation along an opposite direction; maybe*
set $p = -p$, $\texttt{fl} = \texttt{fr}$ *and* $\texttt{opp} = 1$*; then go to (S.1).*
elseif $\texttt{nE} \geq 1$ *% end of the extrapolation step*
update $\alpha_e = \alpha_e / \gamma_e$ *and* $\texttt{xm} = \texttt{xm} + \alpha_e p$. *Then set* $\texttt{good} = 1$, $\mathbf{a}_t = \alpha_e$, *and* $\texttt{fm} = \texttt{fe}$.
break; *% a good decrease in the function value found*
else *% no gain*
break; *% extrapolation step was unsuccessful*
end

2.3.2 The **SLS** *algorithm*

For each random direction generated, a stochastic line search, called **SLS**, using **extrapolationStep** 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.
- If no gain is found, the step size is reduced.

Algorithm 2 *(A stochastic line search (***SLS***))*

Input*:* `xm` *and* `fm` *(the old best point and its function value),* $\Delta > 0$ *(the threshold for good improvement),* $\lambda \geq 0$ *(approximation for the Lipschitz constant),* `nf` *(the number of function evaluations),* `secinit` *(initial time in seconds),* `nfmax` *(maximum number of function evaluations),* `secmax` *(maximum time is seconds),* $\mathbf{a}$ *(the list of extrapolation step sizes), and* **fun** *(the function handle).*
Tuning parameters*:* $\gamma_e > 1$ *(the factor for extrapolation test),* $E \geq 1$ *(maximum number of extrapolations),* $\delta_{\min}/\delta_{\max}$ *(minimum/maximum norm of trial steps),* γ_λ *(factor for adjusting* δ*),* $\alpha_{\min} \in (0, 1)$ *(minimum threshold for step sizes), and* $R > 0$ *(the number of random search directions).*
Output*:* `xm` *and* `fm` *(the best point and its function value),* $\lambda \geq 0$ *(approximation for the Lipschitz constant),* `done` *(*`secmax` *or* `nfmax` *reached?),* `nf` *(the number of function evaluations), and* $\mathbf{a}$ *(the updated list of extrapolation step sizes).*
(S.0) Set $t = 0$ *and* $\texttt{good} = 0$.
(S.1) Replace t *by* $t + 1$, *compute the random direction by* (11) *and* δ *by* (14). *Then scale it by* (15).
(S.2) Perform **extrapolationStep** *to get a decrease in* f *and the stopping variable* `done`*:*
if *(*`done` *is true),* **break**; **end**;

(S.3) If there is no decrease in f, $\texttt{nE} = 0$, reduce the tth step size by

$$\mathbf{a}_t = \max(\mathbf{a}_t/\gamma_e, \alpha_{\min}). \tag{20}$$

(S.4) **If** *($t == R$),* **break***;* **else***, go to (S.1);* **end***;*

In (20), the extrapolation step sizes need to be reduced whenever there is no decrease in the function value. Hence, they need to be controlled by the tuning parameter $\alpha_{\min}$.

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

Theorem 1 *Assume that (A1) holds and let* $\texttt{nf}$ *be the counter for the number of function evaluations, R be the number of random search directions, and Δ_f be the improvement on the function value in* **SLS***. Moreover, let $\overline{\alpha} := \min(\alpha_e, 1)$ and $\overline{\Delta} := \overline{\alpha}\Delta$, where α_e is the step size generated by* **extrapolationStep***.*

(i) f decreases by at least

$$\overline{\Delta}_f := \overline{\Delta}\max(\texttt{nf} - 2R - 1, 0) \tag{21}$$

(Note that $\overline{\Delta}_f$ may be zero, catering for the case of no strict decrease).

(ii) Suppose that $0 < \eta < 1$ and $R \geq \log_2 \eta$. If f did not decrease by more than a multiple of 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

$$\|g\|_* \leq \sqrt{cn}\Gamma(\delta), \tag{22}$$

where c is the constant in Proposition 2 and

$$\Gamma(\delta) := L\delta + \frac{2\Delta}{\delta}. \tag{23}$$

Proof (i) Clearly, the function value of the best point does not increase. Thus (i) holds if $\texttt{nf} - 2R - 1 \leq 0$. If this is not the case, then $\texttt{nf} \geq 2R + 2$. But in the while loop of **SLS**, R directions p are generated and at most two function values are computed, unless an extrapolation step is performed. In the latter case, at least $\texttt{nf} - 2R - 1$ additional function values are computed during the extrapolation stage, each time with a gain of at least $\overline{\Delta}$. Thus the total gain is at least (21).

(ii) Assume that f did not decrease by more than $\overline{\Delta}$. For $t = 1, \ldots, R$, let p_t be the tth random direction generated by (15), and let x_t be the best point obtained before searching in direction p_t. Then, from the definition of $\texttt{ext}$ in (S.3) of Algorithm 1 and Proposition 1, we get

$$|g(x_t)^T p_t| \leq \overline{\Delta} + \frac{L}{2}\|p_t\|^2 \leq \Delta + \frac{L}{2}\|p_t\|^2 = \frac{\delta}{2}\Gamma(\delta), \quad \text{for all } t = 1, \ldots, R.$$

Since the random direction was generated by (11), Proposition 2 implies that

$$\|g(x_t)\|_* = \|g(x_t)\|_*\|p_t\|/\delta \leq \left(2\sqrt{cn}|g(x_t)^T p_t|\right)/\delta \leq \sqrt{cn}\Gamma(\delta), \quad \text{for all } t = 1, \ldots, R,$$

holds with probability $\frac{1}{2}$ or more. Therefore $\|g(x_t)\|_* \leq \sqrt{cn}\Gamma(\delta)$ fails with a probability $\Pr_t < \frac{1}{2}$ for all $t = 1, \ldots, R$. Therefore, the probability Pr that (22) holds for at least one of the gradients $g = g(x_t)$ ($t \in \{1, \ldots, R\}$) is $\Pr = 1 - \prod_{t=1}^{R-1} \Pr_t \geq 1 - 2^{-R}$. □

3 A stochastic descent algorithm for BBO

In this section, we first consider a fixed decrease search for which an upper bound of the gradient norm for at least one of the gradients encountered or of the total number of function evaluations is found. Then the primary version of our algorithm is given.

3.1 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 stochastic line search **SLS** in order to improve the function value each time by a multiple of Δ.

Algorithm 3 *(A fixed decrease search (***FDS***))*

Input*:* `xm` *and* `fm` *(the old best point and its function value),* $\Delta > 0$ *(the threshold for good improvement),* $\lambda \geq 0$ *(approximation for the Lipschitz constant),* `secinit` *(initial time in seconds),* `nfmax` *(maximum number of function evaluations),* `secmax` *(maximum time is seconds),* **a** *(the list of extrapolation step sizes),* `nf` *(the number of function evaluations), and* **fun** *(the function handle).*
Tuning parameters*:* $\gamma_e > 1$ *(the factor for extrapolation test),* $E \geq 1$ *(maximum number of extrapolations),* $\delta_{\min}/\delta_{\max}$ *(minimum/maximum norm of trial steps),* γ_λ *(factor for adjusting* δ*),* $\alpha_{\min} \in (0,1)$ *(minimum threshold for step sizes), and* $R > 0$ *(the number of random search directions).*
Output*:* `xm` *and* `fm` *(the best point and its function value),* $\lambda \geq 0$ *(approximation for the Lipschitz constant),* `done` *(*`secmax` *or* `nfmax` *reached?),* `nf` *(the number of function evaluations), and* **a** *(the updated list of extrapolation step sizes).*
(S.0) Perform **SLS** *to get a decrease in* f *and the stopping variables* `done` *and* `good`*.*
(S.1) **if** *(*`done` *is true or* `good` *is false),* **break***;* **else***, go to (S.0);* **end***;*

Theorem 2 *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 iterations of* **FDS***, the initial value of* f *and the minimal function value, respectively. Then:*

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

$$(2R+1)\min(\frac{\Delta_f}{\overline{\Delta}},1)N \leq (2R+1)N \leq (2R+1)\frac{f_0-\widehat{f}}{\alpha_{\min}\Delta},$$

where $\alpha_{\min} \in (0,1)$ *and* $\overline{\Delta} = \min(\alpha_e,1)\Delta$*.*

(ii) Given $\eta_1 = 2^{-R}$ *and* $0 < \delta_{\min} < \delta_{\max} < \infty$*, with probability* $\geq 1-\eta_1$*,*

$$\|g\|_* \leq \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta_{\max}}\Big) \tag{24}$$

holds for at least one of the gradients g *encountered. Here* c *is the constant from Proposition 2 and, if* λ_0 *denotes the value of* λ *before the first execution of* **FDS***,*

$$L' := \frac{L^2\gamma_\delta}{\lambda_0} + 4L + 4\frac{\lambda_0+L}{\gamma_\delta}. \tag{25}$$

Proof By (A2), $\widehat{f}$ is finite. Denote by f_k the result of the kth execution of **FDS**. By the definition of `good`, $f_k \leq f_{k-1} - \min(1,(\alpha_e)_k)\Delta$ for all but the last value of k. We then conclude that

$$\widehat{f} \leq f_N \leq f_0 - \sum_{i=1}^{N}\min((\alpha_e)_i,1)\Delta \leq f_0 - N\alpha_{\min}\Delta,$$

by (20), so that $N \leq (f_0-\widehat{f})/(\alpha_{\min}\Delta)$.

If $\Delta_f < \overline{\Delta}$, then each iteration uses at most $2R+1$ function evaluations because **SLS** is performed in each iteration (In **SLS**, each direction p is generated and at most two function values are computed, unless an extrapolation step is performed with one function evaluation). Otherwise, it uses at most $(2R+1)\Delta_f/\overline{\Delta}$ function evaluations; hence (ii) follows.

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

$$\Gamma(\delta) \leq L\delta_{\min} + \sqrt{L'\Delta} + \frac{2\Delta}{\delta_{\max}}. \tag{26}$$

By the definition of δ in (14), 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

$$\Lambda := L\sqrt{\frac{\gamma_\delta}{\lambda}} + 2\sqrt{\frac{\lambda}{\gamma_\delta}}. \tag{27}$$

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 (26) follows since by (13) and (27), $\Lambda^2 = \frac{L^2\gamma_\delta}{\lambda} + 4L + \frac{4\lambda}{\gamma_\delta} \leq L'$. □

3.2 The basic **VSBBO** algorithm

We now have all ingredients to formulate the basic version of *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 either Δ is below a minimum threshold or `done` is true, the algorithm stops.

Algorithm 4 *(***Vienna stochastic black box optimization (VSBBO)***)*

Input*: x (initial point) and* **fun** *(the function handle),* `nfmax` *(maximum number of function evaluations), and* `secmax` *(maximum time is seconds).*
Tuning parameters*: $\gamma_e > 1$ (the factor for extrapolation test), $E \geq 1$ (maximum number of extrapolations), $\delta_{\min}/\delta_{\max}$ (minimum/maximum norm of trial steps), γ_λ (factor for adjusting δ), $Q > 1$ (factor for adjusting Δ), $\Delta_{\min}$ (minimal threshold for Δ), $\Delta_{\max}$ (maximum threshold for Δ), $\alpha_{\min} \in (0,1)$ (minimum threshold for step sizes), $\lambda_{\max}$ (the initial Lipschitz constant), and $R > 0$ (the number of random search directions).*
Output*:* `xm`*,* `fm` *(the best point and its function value), and* `nf` *(the number of function evaluations).*
(S.0) Set `secinit` = `cputime`*,* `done` $= 0$*, $\lambda = \lambda_{\max}$, $\delta = \delta_{\max}$, $\Delta = \Delta_{\max}$, and* `xm` $= x$*. Then compute* `fm` $=$ **fun**(`xm`) *and set* `nf` $= 1$*.*
(S.1) Perform **FDS** *to update the best point and to get the stopping variable* `done` *(by* **extrapolationStep***).*
(S.2) **if** *($\Delta \leq \Delta_{\min}$ or* `done`*),* **break***;* **end***;*
(S.3) Reduce $\Delta = \Delta/Q$ and go to (S.1).

4 Complexity analysis of VSBBO

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

4.1 The general (nonconvex) case

Proposition 3 *Assume that (A1) and (A2) hold. Moreover, let f_0 be the initial and $\widehat{f}$ the optimal function value. If $\Delta_{\min} > 0$, the number of function evaluations needed up to iteration k by* **VSBBO**, *started at x_0, is*

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

Proof By construction, the kth call to **FDS** uses $\Delta = Q^{1-k}\Delta_{\max}$, hence needs by Theorem 2(i) at most

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

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

$$n_k \le \sum_{j=1}^{k}(2R+1)\frac{f_0 - \widehat{f}}{\alpha_{\min}Q^{1-j}\Delta_{\max}} = (2R+1)\frac{Q(f_0 - \widehat{f})}{\alpha_{\min}(Q-1)\Delta_{\min}}$$

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

Theorem 3 *Assume that (A1) and (A2) hold. With c from Proposition 2, let $\zeta \ge 9c$, $0 < \eta < 1$, let K be a positive integer, and let $\varepsilon > 0$. If the parameters are chosen such that*

$$\frac{\varepsilon^2}{\zeta L' n} \le \Delta_{\min} \le \frac{\varepsilon^2}{9cL'n}, \tag{28}$$

$$K \ge 1 - \log_2 \frac{\Delta_{\min}}{\Delta_{\max}}, \tag{29}$$

$$R \ge \log_2 \frac{K+1}{\eta}, \tag{30}$$

$$0 \le \delta_{\min} \le \frac{\varepsilon}{3L\sqrt{cn}}, \quad \frac{2\varepsilon}{3L'\sqrt{cn}} \le \delta_{\max} \le \infty, \tag{31}$$

then Algorithm 4 finds after at most $\mathcal{O}(nR\varepsilon^{-2})$ function evaluations with probability $\ge 1-\eta$ a point x with

$$\|g(x)\|_* \le \varepsilon. \tag{32}$$

Proof By the rule of updating Δ in **VSBBO**, $\Delta_k = Q^{1-k}\Delta_{\max} \le \Delta_{\min}$ for $k \ge K$. Hence at most K steps of **FDS** are performed. By (30), we have $\eta_1 = 2^{-R} \le \eta/(K+1)$. Thus by Theorem 2(ii), we have, with probability $\ge 1-(K+1)\eta_1 \ge 1-\eta$, for at least one of the gradients g encountered,

$$\|g\|_* \le \min_{j=0:K} \Gamma(\delta_j) \le \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\Delta_{\min}} + \frac{2\Delta_{\min}}{\delta_{\max}}\Big) \le \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} = \varepsilon. \tag{33}$$

Here we used (28) and (31). Moreover, by Proposition 3 and (28),

$$\begin{aligned} n_K &\le (2R+1)\frac{Q(f_0 - \widehat{f})}{\alpha_{\min}(Q-1)\Delta_{\min}} \\ &\le (2R+1)\frac{\zeta L' n Q}{\alpha_{\min}(Q-1)\varepsilon^2}(f_0 - \widehat{f}) = \mathcal{O}\Big(nR\varepsilon^{-2}\Big). \end{aligned}$$

□

Choosing $\Delta_{\min} = \mathcal{O}(\varepsilon)$ with (28) is possible since $\zeta \ge 9c$, and k, R, $\delta_{\min}$, $\delta_{\max}$ can clearly be chosen to satisfy (29)–(31) and displays

$$K = \mathcal{O}(\log \frac{n}{\varepsilon^2}), \quad R = \mathcal{O}(\log \frac{n}{\varepsilon^2} + \log \eta^{-1}).$$

4.2 The convex case

Theorem 4 *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. With c from Proposition 2, let $\zeta \geq 9c$, $0 < \eta < 1$, and let K be a positive integer. For any $\varepsilon > 0$, if (30)–(31) hold then* **VSBBO** *finds after at most $\mathcal{O}\Big(nR\varepsilon^{-1}\Big)$ function evaluations with probability $\geq 1-\eta$ a point x with*

$$\|g\|_* \leq \varepsilon, \quad f - \widehat{f} \leq \varepsilon r_0, \tag{34}$$

where r_0 is given by (9).

Proof By (A2), f has a minimizer $\widehat{x}$ and $r_0 < \infty$. By Theorem 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 (32); hence $k-1 \leq K$. The convex case is characterized by (6), so that

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

We know from Theorem 3 that, with probability $\geq 1-\eta$, (32) holds and hence

$$f_{k-1} - \widehat{f} \leq g_{k-1}^T(x_{k-1} - \widehat{x}) \leq \|g_{k-1}\|_* \|x_{k-1} - \widehat{x}\| \leq \varepsilon r_0 \tag{35}$$

by (9). (34) now follows from (32) and (35). 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 **SLS** at kth execution of **FDS**, so that

$$f_k \leq f_k^{N_k} - \alpha_{\min}\Delta_k \leq f_k^{N_k - 1} - 2\alpha_{\min}\Delta_k \leq \ldots \leq f_{k-1} - N_k\alpha_{\min}\Delta_k, \tag{36}$$

where $f_{k-1} := f_k^0$ and $f_k := f_k^{N_k+1}$. From (35), (36) and the rule of updating Δ in **VSBBO**, we find

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

leading to

$$N_k \leq Q^{1-k}\frac{\varepsilon r_0}{\alpha_{\min}\Delta_{\min}} \leq Q^{1-k}\frac{r_0 c\zeta L' n}{\alpha_{\min}\varepsilon} \tag{37}$$

by (28). The bound for the number of function evaluations is now obtained from Theorem 2 and (37):

$$\begin{aligned} n_K &\leq (2R+1)\sum_{j=1}^{K} N_j \leq (2R+1)\frac{r_0 c\zeta L' n}{\alpha_{\min}\varepsilon}\sum_{j=1}^{K} Q^{1-j} \\ &\leq (2R+1)\frac{Qr_0 c\zeta L' n}{\alpha_{\min}\varepsilon(Q-1)} = \mathcal{O}\Big(nR\varepsilon^{-1}\Big). \end{aligned}$$

□

4.3 The strongly convex case

Theorem 5 *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. Under the assumptions of Theorem 3,* **VSBBO** *finds after at most $\mathcal{O}\Big(nR\log\varepsilon^{-1}\Big)$ function evaluations with probability $\geq 1-\eta$ a point x with*

$$\|g\|_* \leq \varepsilon, \quad f - \widehat{f} \leq \frac{\varepsilon^2}{2\sigma}, \quad \|x - \widehat{x}\| \leq \frac{\varepsilon}{\sigma^2}. \tag{38}$$

Proof By Theorem 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 (32); hence $k-1 \le K$. The strongly convex case is characterized by (7), so that f has a minimizer $\widehat{x}$ and

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

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 $f(y) \ge f(x) - \dfrac{1}{2\sigma}\|g(x)\|_*^2$ for $y \in \mathcal{L}(x_0)$. Therefore

$$\widehat{f} \ge f(x) - \frac{1}{2\sigma}\|g(x)\|_*^2. \tag{39}$$

The replacement of x by x_{k-1} in (39) and (32) gives, with probability $\ge 1-\eta$,

$$f_{k-1} - \widehat{f} \le \frac{\|g_{k-1}\|_*^2}{2\sigma} \le \frac{\varepsilon^2}{2\sigma}. \tag{40}$$

Since the gradient vanishes at the optimal point, we get from Theorem 3 and (40)

$$\|\widehat{x} - x_{k-1}\|^2 \le \frac{2}{\sigma}(f_{k-1} - \widehat{f}) \le \frac{\varepsilon}{\sigma^2} \tag{41}$$

with probability $\ge 1-\eta$. By the rule of updating Δ in **VSBBO**, we may use (36), where N_k+1 is the number of steps performed by **SLS** at the kth execution of **FDS**. By (40),

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

so that

$$N_k \le \frac{\varepsilon^2}{2\sigma\alpha_{\min}\Delta_{\min}} \le \frac{c\zeta L' n}{2\alpha_{\min}\sigma} \tag{42}$$

by (28). Now Theorem 2 implies

$$\begin{aligned} n_K &\le (2R+1)\sum_{j=1}^{K} N_j \le (2R+1)\frac{c\zeta L' n}{2\alpha_{\min}\sigma}K \\ &= (2R+1)\frac{c\zeta L' n}{2\alpha_{\min}\sigma}\Big(1 + \log_Q \frac{\Delta_{\max}}{\Delta_{\min}}\Big) \\ &= (2R+1)\frac{c\zeta L' n}{2\alpha_{\min}\sigma}\Big(1 + \log_Q \frac{c\zeta L' n \Delta_{\max}}{\varepsilon^2}\Big) = \mathcal{O}\Big(nR\log\varepsilon^{-1}\Big). \end{aligned}$$

□

5 Some new heuristic techniques

In this section, we describe several heuristic techniques that improve the basic Algorithm 4. While only convergence to a local minimizer is guaranteed, the addition of these heuristic techniques results in an efficient global solver. Indeed, in our comprehensive numerical experiment, reported in Section 7, the resulting **VSBBO** algorithm found in most cases either a global minimizer or, where this could not be checked, at least a point of similar quality with the best competitive global solvers.

More specifically, we discuss the occasional use of alternative search directions (two cumulative directions and a random subspace direction) and heuristics for estimating key parameters unspecified by the general theory – the initial desired gain, the Lipschitz constant, and the scaling vector. Moreover, we discuss the use of approximate gradients estimated by finite differences with steps extracted from the extrapolation steps. In Section 6, we combine Algorithm 3.3 with these heuristic techniques, resulting in the global solver **VSBBO**.

5.1 Cumulative directions

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

(i) The first cumulative direction is model independent, computed by

$$p = x - x_{\text{init}}, \tag{43}$$

where x is the best point and x_{init} the initial point of the current improved version of **SLS**. 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

$$f\Big(x + \sum_{i\in I} \alpha_i p_i\Big) \approx f(x) - \sum_{i\in I} e_i(\alpha_i) \tag{44}$$

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 an improved version of **SLS**.

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 $e(\alpha) := \dfrac{\alpha}{2}(d - \alpha h)$. Let us recall the function values `fl`, `fm`, and `fr` satisfying either (17) or (19). If $\texttt{fr} < \texttt{fm}$, the last evaluated point was the best one, so $\texttt{fr} \le \min(\texttt{fl}, \texttt{fm})$. In this case, (19) holds and we have

$$d := 4\texttt{fm} - 3\texttt{fr} - \texttt{fl}, \tag{45}$$

and

$$h := \texttt{fr} + \texttt{fl} - 2\texttt{fm}. \tag{46}$$

Otherwise, the last evaluated point was not the best one, so $\texttt{fm} \le \min(\texttt{fl}, \texttt{fr})$. In this case, (17) holds and we compute d by

$$d := \texttt{fl} - \texttt{fr} \tag{47}$$

and h by (46).

Given the tuning parameter $a > 0$, the minimizer of the quadratic interpolant restricted to the interval $[-a, a]$ is

$$\alpha = \begin{cases} a & \text{if } d \ge 0, \\ -a & \text{if } d < 0 \end{cases} \tag{48}$$

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

$$\alpha = \begin{cases} \min(a, d/2h) & \text{if } d \ge 0, \\ \max(-a, d/2h) & \text{if } d < 0. \end{cases} \tag{49}$$

Assuming the validity of the quadratic model (44), we find the model optimizer by additively accumulating the estimated steps αp and gains e into a cumulative step q with anticipated gain r.

5.2 Random subspace direction

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

$$\alpha_{\text{rand}} := \text{rand}(m_s - 1, 1) - 0.5, \quad \alpha_{\text{rand}} := \alpha_{\text{rand}} / \|\alpha_{\text{rand}}\|, \quad p := \sum_{i=1, i\ne b}^{m_s} (\alpha_{\text{rand}})_i (X_{:i} - X_{:b}) \tag{50}$$

where X is the list of the m_s best points found so far and b is the index best point.

5.3 Choosing the initial Δ

Let F be a list of function values whose points are restored in X. First of all, we compute

$$\mathtt{dF} = \underset{i=1:m}{\text{median}}\, |F_i - F_b|. \tag{51}$$

Then the initial desired gain is estimated by

$$\Delta := \begin{cases} \Delta_{\max} & \text{if } \mathtt{dF} = 0, \\ \gamma_{\max} \min(\mathtt{dF}, 1) & \text{otherwise,} \end{cases} \tag{52}$$

where $\Delta_{\max} > 0$, the initial gain, and $\gamma_{\max} > 0$ are the tuning parameters.

5.4 Choosing the initial λ

The initial value for λ is $\lambda_{\max}$ which is the tuning parameter, however, it is updated by (18) provided that the best point is updated by **extrapolationStep**. Our aim is to estimate it by a heuristic formula based on the previous best function values restored in F.

Let λ_{old} be the old estimation for the Lipschitz constant and $\gamma_\lambda > 0$ be a factor for adjusting λ. We would compute λ by

$$\lambda := \begin{cases} \gamma_\lambda / \sqrt{n} & \text{if } \mathtt{dF} = 0 \text{ and } \lambda_{\text{old}} = 0, \\ \lambda_{\text{old}} & \text{if } \mathtt{dF} = 0 \text{ and } \lambda_{\text{old}} \neq 0, \\ \gamma_\lambda \sqrt{\mathtt{dF}/n} & \text{otherwise,} \end{cases} \tag{53}$$

where $\mathtt{dF}$ is computed by (51).

5.5 Choosing the scaling vector

The idea is to estimate a sensible scaling vector s with the goal of adjusting the search direction scaled by (15). We first compute

$$\mathtt{dX}_{:i} = X_{:i} - X_{:b}, \quad \text{for all } i = 1, \cdots, m_s$$

Then s is estimated by the following formula

$$s := \sup_{i=1:m_s} (\mathtt{dX}_{:i}), \quad J = \{i \mid s_i = 0\}, \quad s_J = 1. \tag{54}$$

Finally, the formula (15) is rewritten as

$$p = s \circ p \quad \text{and} \quad p = p(\delta / \|p\|) \tag{55}$$

where $\circ$ denotes componentwise multiplication and δ is computed by (14).

5.6 An estimation for the gradient

Finite difference quasi-Newton methods estimate the gradient with components

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

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

$$\alpha_i := \max\{1, \|x\|_\infty\} \sqrt{\varepsilon_m}, \tag{56}$$

where ε_m is the machine precision; another choice for α is made now. After generating each coordinate search direction, we estimate each component of gradient in a way that is a little different from the forward finite difference approach. The step size generated by **extrapolationStep** is used instead of the general choice (56). The reason of this change is that we don't need to estimate the gradient by another algorithm due to the additional cost. Let describe how to compute the gradient. If **extrapolationStep** cannot decrease the function value in the tth iteration (`nE` $= 0$), `fr` is computed and $\mathbf{a}_t$ is unchanged. Given the old best point f_{init}, the tth component of the gradient is computed by

$$\widehat{g}_{t:} = (\texttt{fr} - f_{\text{init}})/\mathbf{a}_t; \tag{57}$$

otherwise, it is computed by

$$\widehat{g}_{t:} = (\texttt{fm} - f_{\text{init}})/\mathbf{a}_t, \tag{58}$$

where both `fm` and $\mathbf{a}_t$ are updated by **extrapolationStep**. We will add later this computation to an improved version of **SLS**.

6 The implemented version of VSBBO

In this section, we discuss the improved version of the basic algorithm, the *Vienna stochastic black box optimization algorithm* (**VSBBO**). The improvements are of a heuristic nature, very important for efficiency, and do not change the complexity results. Thus **VSBBO** 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 7 show that, in practice, **VSBBO** matches the quality of all state of the art algorithms for BBO. **VSBBO** is implemented in Matlab; the source code is obtainable from

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

It includes many subalgorithms described earlier in Sections 2–4. The others have a simpler structure; hence we skip their details (which can be found at the above website) and only state their goals and those tuning parameters which have not been defined yet. These subalgorithms are **identifyDir**, **lbfgsDir**, **updateSY**, **updateXF**, **updateCum**, **enforceAngle**, **direction**, **MLS**, and **setScales**.

VSBBO initially 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 a multiple of 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 or `done` is true, the algorithm stops.

Before we compute the direction, the type of direction needs to be identified by **identifyDir**. **VSBBO** calls **direction** to generate 5 kinds of direction vectors: coordinate directions, limited memory quasi-Newton directions, random subspace directions, random directions, and cumulative directions, as pointed out earlier in more detail in Subsection 5. Coordinate directions are the coordinate axes e_i, $i = 1, \cdots, n$, in a cyclic fashion. The coordinate direction values enhance the global search properties, decreasing on average with the number of function evaluations used. Moreover, they are used to estimate the gradient by the finite difference approach. Limited memory quasi-Newton directions are computed by **lbfgsDir** (standard limited memory BFGS direction [43]). Due to rounding errors, the computed direction may not satisfy the angle condition (3); hence it needs to be modified by **enforceAngle** discussed in [35]. **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. Their tuning parameters are `cum` (the cumulative step type), $\texttt{ms}_{\max}$ (the maximum number of best points kept), $\texttt{mq}_{\max}$ (the memory for L-BFGS approximation), $0 < \gamma_w < 1$ and $0 < \gamma_{\text{angle}} < 1$ (tiny parameters for the angle condition), `scSub` (random subspace direction scale?), and `scCum` (cumulative direction scale?).

Let C be the number of coordinate directions, R be the number of random directions, and S be the number of subspace directions in each repeated call to the multi-line search; once the cumulative direction and L-BFGS direction are computed. Here T is the number of 5 kinds of directions satisfying $1 \le T \le C + S + R + 2$. C, R, and S are the tuning parameters.

Both **setScales** and **FDS** work by making repeated calls to **MLS**, a multi-line search procedure, which is an improved version of **SLS**. **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 a multiple of Δ. Schematically, it works as follows:

• At first, at most C iterations with coordinate directions are used. They are used to estimate the gradient.

• Then, the L-BFGS direction is used only once since the gradient has been estimated by the finite difference technique using the coordinate directions.

• Next, except in the final iteration, at most S iterations with subspace directions are used. These directions are very useful, especially after performing the coordinate directions and L-BFGS, due to our numerical experiments.

• 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.

MLS calls an improved version of **extrapolationStep**, which is the same as **extrapolationStep**, except that it updates the cumulative step q and the cumulative gain r by **updateCum** whenever the second cumulative direction is used.

An important question is the ordering of the search directions. In Section 7, it will be shown that after coordinate directions using the random subspace direction is very preferable. Changing the ordering other direction is not very effective on the efficiency of our algorithm. However, using all directions is useful, especially using coordinate directions increase the efficiency of **VSBBO** provided that some random and random subspace directions are tried after it.

The statement (i) of Theorem 1 remains valid when R is replaced by T, and the statement (ii) of it remains valid with probability $\geq 1 - 2^{C+S+2-T} = 1 - 2^{-R}$.

Let T_0 be the maximal number of multi-line searches in **setScales** as the tuning parameter. Then, **setScales** uses $(2T+1)T_0$ function evaluations which does not affect on the order of the complexity bounds. Theorems 3, 4, and 5 are valid with probability $P \geq 1 - 2^{2+C+S-T} = 1 - 2^{-R}$, where 5 kinds of directions are used. Given the tuning parameter $\texttt{alg} \in \{0, 1, 3, 4, 5\}$ (algorithm type), we now discuss the factor of bounds depending on the number of search directions used in **MLS**. We would have the following cases:

(i) In the first case ($\texttt{alg} = 0$), $T = R < n$ random directions are used. Then complexity results considered as Table 3 are valid. This variant of **VSBBO** is denoted by **VSBBO-basic1**.

(ii) In the second case ($\texttt{alg} = 1$), $T = R \geq n$ random directions are used. Then complexity results considered as Table 3 are valid but with the factor of n^2. This variant of **VSBBO** is denoted by **VSBBO-basic2**.

(iii) In the third case ($\texttt{alg} = 2$), random, random subspace, and cumulative directions are used whose total number is $T = S + R + 1 < n$. The complexity results considered as Table 3 are valid. This variant of **VSBBO** is denoted by **VSBBO-C-Q**, ignoring the coordinate and limited memory quasi Newton directions.

(iv) In the fourth case ($\texttt{alg} = 3$), coordinate, random, random subspace, and cumulative directions are used whose total number is $T = C + S + R + 1 > n$. The complexity results considered as Table 3 are valid but with the factor of n^2. This variant of **VSBBO** is denoted by **VSBBO-Q**, ignoring the limited memory quasi Newton directions.

(v) In the fifth case ($\texttt{alg} = 4$), only subspace directions are ignored. The total number of directions used is $T = C + R + 2 > n$; hence the complexity results are valid but with the factor n^2. This variant of **VSBBO** is denoted by **VSBBO-S**.

(vi) In the sixth case ($\texttt{alg} = 5$), coordinate, L-BFGS, random, random subspace, and cumulative directions are used successively whose total number is $T = C + S + R + 2 > n$. The complexity results considered as Table 3 are valid but with the factor of n^2. This variant of **VSBBO** is the default version.

This defines six versions of **VSBBO**, the full algorithm and 5 simplified variants. In Section 7, we compare them and show that each simplification degrades the algorithm. This means that all heuristic components of **VSBBO** are necessary for best performance.

7 Numerical results

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

7.1 Default parameters for **VSBBO**

For our tests we used the following parameter choices:

Common tuning parameters: $E = 10$; $\delta_{\min} = 10^{-4}\sqrt{n}$; $\delta_{\max} = 0.1\sqrt{n}$; $\Delta_{\min} = 0$; $\Delta_{\max} = 10^{-3}$; $\gamma_\delta = 10^6$; $\gamma_e = 2$; $Q = 2$; $\lambda_{\max} = 1$;
VSBBO-basic1: $R = \texttt{fix}(n/2) + 1$
VSBBO-basic2: $R = n$
VSBBO-C-Q: $\texttt{ms}_{\max} = 5$; $R = \texttt{fix}(n/2) + 1$; $S = \texttt{fix}(n/5)$; $T_0 = 2 * \texttt{ms}_{\max}$; $\texttt{scCum} = 0$; $\texttt{scSub} = 0$; $\gamma_{\max} = 10^{-3}$; $\texttt{cum} = 1$;
VSBBO-S: $C = n$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $\texttt{scCor} = 0$; $\texttt{cum} = 1$; $\gamma_w = \varepsilon_m$; $\gamma_{\text{angle}} = 10^{-20}$; $\texttt{mq}_{\max} = 5$
VSBBO-Q: $\texttt{ms}_{\max} = 5$; $C = n$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $S = \min(\texttt{fix}(n/10), 5)$; $\texttt{scCor} = 0$; $\texttt{scSub} = 0$; $\texttt{cum} = 1$; $\texttt{ms}_{\max} = 5$; $T_0 = 2 * \texttt{ms}_{\max}$; $\gamma_{\max} = 10^{-3}$;
VSBBO: $\texttt{ms}_{\max} = 5$; $\texttt{mq}_{\max} = 5$; $T_0 = 2 * \texttt{ms}_{\max}$; $C = n$; $S = \min(\texttt{fix}(n/10) + 1, 5)$; $R = \min(\texttt{fix}(n/10) + 1, 20)$; $\texttt{scCum} = 0$; $\texttt{scCor} = 0$; $\texttt{scSub} = 0$; $\texttt{cum} = 1$; $\gamma_{\max} = 10^{-3}$; $\gamma_w = \varepsilon_m$; $\gamma_{\text{angle}} = 10^{-20}$;

Table 4: The values of the tuning parameters

Note that $R = \mathcal{O}(n)$ so that the resulting worst case complexity is $\mathcal{O}(n^2\varepsilon^{-2})$, but fixing R for (moderately large) n was less efficient.

$\Delta_{\min} = 0$ implies that the algorithm stops due to `nfmax` or `secmax`.

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., [1,46,47]. In Table 4, there are 7 integral, 2 binary, 2 ternary, and 12 continuous tuning parameters, giving a total of 23 parameters for tuning our algorithm. A small amount of tuning was done by hand. Automatic tuning of **VSBBO** will be considered elsewhere.

7.2 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; if nothing is said, the default option were used.

• **STP-fs, STP-vs**, and **PSTP**, obtained from the authors of Bergou et al. [6], are three versions of a stochastic direct search method with complexity guarantees.

• **BFO**, obtained from

`https://github.com/m01marpor/BFO`,

is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli & Toint [47].

• **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 tuning parameters oCMAES.MaxFunEvals = nfmax, oCMAES.DispFinal = 0, oCMAES.DispModulo = 0, oCMAES.LogModulo = 0, oCMAES.SaveVariables = 0, oCMAES.MaxIter = nfmax, and 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. [10]. We used **GLOBAL** with the tuning parameters oGLOBAL.MAXFNALL = nfmax, oGLOBAL.MAXFN = nfmax/5, oGLOBAL.DISPLAY ='off', oGLOBAL.N100 = 300, oGLOBAL.METHOD ='unirandi', and oGLOBAL.NG0 = 2.

• **DE**, obtained from

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

is the stochastic differential evolution algorithm by Storn & Price [50].

• **MCS**, obtained from

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

is the deterministic global optimization by multilevel coordinate search by Huyer & Neumaier [30]. We used **MCS** with the tuning parameters iinit = 1, nfMCS = nfmax, smax = $5n + 10$, stop = $3n$, local = 50, gamma = eps; hess = ones(n, n), and 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. [26].

• **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 [12,13].

• **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);
```

It is the standard quasi Newton method while finding step sizes by Wolfe condition. • **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. [37]. We use **fminseach** with the options set by

```
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. [16,17]. The default parameters are used.

• **PSWARM**, obtained from

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

is Particle swarm pattern search algorithm for global optimization presented by Vaz & Vicente [52].

• **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 [32]. The default parameters are used.

• **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 [29].

• **GLODS**, obtained from

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

is Global and Local Optimization using Direct Search, presented by Custódio & Madeira [11].

• **ACRS**, obtained from

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

is a global optimization algorithm presented by Brachetti et al. [8].

• **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 [39].

• **DSPFD**, available at

`pages.discovery.wisc.edu/%7Ecroyer/codes/dspfd_sources.zip`,

is a direct-search MATLAB code for derivative-free optimization by Gratton et al. [25]. The default parameters are used.

• **DESTRESS**, downloaded from

`pages.discovery.wisc.edu/%7Ecroyer/codes/destress_sources.zip`,

is a trust-region method in MATLAB for smooth, unconstrained optimization problems, with second-order convergence guarantees, by Gratton et al. [23]. The default parameters are used.

• **SDS**, **AHDS** and **DSDS**, downloaded from

`pages.discovery.wisc.edu/%7Ecroyer/codes/sounds_sources.zip`,

are symmetrized direct search, approximate Hessian direct search and decoupled step direct search, respectively. **SDS** and **AHDS** suggested by Gratton et al. [24] and **DSDS** proposed by Royer [49]. The default parameters are used for all.

VSBBO and the other stochastic algorithms use random numbers, hence give slightly different results when run repeatedly. Due to run time constraints, 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 value). 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).

7.3 Test problems used

We compare 28 competitive solvers from the literature on all 549 unconstrained problems from the `CUTEst` [22] collection of test problems for optimization with up to 5000 variables, in case of variable dimension problems for all allowed dimensions in this range. For problems in dimension $n > 100$, only the most robust and fast solvers were compared. To avoid guessing the solution of toy problems with a simple solution (such as all zero or all one), we shifted the arguments by $\xi_i = (-1)^{i-1}2/(2+i)$ for all $i = 1, \ldots, n$.

`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

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

seconds of run time and at most

$$\texttt{nfmax} := \begin{cases} 2n^2 + 1000n + 5000 & \text{if } 1 \le n \le 100, \\ 100n & \text{if } 101 \le n \le 5000 \end{cases}$$

function evaluations for a problem with n variables. We ran all solvers by monitoring in the function evaluation a routine the number of function values and the time used until the bound of this number was met or an error occurred. We saved time and number of function values at each improved function value and evaluated afterwards when the target accuracy was reached. In order to get the above choices for `nfmax` and `secmax`, we made preliminarily run to ensure that the best solver can solve the most test problems. Both `nfmax` and `secmax` are input parameters for all solvers.

A problem with dimension n is considered solved if the target accuracy satisfies

$$q_f := (f - f_{\text{best}})/(f_{\text{init}} - f_{\text{best}}) = \begin{cases} 10^{-4} & \text{if } 1 \le n \le 100, \\ 10^{-3} & \text{if } 101 \le n \le 5000, \end{cases}$$

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.

The best point known to us was obtained numerous attempts for finding the best local minimizer or global minimizer for all 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. [7], **ASACG** presented by Hagar & Zhang [27] and **LMBOPT** implemented by Kimiaei et al. [35]. The condition $\|g_k\|_\infty \le 10^{-5}$ was satisfied for all test problems except those listed in Appendix B.

For a more refined statistics, we use our test environment (Kimiaei & Neumaier [34]) for comparing optimization routines on the `CUTEst` test problem collection by Gould et al. [22]. For a given collection S of solvers, the strength of a solver $so \in S$ – relative to an 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

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

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 problems tried by all solvers and solved by at least one of them, from a total of all 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 $T_{\text{mean}} := \dfrac{\sum \text{solved}}{\#\text{ solved}}$. Failure reasons were reported in the anomaly columns:

- n indicates that `nf` $\geq$ `nfmax was reached`.
- t indicates that `sec` $\geq$ `secmax 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 was stopped. Table entries use the maximal allowed time (`secmax` sec) for each unsuccessful run.

Besides performance plots [18] for the two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds), performance plots are shown in Figures 1–8 and the other results are give all results in Tables 5–8.

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

The low dimensional test problems ($n \leq 20$) unsolved by all 28 solvers are HATFLDFL, FLETCBV3, and FLETCHBV. In summary, the results for the remaining problems are summarized in Table 5.

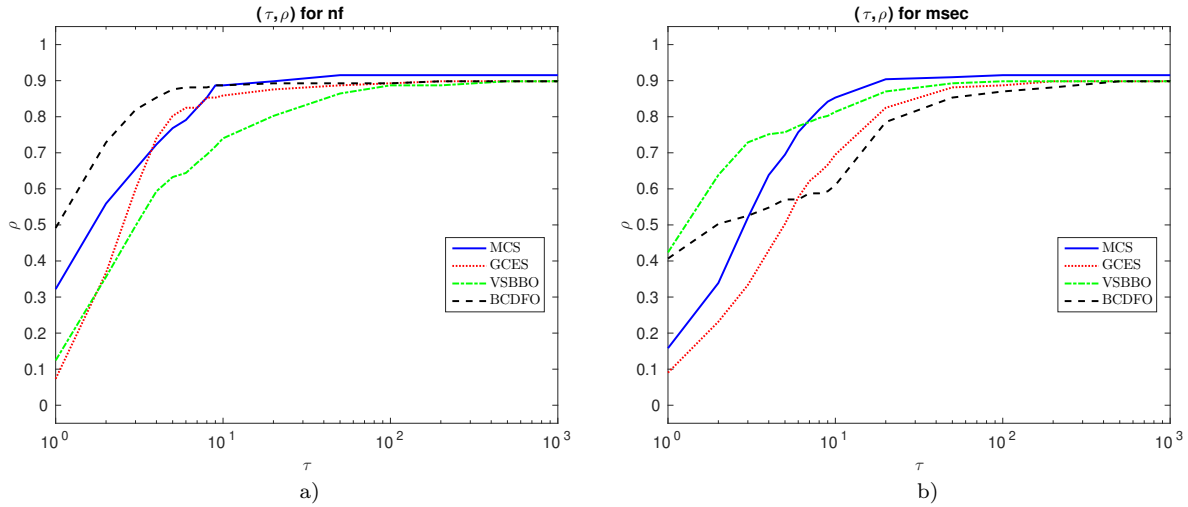

Fig. 1: Small dimensions 2–20: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver.

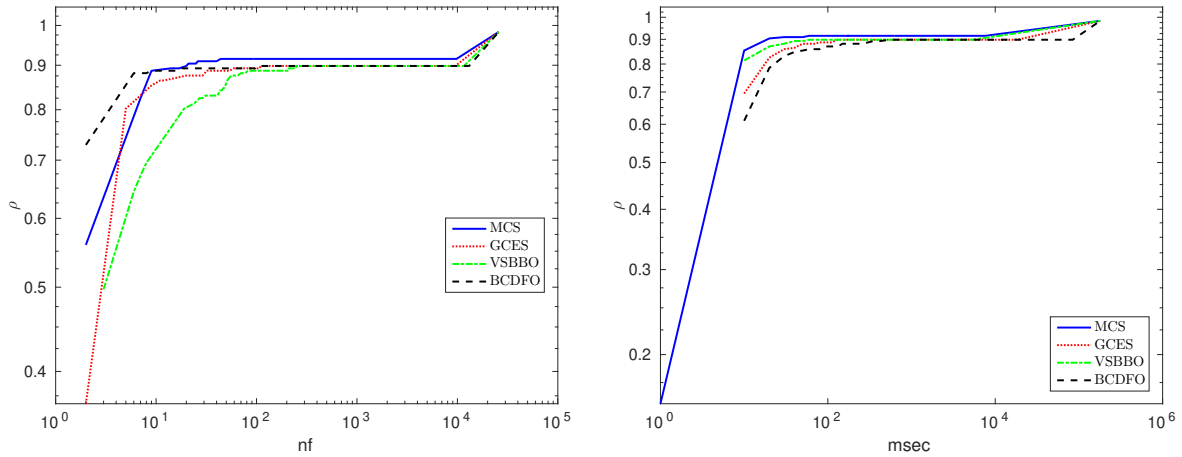

Fig. 2: Small dimensions 2–20: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

From Table 5 and Figures 1–2, we see that on problems with few variables,

• **VSBBO** and **VSBBO-Q** have above average performance and can compete in robustness with the best solvers **MCS**, **GCES**, **NELDER**, and **BCDFO**.

• **VSBBO** and **VSBBO-Q** are the best solvers in terms of the number of solved problems and the `nf` efficiency, respectively in comparison with **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**. A question that may be asked here is why **VSBBO** and **VSBBO-Q** are the best. The answer is that using the coordinate directions is very effective provided that, after these directions, random subspace directions are tried. Note that **VSBBO-C-Q** uses the random subspace directions after performing random directions but it is weaker than **VSBBO** and **VSBBO-Q**.

• **VSBBO-basic1**, the basic version of **VSBBO**, for which complexity results considered as Table 3 are valid is more robust than solvers proposed by Gratton et al. [25] (**DSPFD**) and Bergou et al. [6] (**STP-vs**, **PSTP**, and **STP-fs**), discussed earlier in Section 1.2. As pointed out earlier, there is no plan to test whether the function value is decreased or not; this is a reason why three solvers proposed by Bergou et al. are numerically poor.

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

We tested all 28 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.

Table 5: 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 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	22	21	428	3	0	12	41	13
GCES	**gecs**	159	5	3	1684	1	0	17	29	9
VSBBO	**vsbb**	159	3	2	325	18	0	0	25	18
BCDFO	**bcd**	159	41	37	3796	0	1	17	53	21
NMSMAX	**nmsm**	157	10	9	193	12	0	8	35	41
NELDER	**neld**	157	14	11	190	0	0	20	36	42
PSM	**psm**	156	7	6	1222	10	2	9	37	14
SDBOX	**sdb**	144	5	4	205	33	0	0	25	43
FMINUNC	**func**	142	54	51	84	0	0	35	52	53
CMAES	**cma**	142	3	2	457	35	0	0	10	10
BFO	**bfo**	140	1	1	269	0	0	37	19	19
SDS	**sds**	132	3	1	285	0	0	45	16	22
AHDS	**ahds**	130	2	0	283	12	0	35	11	14
ADSMAX	**adsm**	126	1	1	662	37	0	14	14	10
DSDS	**dsds**	125	2	0	345	12	0	40	8	11
MDSMAX	**mdsm**	123	3	2	318	53	0	1	12	18
PSWARM	**psw**	122	8	5	491	33	0	22	9	6
HOOKE	**hook**	120	0	0	196	13	0	44	18	24
DSPFD	**dspf**	113	4	4	995	0	0	64	15	11
FMINSEARCH	**fmin**	97	0	0	575	20	0	60	7	6
GLOBAL	**glo**	93	3	2	166	21	0	63	7	12
DE	**de**	84	0	0	769	68	0	25	0	2
DESTRESS	**dest**	80	0	0	284	33	0	64	6	8
STP-vs	**svs**	57	2	0	258	120	0	0	4	8
PSTP	**pst**	55	2	1	913	122	0	0	2	3
STP-fs	**sfs**	52	0	0	308	125	0	0	3	5
ACRS	**acr**	50	1	0	330	83	0	44	3	4
MDS	**mds**	13	3	1	133	96	0	68	2	3
166 of 177 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	159	36	31	325	18	0	0	66	60
VSBBO-Q	**vsbbq**	158	56	50	336	19	0	0	68	66
VSBBO-S	**vsbbs**	148	34	28	414	29	0	0	54	55
VSBBO-basic1	**vsbb1**	129	26	24	632	46	0	2	39	39
VSBBO-C-Q	**vsbcq**	128	18	14	831	46	0	3	32	28
VSBBO-basic2	**vsbb2**	127	13	10	636	47	0	3	36	39
137 of 177 problems solved										mean efficiency in %
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	129	59	58	632	46	0	2	55	56
DSPFD	**dspf**	113	64	63	995	0	0	64	49	43
STP-vs	**svs**	57	4	4	258	120	0	0	10	20
PSTP	**pst**	55	6	6	913	122	0	0	6	7
STP-fs	**sfs**	52	5	5	308	125	0	0	8	15

From Table 6 and Figures 3–4, we see that on problems with a medium number of variables,

• **VSBBO-Q** and **VSBBO** are outstanding in robustness, but **FMINUNC** is the best in terms of `nf` and `sec` efficiency.

• **VSBBO-Q** and **VSBBO** are the best version of **VSBBO** in terms of the number of solved problems, #100, !100, the `nf` and `msec` efficiencies, respectively.

• **VSBBO-basic1** is more robust than than solvers proposed by Gratton et al. [25] (**DSPFD**) and Bergou et al. [6] (**STP-vs**, **PSTP**, and **STP-fs**).

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

We tested the 6 most robust solvers from Table 6 on large dimensional test problems ($101 \leq n \leq 1000$). The problems unsolved by the 6 solvers are EIGENALS, EIGENBLS, EIGENCLS, SPMSRTLS:499, GEN-

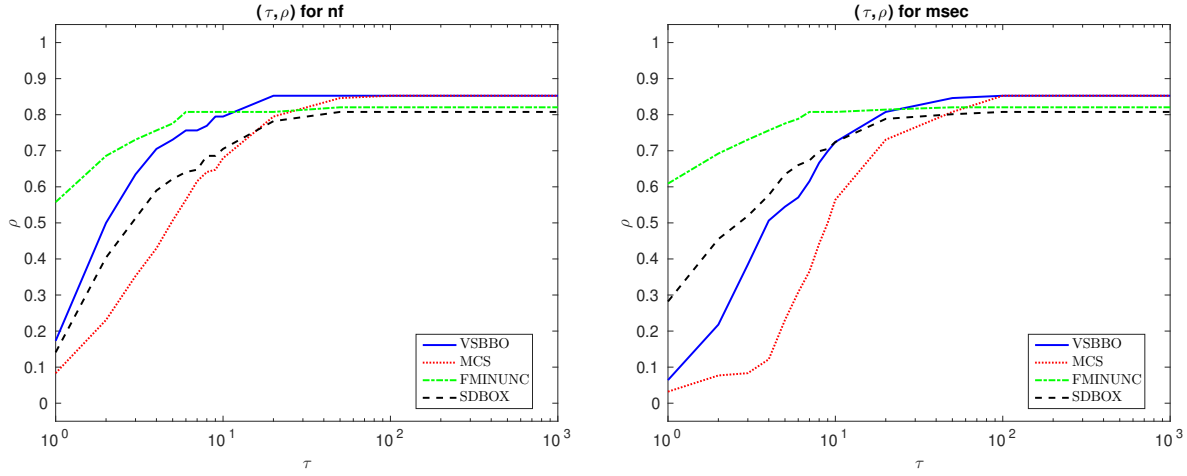

Fig. 3: Medium dimensions 21–100: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

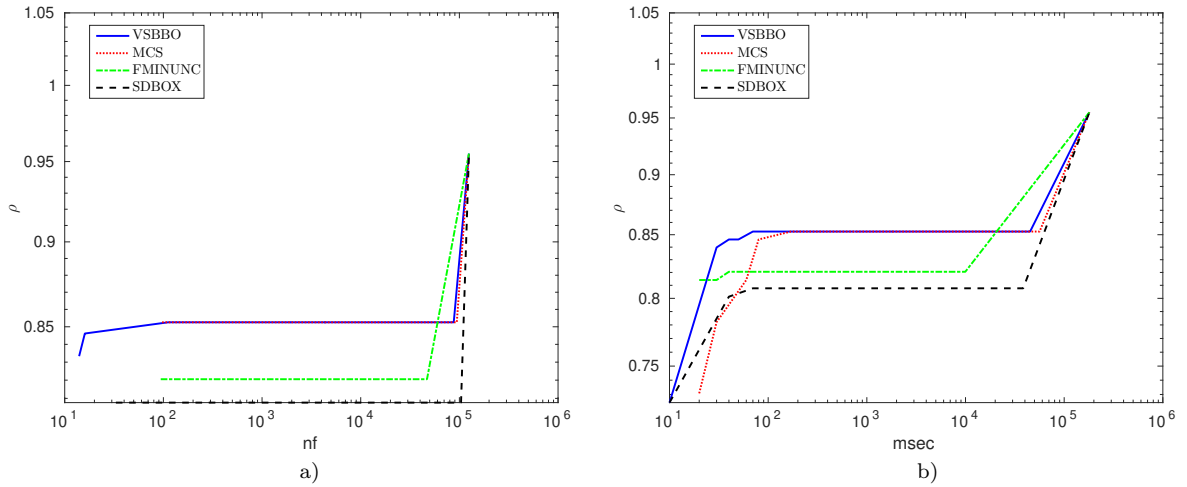

Fig. 4: Medium dimensions 21–100: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

ROSE:500, PENALTY2:500, SINQUAD:500, MSQRTALS:529, MSQRTBLS:529, NONMSQRT:529, CURLY10, CURLY20, CURLY30, DIXON3DQ:1000, FLETCBV2:1000, FLETCBV3:1000, FLETCHCR:1000, NONCVXU2, SENSORS:1000, SINQUAD:1000, and SPMSRTLS:1000.

Table 6: The summary results for medium dimensions 21–100

stopping test: $q_f \leq$ `0.0001`, `sec` $\leq$ `180`, `nf` $\leq 2n^2 + 1000n + 5000$										
151 of 156 problems 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**	133	11	11	2473	23	0	0	39	25
MCS	**mcs**	133	5	5	3852	3	0	20	22	14
FMINUNC	**func**	128	76	74	911	0	0	28	62	68
SDBOX	**sdb**	126	19	15	1615	30	0	0	37	45
ADSMAX	**adsm**	115	9	6	5021	36	0	5	25	14
CMAES	**cma**	101	1	0	14852	52	3	0	7	2
PSWARM	**psw**	100	3	2	11142	51	0	5	5	3
SDS	**sds**	97	2	0	3304	0	3	56	6	11
NMSMAX	**nmsm**	94	3	3	5214	60	2	0	13	10
MDSMAX	**mdsm**	90	1	0	4172	66	0	0	4	9
NELDER	**neld**	90	1	1	25525	5	29	32	13	4
DSPFD	**dspf**	89	8	8	21346	0	31	36	18	3
BFO	**bfo**	83	0	0	4781	0	4	69	5	5
HOOKE	**hook**	76	1	1	4616	11	0	69	11	7
DE	**de**	73	0	0	3670	48	0	35	1	4
GLOBAL	**glo**	36	1	1	1925	31	0	89	4	5
DESTRESS	**dest**	34	0	0	1757	75	0	47	1	2
PSM	**psm**	31	4	4	28431	0	125	0	8	1
DSDS	**dsds**	31	2	0	2190	89	4	32	3	4
AHDS	**ahds**	30	2	0	2503	113	4	9	1	2
GCES	**gecs**	27	0	0	42836	0	124	5	4	0
BCDFO	**bcd**	20	11	10	29203	0	135	1	7	1
STP-vs	**svs**	18	0	0	6906	137	1	0	0	0
PSTP	**pst**	17	2	1	4408	138	1	0	2	2
STP-fs	**sfs**	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
138 of 156 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-Q	**vsbbq**	135	45	41	2188	21	0	0	73	70
VSBBO	**vsbb**	133	40	35	2473	23	0	0	70	66
VSBBO-S	**vsbbs**	124	37	36	3478	32	0	0	59	62
VSBBO-basic1	**vsbb1**	94	6	5	5096	61	1	0	23	21
VSBBO-basic2	**vsbb2**	93	7	7	5156	63	0	0	23	23
VSBBO-C-Q	**vsbcq**	42	9	8	9087	113	1	0	10	10
100 of 156 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	94	20	18	5096	61	1	0	41	58
DSPFD	**dspf**	89	76	74	21346	0	31	36	53	18
STP-vs	**svs**	18	0	0	6906	137	1	0	1	4
PSTP	**pst**	17	5	5	4408	138	1	0	4	5
STP-fs	**sfs**	15	1	1	7302	140	1	0	1	3

A comparison of 6 solvers in Table 7 and Figures 4–5 shows that **VSBBO** is the most robust solver. This solver is better than **SDBOX** in terms of the `nf` efficiency and than **FMINUNC** in terms of the number of solved problems. **VSBBO-basic1**, the basic version of **VSBBO**, is more robust than **MCS** and **DSPFD**. Due to the use of L-BFGS direction, **VSBBO** is better than **VSBBO-Q** but the converse was true for small scale and medium problems. Moreover, **VSBBO-S** and **VSBBO-Q** have the same quality and are more efficient than **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**.

7.7 Results for very large dimensions ($1001 \leq n \leq 5000$)

We tested the 3 most robust solvers from Table 7 on very large dimensional test problems ($101 \leq n \leq 1000$). The problems unsolved by the three best solvers are MSQRTALS:1024, MSQRTBLS:1024, NONMSQRT:1024, EIGENALS:2550, EIGENBLS:2550, EIGENCLS:2652, MSQRTALS:4900, MSQRTBLS:4900, SPMSRTLS:4999, FLETCBV2:5000, NONCVXU2:5000, NONCVXUN:5000, and SINQUAD:5000.

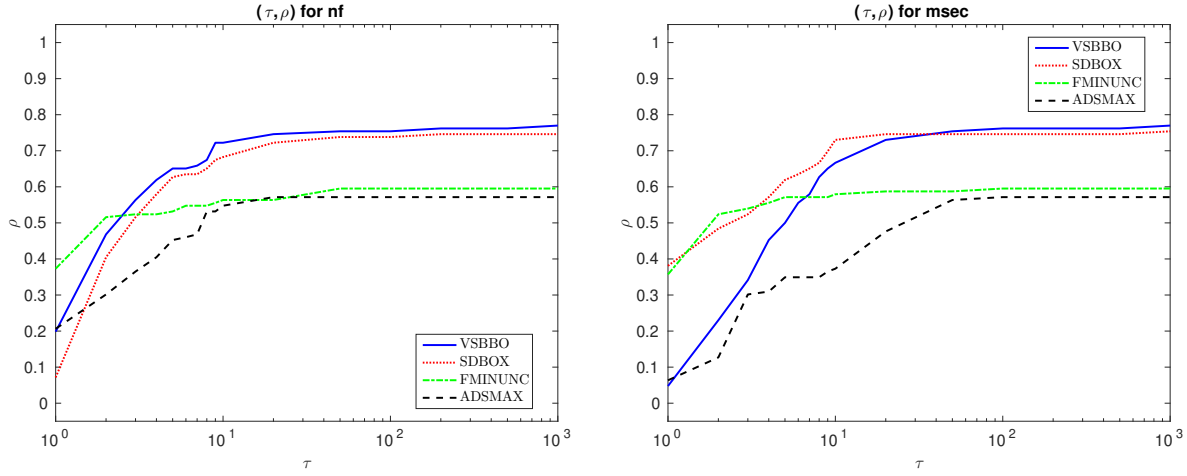

Fig. 5: Large dimensions 101–1000: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

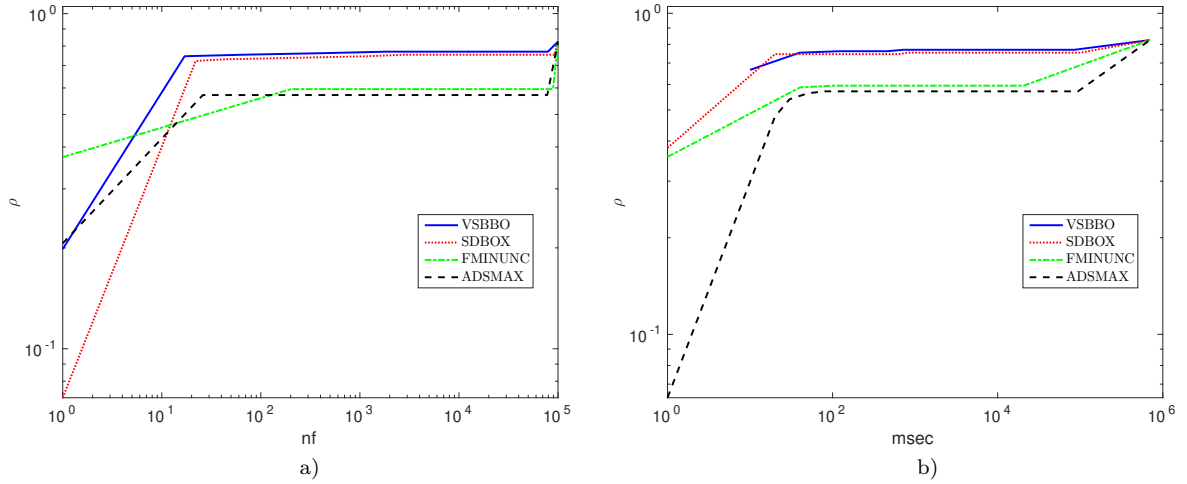

Fig. 6: Large dimensions 101–1000: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

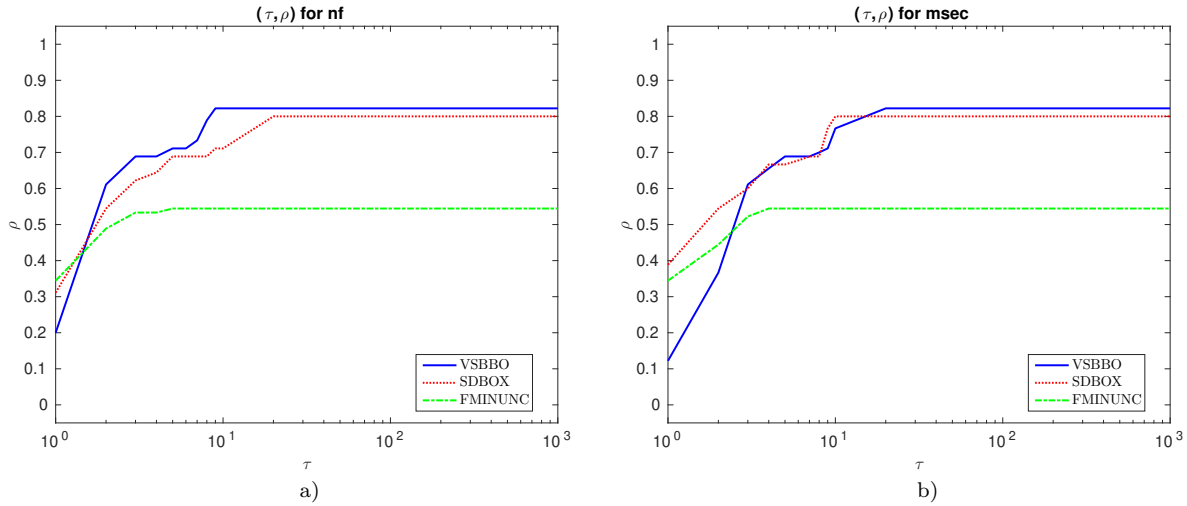

Fig. 7: Very large dimensions 1001–5000: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ notes the fraction of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

Table 7: The summary results for for large dimensions 101–1000

stopping test:		$q_f \le$ 0.001,		sec $\le$ 700,		nf $\le$ 100*n				
104 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**	97	21	20	9471	28	1	0	44	30
SDBOX	**sdb**	95	9	8	5240	30	1	0	39	51
FMINUNC	**func**	75	45	42	2688	28	0	23	46	50
ADSMAX	**adsm**	72	25	24	11076	49	1	4	32	19
DSPFD	**dspf**	18	4	3	140555	0	71	37	6	1
MCS	**mcs**	17	7	4	24057	7	1	101	6	3
101 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	97	39	23	9471	28	1	0	67	60
VSBBO-S	**vsbbs**	94	40	28	8166	31	1	0	63	65
VSBBO-Q	**vsbbq**	94	38	24	9598	31	1	0	63	57
VSBBO-basic2	**vsbb2**	43	3	2	13033	82	1	0	13	12
VSBBO-basic1	**vsbb1**	42	6	5	13712	83	1	0	15	14
VSBBO-C-Q	**vsbcq**	22	3	2	12359	103	1	0	7	7
104 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	97	46	45	9471	28	1	0	55	42
FMINUNC	**func**	75	59	58	2688	28	0	23	54	56
43 of 126 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO-basic1	**vsbb1**	42	29	28	13712	83	1	0	28	33
DSPFD	**dspf**	18	15	14	140555	0	71	37	13	2

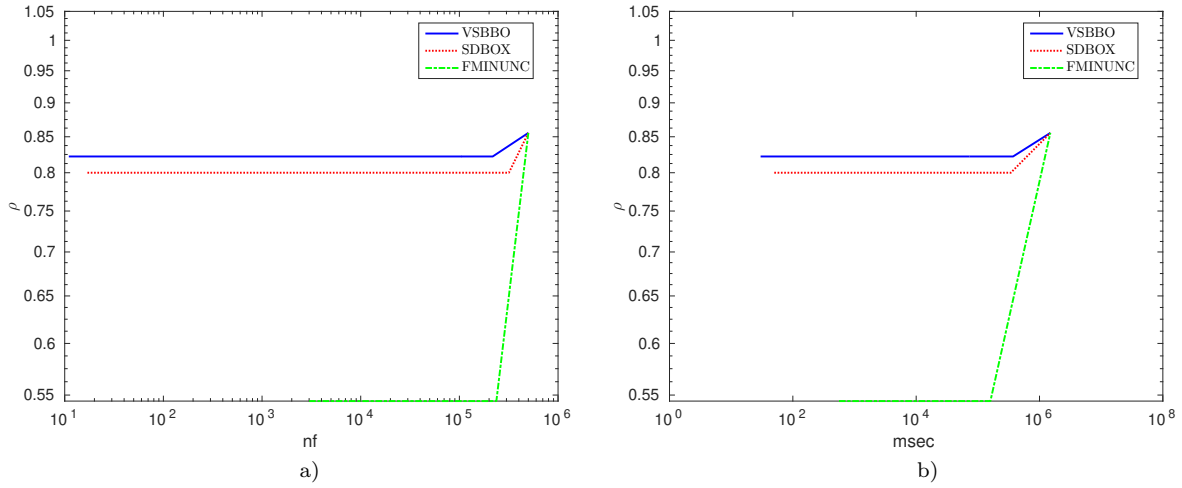

Fig. 8: Very large dimensions 1001–5000: Performance plots for (a) nf/(best nf) and (b) msec/(best msec). ρ notes the fraction of problems solved within the number of function evaluations and time in milliseconds used by the best solver. Problems solved by no solver are ignored.

In Table 8 and Figures 7–8, we give a comparison of best solvers **SDBOX**, **VSBBO**, and **FMINUNC** for very large dimensions 1001 up to 5000. The high quality of **VSBBO** is clearly visible. **VSBBO** is the best in terms of the nf efficiency, #100 and !100. **VSBBO** and **VSBBO-Q** have the same behaviour, are a little better than **VSBBO-S**, and are more efficient than **VSBBO-basic1**, **VSBBO-basic2**, and **VSBBO-C-Q**.

8 Conclusion

We constructed an efficient stochastic algorithm for unconstrained black box optimization problems. For the basic version of **VSBBO** with only random directions, the complexity bound for the nonconvex case, with probability arbitrarily close to 1, matches that found by Gratton et al. [25] for another algorithm. We also

Table 8: The summary results for very large dimensions 1001–5000

stopping test:		$q_f \leq$ 0.001,		sec $\leq$ 1500,		nf $\leq$ 100*n				
77 of 90 problems 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**	74	18	18	74585	14	2	0	60	42
SDBOX	**sdb**	72	29	28	61545	13	5	0	54	57
FMINUNC	**func**	49	31	30	36280	30	0	11	47	44
75 of 90 problems solved									mean efficiency in %	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
VSBBO	**vsbb**	74	37	24	74585	14	2	0	75	74
VSBBO-Q	**vsbbq**	74	36	22	86023	14	2	0	75	74
VSBBO-S	**vsbbs**	73	23	11	96244	15	2	0	72	61
VSBBO-basic1	**vsbb1**	19	1	0	197927	67	4	0	6	5
VSBBO-basic2	**vsbb2**	17	1	0	163034	70	3	0	6	5
VSBBO-C-Q	**vsbbq**	11	2	1	299000	77	2	0	4	3

proved a complexity bounds for **VSBBO** for the convex and strongly convex cases, with probability arbitrarily close to 1, essentially matching the bounds found by Bergou et al. [6], only valid in expectation.

Numerical results showed that the basic version of **VSBBO** (**VSBBO-basic1**) was more efficient than the stochastic direct search solvers proposed by Gratton et al. [25] (**DSPFD**) and Bergou et al. [6] (**STP-vs**, **STP-vf**, and **PSTP**) whose complexity results were discussed in Section 1.2.

An improved version of our algorithm has additional heuristic techniques that do not affect the order of the complexity results and which turn **VSBBO** into an efficient global solver although our theory guarantees only local minimizers. This version even found in most cases either a global minimizer or, where this could not be checked, at least a point of similar quality with the best competitive global solvers.

Other versions of **VSBBO** were **VSBBO-C-Q**, **VSBBO-S**, and **VSBBO-Q**. **VSBBO-C-Q** used random directions, random subspace directions, cumulative direction and additional heuristic techniques but was not better than **VSBBO-basic1**. Although **VSBBO-S** did not use additional heuristic techniques and random subspace directions, it was more efficient than **VSBBO-basic1** and **VSBBO-C-Q** due to using coordinate directions and then random directions. **VSBBO-Q** used additional heuristic techniques and all directions except the L-BFGS direction. It was slightly weaker than **VSBBO** and **VSBBO-S** and was more efficient than **VSBBO-basic1** and **VSBBO-C-Q**. As a consequence, **VSBBO** was the best in comparison with the others versions.

In terms of the number of solved problems **VSBBO** was for $1 \leq n \leq 20$ more robust than the based-model trust region solvers (**DESTRESS** and **BCDFO**), the direct search solvers (**BFO**, **GCES**, **SDS**, **AHDS**, **DSDS**, and **DSPFD**), the basic deterministic line search solver with only coordinate directions (**SDBOX**), the standard quasi Newton solver (**FMINUNC**), the Nelder–Mead solvers (**FMINSEARCH**, **NELDER**, and **NMSMAX**), the multidirectional search solvers (**MDSMAX** and **MDS**), the Hooke–Jeeves solver (**HOOKE**), the alternating directions solver (**ADSMAX**), the global solvers (**CMAES**, **GLOBAL**, and **ACRS**), and the pattern search solvers (**PSM**, **PSWARM**, and **CLODS**). The quality of **VSBBO** was improved by increasing the dimension.

For $21 \leq n \leq 100$, **VSBBO** was the best solvers in the comparison with the best competitive local and global solvers in terms of the number of solved problems and was the second best solvers in terms of the `nf` efficiency.

For $101 \leq n \leq 1000$, **VSBBO** was the best solver in terms of the number of solved problems and the second best solver in terms of the `nf` efficiency. Finally, for $1001 \leq n \leq 5000$, **VSBBO** was more efficient and robust than **SDBOX** and **FMINUNC** which were the two good solvers for large scale problems. As a result, when the problem size grew, the performance of **VSBBO** became competitive.

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.

A Appendix: Estimation of c

The following theorem was recently proved by Pinelis [44].

Theorem 6 *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*

$$(p^T p)(q^T q) \le c_0 n (p^T q)^2 \tag{59}$$

with probability $\ge 1/2$.

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

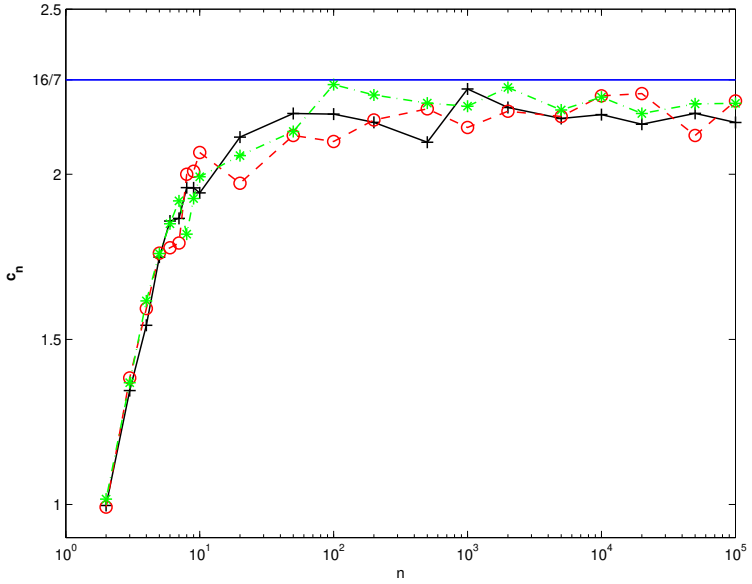

Fig. 9: The plot of c_n versus the dimension n suggests that $c_0 \approx 16/7$.

```
% 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 $16/7 = 2.2857....$

Algorithm 5 *(Estimating the Pinelis constant (***PinConst***))*
Purpose*: Estimate c_0 satisfying (59) with probability $\ge 1/2$*
Input*: N (the total number of gradient evaluations), D (vector of dimensions used)*
Output*: c_0*
(S.0) Set $M = |D|$ and $i = 1$.
(S.1) Replace i by $i+1$ and set $k = 1$.
(S.2) Replace k by $k+1$.
(S.3) Generate random g_k and p_k with length D_i and compute

$$\texttt{gain}(k) = \frac{\|g_k\|_2 \|p_k\|_2}{|g_k^T p_k|}.$$

(S.4) If k exceeds N, go to (S.2); otherwise, compute $\texttt{medgain}(i) = \texttt{median}(\texttt{gain})$ *and* $c(i) = \texttt{medgain}(i))^2/D_i$.
(S.5) If i exceeds M, **PinConst** *ends up; otherwise go to (S.1).*

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

Here we list the `CUTEst` test problems for which our best point did not satisfy the condition

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

problem	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	$1.81e+01$	$3.42e-05$	$6.51e-05$
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	$1.16e+04$	$3.94e-05$	$7.19e-04$

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. A.S. Bandeira, K. Scheinberg, and L.N. Vicente, Convergence of trust-region methods based on probabilistic models, SIAM J. Optim., 24(3) (2014), 1238–1264.
4. 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.
5. A.S. Berahas, R.H. Byrd, J. Nocedal, Derivative-free optimization of noisy functions via quasi-Newton methods, SIAM J. Optim., 29(2) (2019), 965–993.

6. E.H. Bergou, E. Gorbunov and P. Richtárik, Stochastic three points method for unconstrained smooth minimization, (2019), https://arxiv.org/abs/1902.03591.
7. 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.
8. 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.
9. A.R. Conn, K. Scheinberg, and L.N. Vicente, Introduction to derivative-free optimization, SIAM, Philadelphia, PA, 2009.
10. T. Csendes, L. Pál, J.O.H. Sendin and J.R. Banga, The GLOBAL optimization method revisited, Optim. Lett. 2 (2008), 445–454.
11. A.L. Custódio and J.F.A. Madeira, GLODS: Global and Local Optimization using Direct Search, J. Glob. Optim., 62 (2015), 1–28.
12. A.L. Custódio, L.N. Vicente, Using sampling and simplex derivatives in pattern search methods, SIAM J. Optim 18 (2007), 537–555.
13. A.L. Custódio, H. Rocha, L.N. Vicente, Incorporating minimum Frobenius norm models in direct search, Comput. Optim. Appl. 46 (2010), 265–278.
14. Y. Diouane, S. Gratton, and L.N. Vicente, Globally convergent evolution strategies, Math. Program. 152 (2015), 467–490.
15. Y. Diouane, S. Gratton, and L.N. Vicente, Globally convergent evolution strategies for constrained optimization, Comput. Optim. Appl. 62 (2015), 323–346.
16. M. Dodangeh, L.N. Vicente, Worst case complexity of direct search under convexity, Math. Program. 155(1–2) (2016), 307–332.
17. 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.
18. E. Dolan and J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2002), 201–213.
19. Yu.G. Evtushenko, Numerical Methods for Finding Global Extrema (Case of Nonuniform Mesh), Zh. Vychisl. Mat. mat. Fiz, 11 (1971), 1390–1403
20. C. Elster and A. Neumaier, A grid algorithm for bound constrained optimization of noisy functions, IMA J. Numer. Anal. 15 (1995), 585–608.
21. C. Elster and A. Neumaier, A trust region method for the optimization of noisy functions, Computing 58 (1997), 31–46.
22. 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.
23. S. Gratton, C.W. Royer and L.N. Vicente, A decoupled first/second-order steps technique for nonconvex nonlinear unconstrained optimization with improved complexity bounds. Math. Program. 179 (2020), 195–222.
24. S. Gratton, C.W. Royer and L.N. Vicente, A second-order globally convergent direct-search method and its worst-case complexity, Optimization 65(6) (2016), 1105–1128.
25. S. Gratton, C.W. Royer, L. N. Vicente and Z. Zhang, Direct search based on probablistic descent, SIAM J. Optim. 25 (2015), 1515–1541.
26. 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.
27. W.W. Hager and H. Zhang, A New Active Set Algorithm for Box Constrained Optimization, SIAM J. Optim 17(2) (2006), 526–557.
28. 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.
29. N.J. Higham, Optimization by direct search in matrix computations, SIAM J. Matrix Anal. Appl. 14(2)(1993), 317–333.
30. W. Huyer and A. Neumaier, Global optimization by multilevel coordinate search, J. Glob. Optim. 14 (1999), 331–355.
31. W. Huyer and A. Neumaier, SNOBFIT–stable noisy optimization by branch and Fit, ACM. Trans. Math. Software 35, Article 9 (2008).
32. C.T. Kelley, Iterative methods for optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
33. D.E. Kvasov and Ya.D. Sergeyev, Lipschitz gradients for global optimization in a one-point-based partitioning scheme, J. Comput. Appl. Math., 236(16) (2012), 4042–4054.
34. M. Kimiaei, A. Neumaier, Testing and tuning optimization algorithm, in preparation (2019).
35. M. Kimiaei, A. Neumaier, B. Azmi, LMBOPT–a limited memory method for bound-constrained optimization, (2019), https://www.mat.univie.ac.at/~neum/software/LMBOPT/.
36. J. Konečný and P. Richtárik, Simple complexity analysis of simplified direct search, Manuscript (2014), https://arxiv.org/abs/1410.0390.
37. 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.
38. J. Larson, M. Menickelly, S.M. Wild, Derivative-free optimization methods, Acta Numer. 28 (2019), 287–404.
39. S. Lucidi, M. Sciandrone, A Derivative-Free Algorithm for Bound Constrained Optimization, Comput. Optim. Appl. 21(2) (2002), 119–142.
40. Y. Nesterov, Random gradient-free minimization of convex functions, CORE discussion paper #2011/1, unpublished (2011).
41. Y. Nesterov and V. Spokoiny, Random gradient-free minimization of convex functions, Found. Comput. Math. 17 (2017), 527–566.

42. A. Neumaier, H. Fendl, H. Schilly and T. Leitner, Derivative-free unconstrained optimization based on QR factorizations, Soft Computing 15 (2011), 2287–2298.
43. J. Nocedal, S. Wright, Numerical Optimization. Springer New York, 2 edition, 1999.
44. I. Pinelis, A probabilistic angle inequality, MathOverflow (2018). https://mathoverflow.net/q/298590
45. J. Pintér, Globally convergent methods for n-dimensional multiextremal optimization, Optimization, 17(2) (1986), 187–202.
46. M. Porcelli, Ph. L. Toint, A note on using performance and data profiles for training algorithms, ACM Trans. Math. Softw. 45, 2, Article 20 (April 2019), 10 pages.
47. 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.
48. L.M. Rios and N.V. Sahinidis, Derivative-free optimization: A review of algorithms and comparison of software implementations, J. Glob. Optim. 56 (2013), 1247–1293.
49. C.W Royer, Derivative-free optimization methods based on probabilistic and deterministic properties: complexity analysis and numerical relevance, PhD Thesis, (2016), http://pages.discovery.wisc.edu/\%7Ecroyer/docs/thesisRoyer.pdf
50. 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.
51. P.J.M. Van Laarhoven and E.H.L. Aarts, Simulated annealing: theory and applications, Kluwer, Dordrecht (1987).
52. 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.
53. L.N. Vicente, Worst case complexity of direct search, EURO J. Comput. Optim. 1(1–2) (2013), 143–153.