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

Efficient global unconstrained black box optimization

Morteza Kimiaei[1] · **Arnold Neumaier**

the date of receipt and acceptance should be inserted later

Abstract For the unconstrained optimization of black box functions, this paper introduces a new randomized algorithm called **VRBBO**. In practice, **VRBBO** 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 **VRBBO** 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, a basic version of our algorithm finds with $\mathcal{O}(n\varepsilon^{-2})$ function evaluations a point whose unknown exact gradient 2-norm is below a given threshold $\varepsilon > 0$, where n is the dimension. In the smooth convex case, this number improves to $\mathcal{O}(n\log\varepsilon^{-1})$ and in the smooth (strongly) convex case to $\mathcal{O}(n\log n\varepsilon^{-1})$. This matches known recent complexity results for reaching a slightly different goal, namely the expected unknown exact gradient 2-norm is below a given threshold $\varepsilon > 0$.

Numerical results show that **VRBBO** is effective and robust in comparison with the state of the art local and global solvers on the unconstrained `CUTEst` test problems

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

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

by Gould [26] for optimization and the test problems by Jamil & Yang [37] for global optimization with 2 up to 5000 variables.

Keywords Derivative-free optimization · complexity bounds · global optimization · sufficient decrease · line search

2000 AMS Subject Classification: primary 90C56.

August 1, 2021

1 Introduction

In this paper we consider unconstrained black box optimization (BBO) or derivative-free optimization (DFO); see, e.g., [3,15,58]. The labels BBO and DFO are used in practice synonymously, though with slightly different emphasis. Algorithms for BBO/DFO repeatedly call an oracle (a black box, to which BBO refers) that returns for any given $x \in \mathbb{R}^n$ a real function value $f(x)$ uniquely determined by x, possibly also ∞ or `NaN` (not a number). In this way, they try, using no other information about f (such as continuity, Lipschitz constants, differentiability, or derivative information, to which DFO refers), to find a minimizer of the underlying function f. However, although no such information is used for executing the algorithms, the motivation and analysis of the algorithms always assumes, at least informally, that the function f has reasonable mathematical properties. To be able to give performance guarantees, such properties are essential.

1.1 Related work

A huge amount of literature exists about BBO problems and how to solve them, and we only mention a few pointers to the literature. A thorough survey for derivative-free optimization methods was given by Larson et al. [41]. Another useful paper suggested by Rios & Sahinidis [58] discusses the practical behaviour of derivative-free optimization software packages. The techniques for solving BBO problems fall into two classes, **deterministic** and **randomized** methods. We mainly discuss the randomized case; for deterministic methods see, e.g., the book by Conn et al. [15] and its many references. Randomized methods for BBO going back by Rastrigin [57], Polyak [51], and van Laarhoven & Aarts [60] were later discussed especially in the framework of evolutionary optimization [8,31,59]. There are also randomized BBO optimization algorithms for deterministic problems, e.g., Bandeira et al. [9] and Holland [33]. For deterministic global BBO optimization see, e.g., Hansen [30] and for stochastic global BBO optimization see, e.g., Zhigljavsky [63]. Other useful BBO references are Audet & Hare [4], Moré & Wild [45], and Müller & Woodbury [46].

Previous BBO software of the optimization group at the university of Vienna includes the deterministic algorithms **GRID** [23, 24] and **MCS** [34] and the randomized algorithms **SnobFit** [35] and **VXQR** [49]. Software by many others is mentioned in Section 7.3.

1.2 Known complexity results

This section discusses known complexity results in the deterministic and randomized cases.

Throughout the paper, we use a scaled 2-norm $\|p\|$ and its dual norm $\|g\|_*$ of $p, g \in \mathbb{R}^n$, defined in terms of a positive scaling vector $s \in \mathbb{R}^n$ by

$$\|p\| := \sqrt{\sum_i p_i^2/s_i^2}, \quad \|g\|_* := \sqrt{\sum_i s_i^2 g_i^2}. \tag{1}$$

For the choice of a suitable scaling vector see Section 5.

Assumptions. For the mathematical analysis of our algorithm we assume, as customary in the literature on complexity results, that
(A1) the function f is continuously differentiable on $\mathbb{R}^n$, and the gradient $g(x)$ of $x \in \mathbb{R}^n$ is Lipschitz continuous with Lipschitz constant L;
(A2) the level set $\mathcal{L}(x^0) := \{x \in \mathbb{R}^n \mid f(x) \le f(x^0)\}$ of f at x^0 is compact.
Under these conditions, a global minimizer $\widehat{x}$ of f exists and

$$\widehat{f} := f(\widehat{x}) := \inf\{f(x) \mid x \in \mathbb{R}^n\} \tag{2}$$

is finite. x is called an **ε-approximate stationary point** if

$$\|g(x)\|_* \le \varepsilon \tag{3}$$

holds. For fixed $\varepsilon > 0$, ε-approximate stationary points may also exist in regions where the graph of f is sufficiently flat, although no stationary point is nearby. If such a point is encountered, the convergence speed of optimization methods may be extremely slowed down so that a spurious apparent local minimizer is found.

If $\sigma > 0$ and the condition

$$f(x') \ge f(x) + g(x)^T(x' - x) + \frac{\sigma}{2}\|x' - x\|^2 \quad \text{for all } x, x' \in \mathbb{R}^n \tag{4}$$

holds then f is called **σ-strongly convex**. In this case, the global optimizer $\widehat{x}$ is unique, and (3) guarantees that iterations are near $\widehat{x}$.

Proposition 1 *If $f(x)$ is a σ-strongly convex quadratic function, then* (3) *implies $f(x) - \widehat{f} \le \varepsilon^2/(2\sigma)$ and $\|x - \widehat{x}\|^2 \le \varepsilon/\sigma^2$ for $x \in \mathbb{R}^n$.*

Proof For fixed x, the right-hand side of (4) is a convex quadratic function of $\widehat{x}$, minimal when its gradient vanishes. By (1), this is the case iff $\widehat{x}_i$ takes the value $x_i - \dfrac{s_i}{\sigma} g_i(x)$ for $i = 1, \cdots, n$, so that $\widehat{f} \geq f(x) - \dfrac{1}{2\sigma}\|g(x)\|_*^2$ for $x \in \mathbb{R}^n$. Therefore we conclude from (3) and (4) that for $x \in \mathbb{R}^n$

$$f(x) - \widehat{f} \leq \frac{1}{2\sigma}\|g(x)\|_*^2 \leq \frac{\varepsilon^2}{2\sigma} \quad \text{and} \quad \|x - \widehat{x}\|^2 \leq \frac{2}{\sigma}\Big(f(x) - \widehat{f} - g(\widehat{x})^T(x - \widehat{x})\Big) \leq \frac{\varepsilon}{\sigma^2}$$

since the gradient vanishes at $\widehat{x}$. $\square$

In exact precision arithmetic, the exact gradient vanishes at a stationary point. But in finite precision arithmetic optimization methods may get stuck in nearly flat regions, so that a spurious apparent local minimizer may be found. For a finite termination a theoretical criterion should be used to get an ε-approximate stationary point. For a given threshold $\varepsilon > 0$, a complexity bound of an unconstrained BBO method tells how many function evaluations, $N(\varepsilon)$, are needed to find with a given probability (or a related goal) a point x^{best} whose function value $f(x^{\text{best}})$ is below the initial function value $f(x^0)$ and the unknown gradient norm $\|g(x^{\text{best}})\|_*$ at this point is below ε, i.e.,

$$f(x^{\text{best}}) \leq \sup\{f(x) \mid x \in \mathbb{R}^n, \|g(x)\|_* \leq \varepsilon, \text{ and } f(x) \leq f(x^0)\}. \tag{5}$$

(3) says that, in term of function evaluations, x^{best} is at least as good as the worst ε-approximate stationary point with the function value at most $f(x^0)$. Since gradients and Lipschitz constants are unknown to us, we could not say which point satisfies (5). But the result implies that the final best point has a function value equal to or better than some point whose gradient was small. If gradients are small only nearby a global optimizer, it will produce a point close to the local optimizer. If some iterate passes close to a non-global local optimizer or a saddle point, it is possible that the algorithm escapes its neighborhood. In this case, only a variant with restarts would produce convergence to a point with a small gradient.

Under the assumptions (A1)–(A2), the appropriate asymptotic form for the expression $N(\varepsilon)$, found by Vicente [62], Dodangeh & Vicente [20], Dodangeh, Vicente & Zhang [21], Gratton et al. [27], Bergou, Gorbunov & Richtárik [11], and Nesterov & Spokoiny [47,48], depends on the properties (smooth, smooth convex, or smooth strongly convex) of f; cf. Subsection 2.1 below.

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 randomized BBO in expectation (Bergou et al. [11] for all cases)

Bergou et al. [11] and Nesterov & Spokoiny [48] 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 [20], Vicente [62] , and Konečný & Richtárik [14]) given in Table 2. Of course, being a randomized 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 improved or not. This is the reason why the algorithms proposed by Bergou et al. [11] are numerically poor, see Section 7.

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

Table 2: Complexity results for deterministic BBO (Vicente [62] for the nonconvex case, Dodangeh & Vicente [20] for the convex and the strongly convex cases, Konečný & Richtárik [14] for all cases)

The best complexity bound for a direct search with probabilistic (rather than expectation) guarantees has been found by Gratton et al. [27], only for nonconvex case. They used Chernoff bounds to prove that a 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, satisfying

$$R > \log\left(1 - \frac{\ln(\gamma_1)}{\ln(\gamma_2)}\right),$$

where $0 < \gamma_1 < 1$ is a factor for reducing step sizes and $\gamma_2 > 1$ is a factor for expanding step sizes. If $\gamma_1 = 0.5$ and $\gamma_2 = 2$, then $R = 2$.

1.3 Our contribution

We describe and test a new, practically very efficient randomized method, called **VRBBO** (short for **Vienna randomized black box optimization**), for which good local complexity results can be proved, and which is competitive in comparison with the state-of-the-art local and global BBO solvers. An algorithm loosely related to **VRBBO** (but without complexity guarantees) is the **Hit-and-Run** algorithm by Bélisle [10].

A basic version of VRBBO. In Section 2, an extrapolation step, called **extrapolationStep** is discussed and then a multi-line search with random directions, called **MLS-basic**, is constructed, trying **extrapolationStep**. Section 3 first introduces a basic version of our fixed decrease search algorithm, called **FDS-basic** to hopefully get a decrease in the function value. It has repeated calls to **MLS-basic** until the function value is decreased. Then a basic version of **VRBBO**, called **VRBBO-basic**, is introduced, which has repeated calls to **FDS-basic** until an ε approximate stationary point is found; see Flowchart 1.

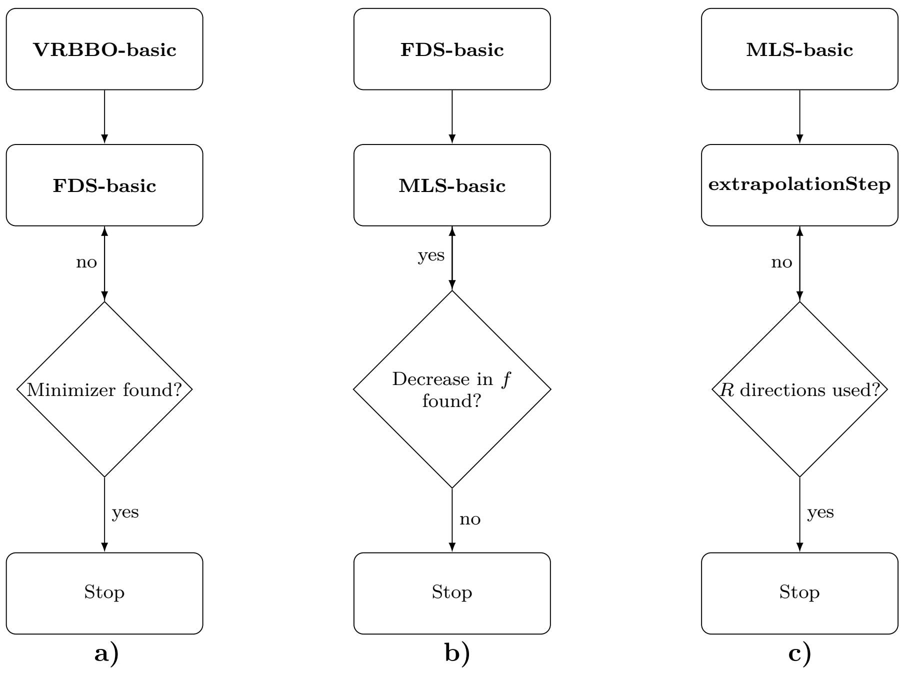


Fig. 1: Flowchart for (a) **VRBBO-basic**, (b) **FDS-basic**, (c) **MLS-basic**. Here R is the number of random direction used by **MLS-basic**.

Complexity results for VRBBO-basic. Section 4 discusses our complexity bound for the nonconvex case with the same order and factor as the one found by Gratton et al. [27] obtained by Chernoff bound. But with the difference that the constant factor of our bound is obtained from a result of Pinelis [50]. Both complexity results are better by the factor of R/n than those given in Table 1 and are more reasonable than those given in Table 2. Our complexity bounds for the convex and strongly convex cases are proven 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. [27] for the nonconvex case. As discussed in Section 1.2, Gratton et al.'s results for the nonconvex case allow $R = 2$, while **VRBBO-basic** needs

$$R = \Omega(\log \eta^{-1}) \quad \text{for a given } 0 < \eta \le \tfrac{1}{2}.$$

But $\log \eta^{-1}$ cannot be large for reasonable value of η. In practice, **VRBBO-basic** works best with a much larger value $R = \mathcal{O}(n)$.

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

Table 3: Complexity results for randomized BBO with probability $1-\eta$, for fixed $0 < \eta \le \frac{1}{2}$ (Gratton et al. [27] for the nonconvex case with $R = 2$ and present paper for all cases, with $R = \Omega(\log \eta^{-1})$.

Heuristic techniques. We add many new useful heuristic techniques – discussed in Section 5 – to **VRBBO-basic** that make it very competitive, in particular:
• Several kinds of search directions ensure good practical performance.
• Adaptive heuristic estimations for the Lipschitz constant are used.
• A sensible scaling vector is estimated.
• The gradient vector is estimated by a randomized finite difference approach.
These heuristic techniques improve the performance in practice, leading to the **FDS** and **VRBBO** implemented documentations in Section 6.

Numerical results. In Section 7 we compare all solvers (including some good global ones) on the unconstrained `CUTEst` test problems by Gould [26] for optimization and the test problems, called `GlobalTest`, by Jamil & Yang [37] for global optimization with 2 up to 5000 variables. The numerical results of show that **VRBBO** matches the quality of global state-of-the-art algorithms for finding, a global minimizer with reasonable accuracy. Although our theory guarantees only local minimizers, the **FDS** together with our heuristic techniques turn **VRBBO** into an efficient global solver. For example, **FDS** takes for large Δ initially only large steps, hence has a global character.

2 A new line search technique

In this section, we describe a method that tries to achieve a decrease in the function value using line searches along specially chosen random directions. In our algorithm random directions are used because it is known that randomized black box optimization methods have a worst case complexity by a factor of n lower than that of deterministic algorithms (see cf. [9]).

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 basic multi-line search, for which strong probabilistic results can be proved.

The details are chosen in such a way that failure to achieve the desired descent implies that, with probability arbitrarily close to one, a bound on the unknown gradient vector is obtained.

2.1 Probing a direction

Let $\mathit{\Delta} \geq 0$ be the **threshold for improvements** on the function value and let $f(x) - f(x \pm p)$, for every $x, p \in \mathbb{R}^n$, be the **gain** along $\pm p$. First we give a theoretical test that either results in a gain of $\mathit{\Delta}$ or more in function value, or gives a small upper bound for the norm of at least one of the unknown gradients encountered though our algorithm never calculates ones.

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

$$f(x+p) - f(x) = g(x)^T p + \frac{1}{2}\gamma \|p\|^2, \tag{6}$$

where γ depends on x and p and satisfies one of

$$|\gamma| \leq L, \quad \text{(general case)} \tag{7}$$

$$0 \leq \gamma \leq L, \quad \text{(convex case)} \tag{8}$$

$$0 < \sigma \leq \gamma \leq L. \quad \text{(strongly convex case)} \tag{9}$$

Here σ comes from (4). In all three cases,

$$g(x)^T p - \frac{1}{2}L\|p\|^2 \leq f(x+p) - f(x) \leq g(x)^T p + \frac{1}{2}L\|p\|^2. \tag{10}$$

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

$$r_0 := \sup \Big\{ \|x - \widehat{x}\| \mid x \in \mathbb{R}^n \ \text{ and } \ f(x) \leq f(x^0) \Big\} < \infty. \tag{11}$$

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

$$L \geq \frac{|f(x+p) + f(x-p) - 2f(x)|}{\|p\|^2}, \tag{12}$$

and at least one of the following holds:
(i) $f(x+p) < f(x) - \mathit{\Delta}$,
(ii) $f(x+p) > f(x) + \mathit{\Delta}$ and $f(x-p) < f(x) - \mathit{\Delta}$,
(iii) $|g^T p| \leq \mathit{\Delta} + \frac{1}{2}L\|p\|^2$.

Proof Taking the sum of (10) and the formula obtained from it by replacing p with $-p$ gives (12).

Assume that (iii) is violated, so that $\mathit{\Delta} + \frac{1}{2}L\|p\|^2$. If $g(x)^T p \leq 0$, then by (10)

$$f(x+p) - f(x) \leq g(x)^T p + \frac{1}{2}L\|p\|^2 = -|g^T p| + \frac{1}{2}L\|p\|^2 < -\mathit{\Delta}. \tag{13}$$

If $g(x)^Tp \geq 0$, then similarly

$$f(x-p) - f(x) \leq g(x)^T(-p) + \frac{1}{2}L\|p\|^2 = -|g^Tp| + \frac{1}{2}L\|p\|^2 < -\varDelta. \tag{14}$$

If (13) holds we conclude that (i) holds. If (14) holds we get the second half of (ii), and the first half follows from

$$f(x+p) - f(x) \geq g(x)^Tp - \frac{1}{2}L\|p\|^2 > \varDelta.$$

□

Proposition 2 will play a key role in the construction of our basic multi-line search **MLS-basic** detailed in Subsection 2.3:
• It establishes the well-known (Evtushenko [25]) lower bound (12) for the Lipschitz constant L which can be used to find reasonable approximations for L.
• If (i) holds, then the step p gives a gain of at least $\varDelta$, called the **sufficient gain**.
• If (ii) holds, then the step $-p$ gives a sufficient gain.
• If neither (i) nor (ii) holds (no sufficient gain is found along $\pm p$) then (iii) holds, giving a useful upper bound for the directional derivative.

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

2.2 Random search directions

For our complexity results, we need that sufficiently many search directions p satisfy the **angle condition** of the form

$$\sup \frac{g^Tp}{\|g\|_*\|p\|} \leq -\varDelta^{\mathrm{a}} < 0. \tag{15}$$

Here g is the gradient of the current best point and $\varDelta^{\mathrm{a}} > 0$ is a tuning parameter for the angle condition.

Random directions are uniformly independent and identically distributed (**i.i.d**) in $[-\frac{1}{2}, \frac{1}{2}]^n$, computed by

$$p = \mathrm{rand}(n,1) - 0.5, \tag{16}$$

where $\mathrm{rand}(n,1)$ generates a random vector uniformly distributed in $[0,1]^n$.

The following variant of the angle condition (15) plays a key role to get our complexity bounds.

Proposition 3 *For random search directions generated by* (16) *and scaled by*

$$p := p(\delta/\|p\|). \tag{17}$$

satisfies $\|p\| = \delta$ *and, with probability* $\geq \dfrac{1}{2}$,

$$\|g(x)\|_* \|p\| \leq 2\sqrt{cn}|g(x)^T p| \tag{18}$$

with a positive constant $c \leq 12.5$.

Proof As defined earlier in Section 1.2, $s \in \mathbb{R}^n$ is a scaling vector. Denote by s_i the ith component of s and define $\overline{p}_i := p_i/s_i$ and $\overline{g}_i := s_i g_i$. Then by (1), $g^T p = \overline{g}^T \overline{p}$ and $\|g\|_* = \|\overline{g}\|_2$ and $\|p\| = \|\overline{p}\|_2$; so the results of Subsection 9.1 apply after scaling and give $c = c_0/4 \leq 12.5$. □

This simulation result from Subsection 9.1 suggests that $c \approx 4/7$.

2.3 A multi-line search

In this section, we construct a multi-line search algorithm, called **MLS-basic**. It polls in random directions (satisfying (18), with probability $\geq \frac{1}{2}$, generated by (16), and scaled by (17)) in a line search fashion a few objective function values each in the hope of finding sufficient gains by more than a multiple of Δ.

2.3.1 An extrapolation step

As discussed in Section 1.3, the main ingredient of **VRBBO-basic** is **FDS-basic** which has repeated calls to **MLS-basic** until at least a sufficient gain is found. The accelerated ingredient of **MLS-basic** is **extrapolation** whose goal is to speed up reaching a minimizer by expanding step sizes and computing the corresponding trial points and their function values as long as sufficient gains are found. We discuss how to construct extrapolation steps, called **extrapolationStep**, trying to hopefully find sufficient gains. **extrapolationStep** may perform extrapolation along either the search direction p or its opposite direction.

Let $\{x^k\}_{k\geq 0}$ be the sequence generated by **VRBBO-basic**. In the kth iteration of this algorithm, **FDS-basic** takes as input the $(k-1)$th point `xm` $= x^{k-1}$ and its function value `fm` $= f_{k-1}$ generated by **VRBBON-basic** and returns the kth point $x^k =$ `xm` and its function value $f_k =$ `fm` as output if at least a sufficient gain is found by **MLS-basic**; otherwise $x^k = x^{k-1}$ and $f_k = f_{k-1}$. In fact, after the kth iteration of **VRBBON-basic** is performed, `xm` is the current trial point evaluated by **extrapolationStep**, obtained from a sufficient gain, accepted as a new point, and called the **best point**. Hence all points x^k, for $k = 1, 2, \cdots$, are the best points found by **extrapolationStep**. The last point generated by **VRBBO-basic** is said the **overall best point**.

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 $\texttt{xm}$ for the best point found by **extrapolationStep** 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

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

At this stage, we can compute the lower bound

$$\lambda := \max(\lambda_{\text{old}}, |\texttt{fl}+\texttt{fr}-2\texttt{fm}|/\delta^2) \tag{20}$$

for the Lipschitz constant L, valid by (12). Note that the initial λ_{old} is the tuning parameter $\lambda_{\max}$, however, it is updated by **extrapolationStep** and may be estimated by a heuristic formula.

As defined earlier in Subsection 2.1, $\texttt{df} := \texttt{fm}-\texttt{fr}$ is the gain and given the tuning parameter $0 < \gamma_{\min} < 1$, if the condition

$$\texttt{df} > \overline{\Delta} = \gamma_{\min}\Delta \tag{21}$$

holds, a sufficient gain is found and the corresponding point is updated by overwriting $\texttt{xm}+p$ over $\texttt{xm}$, with the consequence that in this case

$$\texttt{fl} := f(\texttt{xm}-2p), \quad \texttt{fm} := f(\texttt{xm}-p), \quad \texttt{fr} := f(\texttt{xm}). \tag{22}$$

R denotes the number of the random search directions used in **MLS-basic** and $\mathbf{a}$ denotes the list of R **extrapolation step sizes**. All components of the initial list $\mathbf{a}$ are one, i.e., $\mathbf{a}(t) = 1$ for $t = 1, \cdots, R$. These components are expanded or reduced according to whether sufficient gains are found or not. Let n_{sg} be the number of sufficient gains found by **extrapolationStep** to exceed sufficient gains. If the counter n_{sg} remains zero, **extrapolationStep** cannot find a sufficient gain. t is a counter for R taking $0, \cdots, R$. It does not change inside **extrapolationStep**, but it is updated later outside **extrapolationStep** (inside **MLS-basic**).

We must be careful to make sure that the estimation of the Lipschitz constant is correct, especially when an extrapolation step – improving the function value – is tried. This estimation is computed
(i) after an opposite direction is tried. Since there is no sufficient gain along the direction p, its opposite direction is tried. Then λ is estimated according to (19).
(ii) after the first sufficient gain is found. For this estimation, $\texttt{fl}$, $\texttt{fm}$, $\texttt{fr}$ are needed. Since a sufficient gain is found, according to (22), the Lipschitz constant λ is estimated by (20).

In summary, **extrapolationStep** first takes the initial step size $\alpha_e = 1$, which is necessary to approximate a lower bound for the unknown Lipschitz constant L. Then it chooses step sizes from $\mathbf{a}(t)$ later while expanding it until a sufficient gain is found. After a sufficient gain is found α_e is saved as the new $\mathbf{a}(t)$. One of the following cases is happened:
(i) A sufficient gain is found along the direction p.
(ii) A sufficient gain is found along the direction $-p$.
(iii) No sufficient gain is found along $\pm p$.
If either (i) or (ii) holds, **extrapolationStep** is successful at least with a sufficient gain. But if (iii) holds **extrapolationStep** is unsuccessful without any sufficient gain.

Algorithm 1 extrapolationStep, an extrapolation step

Input. The old best point `xm` and its function value `fm`, the search direction p, the threshold for good improvement $\varDelta > 0$, the old approximation $\lambda_{\text{old}} \geq 0$ for the Lipschitz constant L, the tth extrapolation step size $\mathbf{a}(t)$, the norm of trial step δ, and maximum number of function evaluations `nfmax`.

Tuning parameters. $\gamma_e > 1$ (the factor for reducing/expanding step sizes), $\gamma_{\min}$ (the factor for extrapolation test), $E \geq 1$ (maximum number of extrapolations).

Output. A newest best point `xm` and its function value `fm`, a new approximation λ for the Lipschitz constant L, the number n_{sg} of sufficient gains, and the tth extrapolation step size $\mathbf{a}(t)$.

```
    Initialization
1:  Initialize $\alpha_e := 1$ and set $\lambda := \lambda_{\text{old}}$.     ▷ necessary to approximate $\lambda$ below
2:  for $n_{\text{sg}} = 0, \cdots, E$ do
      Try either an extrapolation step along $p$ or $-p$
3:    Compute the trial point xr := xm + $\alpha_e p$ and its function value fr := $f$(xr).
4:    Compute the gain df := fm − fr.
      Stopping test
5:    if nfmax is reached then     ▷ maximal number of function evaluation is reached
6:      if df > 0 then     ▷ a gain is found; update the best point before a termination
7:        Update xm := xr and fm := fr and stop extrapolationStep.
8:      end if
9:    end if
      Estimate the Lipschitz constant $L$ after the first sufficient gain is found
10:   if $n_{\text{sg}}$ reaches one then     ▷ after the first extrapolation is done
11:     Set ft := fm.     ▷ temporarily save the old best function value
12:     Set fm := fe.     ▷ temporarily restore the new best function value
13:     Estimate $L$ by
            $\lambda := \max(\lambda, |\texttt{fl} + \texttt{fr} - 2\texttt{fm}|/\delta^2)$.
14:     Set fm := ft.     ▷ restore the old best function value
15:   end if
      Check whether a sufficient gain is found or not?
16:   if df > $\gamma_{\min}\varDelta$ then     ▷ a sufficient gain is found
17:     if $n_{\text{sg}}$ reaches one then     ▷ the first iteration of an extrapolation
18:       Set fl := fm.     ▷ save the old best function value
19:       Set $\alpha_e := \mathbf{a}(t)$.     ▷ choose the extrapolation step size from $\mathbf{a}(t)$
20:     else
21:       Expand the step size to $\alpha_e := \gamma_e \alpha_e$.
22:       Set fe := fr.     ▷ since fr < fe during an extrapolation
23:     end if
24:   else if $-p$ has been not tries already then     ▷ an extrapolation along $-p$ is tried
25:     Replace $p$ by $-p$.     ▷ opposite direction is chosen
        Estimate $L$ after finding an opposite direction is tried
26:     Set fl := fr.     ▷ update the third former function value
27:     Estimate $L$ by
            $\lambda := \max(\lambda, |\texttt{fl} + \texttt{fr} - 2\texttt{fm}|/\delta^2)$.
28:   else     ▷ no sufficient gain is found along $\pm p$
29:     Terminate the for loop.
30:   end if
31: end for
    Updating the best point and its function value
32: if $n_{\text{sg}} > 0$ then     ▷ extrapolation ends with a sufficient gain
33:   Reduce the step size to $\alpha_e := \alpha_e/\gamma_e$.     ▷ no sufficient gain by the last point
34:   Set xm := xm + $\alpha_e p$, $\mathbf{a}(t) := \alpha_e$, and fm := fe.     ▷ updating the best point information
35: end if
```

2.3.2 A basic version of the **MLS** *algorithm*

For each random direction generated, our basic multi-line search (**MLS-basic**) using **extrapolationStep** is performed where the following happens:
• A step in the current direction is tried.
• If a sufficient gain is found, a sequence of extrapolations is tried.
• If a sufficient negative gain is found, a step in the opposite direction is tried.
• If a sufficient gain is found in the opposite direction, a sequence of extrapolations is tried.
• If no sufficient gain along $\pm p$ is found, the step size is reduced.

Algorithm 2 MLS-basic, a basic multi-line search

Input. The old best point `xm` and its function value `fm`, the threshold $\varDelta > 0$ for good improvement, the old approximation $\lambda \geq 0$ for the Lipschitz constant L, maximal number `nfmax` of function evaluations, and the list $\mathbf{a}$ of extrapolation step sizes.

Tuning parameters. $\gamma_e > 1$ (the factor for reducing/expanding step sizes), $\gamma_{\min}$ (the factor for extrapolation test), $E \geq 1$ (maximum number of extrapolations), $\delta_{\min}/\delta_{\max}$ (minimum/maximum norm of trial steps), γ_δ (factor for adjusting δ), $\alpha_{\min} \in (0, 1)$ (minimum threshold for step sizes), $R > 0$ (the number of random search directions).

Output. A newest best point `xm` and its function value `fm`, a new approximation $\lambda \geq 0$ for the Lipschitz constant L, the number $n_{\rm sg}$ of sufficient gains, and an updated list $\mathbf{a}$ of extrapolation step sizes.

```
     Initialization
 1: Set $\lambda^1 := \lambda$.
 2: for $t = 1, \cdots, R$ do
        Compute search direction
 3:     Compute the $t$th random direction $p^t$ by rand$(n, 1) - 0.5$,
 4:     Update $\delta^t$ by
            $$\delta^t := \max\left(\delta_{\min}, \min\left(\sqrt{\mathbf{a}(t)\gamma_\delta \varDelta/\lambda^t}, \delta_{\max}\right)\right). \quad (23)$$
 5:     Scale the $t$th search direction by $p^t := p^t(\delta^t/\|p^t\|)$.
        Perform an extrapolation
 6:     Perform: $[\mathtt{xm}, \mathtt{fm}, \lambda^{t+1}, \mathbf{a}, n_{\rm sg}] = \mathbf{extrapolationStep}(\mathtt{xm}, \mathtt{fm}, \lambda^t, \mathbf{a}, \varDelta)$.
        Stopping test
 7:     if nfmax is reached then            ▷ maximal number of function evaluation is reached
 8:         Stop MLS-basic.
 9:     end if
        Reducing step sizes
10:     if $n_{\rm sg}$ is zero then                    ▷ there is no sufficient gain
11:         Reduce the $t$th step size to
            $$\mathbf{a}(t) := \max(\mathbf{a}(t)/\gamma_e, \alpha_{\min}). \quad (24)$$
12:     end if
13: end for
```

In (24), the extrapolation step sizes need to be reduced whenever no sufficient gain is found. Hence, they need to be controlled by the tuning parameter $\alpha_{\min}$.

(23) is motivated at the end of this section since it is based on the details of Theorem 1, which asserts that one obtains either a sufficient gain of multiple of Δ or, with probability arbitrarily close to 1, an upper bound of $\|g\|_*$ for at least one of the unknown gradients encountered though our algorithm never calculates ones.

Theorem 1 *Assume that (A1) holds and let* `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* **MLS-basic**. *Moreover, let $\overline{\Delta} = \gamma_{\min}\Delta$ with $0 < \gamma_{\min} < 1$. Here* `nfmax` *is assumed to be sufficiently large.*
(i) f decreases by at least

$$\overline{\Delta}_f := \overline{\Delta}\max(\mathtt{nf} - 2R - 1, 0) \tag{25}$$

(Note that $\overline{\Delta}_f$ may be zero, catering for the case of no strict decrease).
(ii) Suppose that $0 < \eta \le \frac{1}{2}$ and $R := \lceil \log_2 \eta^{-1} \rceil$. If f does not decrease by more than a multiple of Δ then, with probability $\ge 1 - \eta$, the original point or one of the points evaluated with better function values has a gradient g with

$$\|g\|_* \le \sqrt{cn}\,\Gamma(\delta), \tag{26}$$

where c is the constant in Proposition 3 and $\Gamma(\delta)$ is defined by

$$\Gamma(\delta) := L\delta + \frac{2\Delta}{\delta} \quad \textit{for some } \delta > 0. \tag{27}$$

Proof (i) Clearly, the function value of the best point does not increase. Thus (i) holds if $\mathtt{nf} - 2R - 1 \le 0$. If this is not the case, then $\mathtt{nf} \ge 2R + 2$. But in the for loop of **MLS-basic**, 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 $\mathtt{nf} - 2R - 1$ additional function values are computed during the extrapolation stage, each time with a sufficient gain of at least $\overline{\Delta}$. Thus the total sufficient gain is at least (25).
(ii) Assume that f does not decrease by more than $\overline{\Delta}$. For $t = 1, \dots, R$, let p^t be the tth random direction generated by (17), and let x^t be the best point obtained before searching in direction p^t. Then, from Proposition 2, we get

$$|g(x^t)^T p^t| \le \overline{\Delta} + \frac{L}{2}\|p^t\|^2 \le \Delta + \frac{L}{2}\|p^t\|^2 = \frac{\delta}{2}\Gamma(\delta), \quad \text{for all } t = 1, \dots, R.$$

Since the random direction is generated by (16), Proposition 3 implies that

$$\|g(x^t)\|_* = \|g(x^t)\|_*\|p^t\|/\delta \le \Big(2\sqrt{cn}|g(x^t)^T p^t|\Big)/\delta \le \sqrt{cn}\,\Gamma(\delta), \quad \text{for all } t = 1, \dots, R$$

holds with probability $\frac{1}{2}$ or more. Therefore $\|g(x^t)\|_* \leq \sqrt{cn}\Gamma(\delta)$ fails with a probability less than $\frac{1}{2}$ for all $t = 1, \ldots, R$. Therefore, the probability that (26) holds for at least one of the gradients $g = g(x^t)$ $(t \in \{1, \ldots, R\})$ is

$$1 - \prod_{t=1}^{R-1} \Pr\Big(\|g(x^t)\|_* > \sqrt{cn}\Gamma(\delta)\Big) \geq 1 - 2^{-R}.$$

$\square$

Note that (26) is guaranteed to hold although gradients are never computed. Since gradients and Lipschitz constants are unknown to us, we could not say which point satisfies (26). But the result implies that the final best point has a function value equal to or better than some point whose gradient was small. If gradients are small only nearby a global optimizer, it will produce a point close to the local optimizer. If some iterate passes close to a non-global local optimizer or a saddle point, it is possible that the algorithm escapes its neighborhood. In this case, only a variant with restarts would produce convergence to a point with a small gradient.

As discussed earlier in Section 1.3, **VRBBO-basic** tries to find a point satisfying (5). The goal of the scaling of the search direction (16) is that the bound $\sqrt{cn}\Gamma(\delta)$ in (26) becomes below a given threshold $\varepsilon > 0$. This is done by minimizing such a bound. For fixed Δ, the scale-dependent factor (27) is smallest for the choice

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

Accordingly, (23) is used to scale the random directions (16), safeguarded by the sensible positive lower and upper bounds $0 < \delta_{\min} < \delta_{\max} < +\infty$. As can be seen from Subsection 2.3.1, α_e is used for estimating L and here for adjusting δ. Due to our experience, using the unfixed α_e in (23) is useful. In fact, $\widehat{\delta}$ is a special case of the term $\sqrt{\alpha_e\gamma_\delta\Delta/\lambda}$ in (23) with $\alpha_e = 1$ and $\gamma_\delta = 2$.

3 A randomized descent algorithm for BBO

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

3.1 Probing for fixed decrease

Based on the preceding results, we introduce the basic version of a fixed decrease search algorithm (**FDS-basic**) whose goal is to perform calls to the basic multi-line search **MLS-basic** to hopefully find sufficient gains by a multiple of $\varDelta$. If either there is no sufficient gain ($n_{\rm sg} = 0$) by **MLS-basic** or `nfmax` is reached, **FDS-basic** ends. The main ingredient of **VRBBO-basic** is **FDS-basic** which takes for large $\varDelta$ many large steps, hence has a global character.

In the next algorithm, it is assumed that **FDS-basic** is tried in the kth iteration of **VRBBO-basic**.

Algorithm 3 FDS-basic, a basic fixed decrease search

Input. The old best point `xm` and its function value `fm`, the threshold $\varDelta_{k-1} > 0$ for good improvement, $\lambda^{k-1} \ge 0$ for the Lipschitz constant L, maximum number `nfmax` of function evaluations, and the list $\mathbf{a}^{k-1}$ of extrapolation step sizes at the $(k-1)$th iteration of **VRBBO**.

Tuning parameters. $\gamma_e > 1$ (the factor for reducing/expanding step sizes), $\gamma_{\min}$ (the factor for extrapolation test), $E \ge 1$ (maximum number of extrapolations), $\delta_{\min}/\delta_{\max}$ (minimum/maximum norm of trial steps), γ_δ (factor for adjusting δ), $\alpha_{\min} \in (0,1)$ (minimum threshold for step sizes), $R > 0$ (the number of random search directions).

Output. A new best point `xm` and its function value `fm`, a new approximation $\lambda^k \ge 0$ for the Lipschitz constant L, and an updated list $\mathbf{a}^k$ of extrapolation step sizes at the kth iteration of **VRBBO**.

```
   Initialization
1: Set $\lambda^k := \lambda^{k-1}$, $\mathbf{a}^k := \mathbf{a}^{k-1}$, $\varDelta_k := \varDelta_{k-1}$.
2: while true do
     To hopefully get sufficient gains
3:   Perform: $[\mathtt{xm}, \mathtt{fm}, \lambda^k, \mathbf{a}^k, n_{\rm sg}] = \mathbf{MLS\text{-}basic}(\mathtt{xm}, \mathtt{fm}, \lambda^k, \mathbf{a}^k, \varDelta_k)$ .
     Stopping test
4:   if nfmax is reached or $n_{\rm sg}$ is zero then
5:     FDS-basic ends.
6:   end if
7: end while
```

Theorem 2 *Assume that (A1) and (A2) hold,* `nfmax` *is sufficiently large, denote by f_0 the initial value of f and let $\overline{\varDelta} = \gamma_{\min}\varDelta$ with the tuning parameter $0 < \gamma_{\min} < 1$. Then:*
(i) The number of function evaluations of **FDS-basic** *is bounded by*

$$2R + (2R+1)\frac{f_0 - \widehat{f}}{\overline{\varDelta}},$$

where $\widehat{f}$ is the global minimum value.
(ii) Denote by K_f the number of calls to **MLS-basic** *by* **FDS-basic** *and assume that*

$$0 < \eta \le \frac{1}{2}, \quad R = \lceil \log_2 \eta^{-1} \rceil, \quad \text{and} \quad 0 < \delta_{\min} < \delta_{\max} < \infty.$$

Then **FDS-basic** *finds a point x, with probability $\ge 1 - \eta$, satisfying*

$$\|g(x)\|_* \le \sqrt{cn} \min_{t=0:K_f} \varGamma(\delta^t) \le \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\varDelta} + \frac{2\varDelta}{\delta_{\max}}\Big). \tag{28}$$

Here c is the constant from Proposition 3 and, if λ^0 denotes the value of λ before the first execution of **FDS-basic**,

$$L' := \frac{L^2\gamma_\delta}{\lambda^0} + 4L + 4\frac{\lambda^0 + L}{\gamma_\delta} \quad \textit{with } \gamma_\delta > 0. \tag{29}$$

Proof By (A2), $\widehat{f}$ is finite. Denote by f_{k+1} the result of the $(k+1)$th execution of **FDS-basic**. In the worst case in each iteration $\ell \in \{1, \cdots, k\}$ of **FDS-basic** a sufficient gain is found, i.e., the condition

$$f_\ell \le f_{\ell-1} - \overline{\varDelta} \text{ for } \ell \in \{1, \cdots, k\}$$

holds. But in the $(k+1)$th iteration **FDS-basic** cannot find any sufficient gain and ends. We then conclude that

$$\widehat{f} \le f_k \le f_0 - \sum_{i=1}^{k} \overline{\varDelta} \le f_0 - k\overline{\varDelta} = f_0 - k\overline{\varDelta}$$

by (24), so that $k \le (f_0 - \widehat{f})/\overline{\varDelta}$.

Since a sufficient gain is found in each iteration $\ell = 1, \cdots, k$, $2R+1$ function evaluations are used. But in the $(k+1)$th iteration, $2R$ function evaluations are used since there is no sufficient gain. Hence (i) follows.

(ii) K_f is finite due to (i) and we have $2^{-R} \le \eta$. Hence by Theorem 1 with probability $\ge 1 - 2^{-R} \ge 1 - \eta$

$$\|g\|_* \le \sqrt{cn} \min_{t=0:K_f} \varGamma(\delta^t)$$

holds. Thus it is sufficient to show that

$$\varGamma(\delta) \le L\delta_{\min} + \sqrt{L'\varDelta} + \frac{2\varDelta}{\delta_{\max}}. \tag{30}$$

By the definition of δ in (23), we have one of the following three cases:

Case 1: $\delta = \sqrt{\dfrac{\gamma_\delta \varDelta}{\lambda}}$. In this case,

$$\varGamma(\delta) = L\delta + \frac{2\varDelta}{\delta} = L\sqrt{\frac{\gamma_\delta \varDelta}{\lambda}} + 2\sqrt{\frac{\lambda\varDelta}{\gamma_\delta}} = \varLambda\sqrt{\varDelta},$$

where

$$\varLambda := L\sqrt{\frac{\gamma_\delta}{\lambda}} + 2\sqrt{\frac{\lambda}{\gamma_\delta}}. \tag{31}$$

Case 2: $\delta = \delta_{\min} \ge \sqrt{\dfrac{\gamma_\delta \varDelta}{\lambda}}$. In this case,

$$\varGamma(\delta) = L\delta_{\min} + \frac{2\varDelta}{\delta_{\min}} \le L\delta_{\min} + 2\sqrt{\frac{\lambda\varDelta}{\gamma_\delta}} \le L\delta_{\min} + \varLambda\sqrt{\varDelta}.$$

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

$$\varGamma(\delta) = L\delta_{\max} + \frac{2\varDelta}{\delta_{\max}} \le L\sqrt{\frac{\gamma_\delta \varDelta}{\lambda}} + \frac{2\varDelta}{\delta_{\max}} \le \varLambda\sqrt{\varDelta} + \frac{2\varDelta}{\delta_{\max}}.$$

Thus in each case,

$$\varGamma(\delta) \le L\delta_{\min} + \varLambda\sqrt{\varDelta} + \frac{2\varDelta}{\delta_{\max}}.$$

As discussed earlier, L is unknown and we replace it by an approximation value λ. Proposition 2 implies that

$$\lambda^0 \le \lambda \le \max(\lambda^0, L) \le \lambda^0 + L, \tag{32}$$

where λ^0 is the initial value of λ. Now (30) follows since by (31) and (32),

$$\varLambda^2 = \frac{L^2\gamma_\delta}{\lambda} + 4L + \frac{4\lambda}{\gamma_\delta} \le L'.$$

□

3.2 A basic version of the **VRBBO** algorithm

We now have all ingredients to formulate **VRBBO-basic**. It uses in each iteration the fixed decrease search algorithm to update the best point. If either no sufficient gain is found or `nfmax` is not reached in the corresponding **FDS-basic** call, $\varDelta$ is reduced by a factor of Q. Once either $\varDelta$ is below a minimum threshold, **VRBBO-basic** stops.

Algorithm 4 VRBBO-basic, Vienna basic randomized BBO

Input. The initial point x^0 and maximum number `nfmax` of function evaluations.

Tuning parameters. $\gamma_e > 1$ (the factor for reducing/expanding step sizes), $\gamma_{\min}$ (the factor for extrapolation test), $E \geq 1$ (maximum number of extrapolations), $\delta_{\min}/\delta_{\max}$ (minimum/maximum norm of trial steps), γ_δ (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), $R > 0$ (the number of random search directions).

Output. An overall best point `xm` and its function value `fm`.

```
    $\boxed{\text{Initialization}}$
1:  Set $\lambda^0 := \lambda_{\max}$, $\delta^0 := \delta_{\max}$, and $\Delta_0 := \Delta_{\max}$.
2:  Compute the initial function value $f(x^0)$.
3:  for $k = 1, 2, 3, \cdots$ do
        $\boxed{\text{To hopefully find sufficient gains}}$
4:      Choose xm $:= x^{k-1}$ and fm $:= f(x^{k-1})$ as input for FDS-basic.
5:      Perform: $[\mathtt{xm}, \mathtt{fm}, \lambda^k, \mathbf{a}^k] = \textbf{FDS-basic}(\mathtt{xm}, \mathtt{fm}, \lambda^{k-1}, \mathbf{a}^{k-1}, \Delta_{k-1})$.
6:      Choose $x^k :=$ xm and $f(x^k) :=$ fm as the $k$th best point and its function value.
        $\boxed{\text{Stopping test}}$
7:      if nfmax is reached or $\Delta_{k-1} \leq \Delta_{\min}$ then
8:          VRBBO-basic ends.
9:      end if
        $\boxed{\text{Reducing step sizes}}$
10:     Reduce the step size to $\Delta_k := \Delta_{k-1}/Q$.
11: end for
```

As discussed above, $\Delta_{\max}$ and $\lambda_{\max}$ are initially tuning parameters but in an improved version of **VRBBO-basic** they will be estimated by heuristic techniques in Section 5. From Lines 1 and 10, the kth call to **FDS-basic** uses

$$\Delta_k = Q^{1-k}\Delta_{\max}. \tag{33}$$

It will be used in the next section to prove all theorems.

4 Complexity analysis of VRBBO-basic

We now prove the complexity results for the nonconvex, convex, and strongly convex objective functions. We denote by N_k the total number of function evaluations used by **VRBBO-basic** up to iteration k.

4.1 The general (nonconvex) case

Theorem 3 *Let $\{x^k\}$ $(k = 0, 1, 2, \cdots)$ be the sequence generated by* **VRBBO-basic**. *Assume that (A1) and (A2) hold,* `nfmax` *is sufficiently large, and the parameters*

$$0 < \eta \leq \tfrac{1}{2},\ 0 < \gamma_{\min} < 1,\ \Delta_{\max} > 0,\ \delta_{\max} > 0,\ \varepsilon > 0$$

are given. If the parameters are chosen such that

$$\varDelta_{\min} := \Theta(\varepsilon^2/n), \tag{34}$$

$$K := \left\lceil \frac{\log(\varDelta_{\max}/\varDelta_{\min})}{\log Q} \right\rceil, \tag{35}$$

$$R := \left\lceil \log_2 \eta^{-1} \right\rceil, \tag{36}$$

$$\delta_{\min} := \mathcal{O}(\varepsilon/\sqrt{n}), \tag{37}$$

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

$$\|g(x)\|_* = \mathcal{O}(\varepsilon). \tag{38}$$

Proof We conclude from (33)–(35) that

$$\varDelta_\ell = Q^{1-\ell}\varDelta_{\max} \leq \varDelta_{\min} \quad \text{for } \ell \geq K. \tag{39}$$

Hence at most K steps of **FDS-basic** are performed. By (36), we have $\eta_1 = 2^{-R} \leq \eta$. Thus by Theorem 2(ii) we have from (34) and (37), with probability $\geq 1-\eta_1 \geq 1-\eta$, for at least one of the function values encountered,

$$\|g\|_* \leq \min_{j=0:K} \varGamma(\delta^j) \leq \sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\varDelta_{\min}} + \frac{2\varDelta_{\min}}{\delta_{\max}}\Big) = \mathcal{O}(\varepsilon).$$

From (33) and Theorem 2(i), the Kth call to **FDS-basic** uses at most

$$2R + (2R+1)\frac{f_0 - \widehat{f}}{\gamma_{\min} Q^{1-K}\varDelta_{\max}}$$

function evaluations; $\widehat{f}$ comes from (2) and is the global minimum value. Then

$$N_K \leq 1+\sum_{j=1}^{K}\Big(2R+(2R+1)\frac{f_0-\widehat{f}}{\gamma_{\min}Q^{1-j}\varDelta_{\max}}\Big) = 1+2RK+(2R+1)\frac{f_0-\widehat{f}}{\gamma_{\min}\varDelta_{\max}}\frac{Q^K-1}{Q-1}.$$

Choosing $\varDelta_{\min} = \mathcal{O}(\varepsilon^2/n)$ with (34) is possible, and K, R, $\delta_{\min}$ can clearly be chosen to satisfy (35)–(37) and displays $K = \mathcal{O}(\log \dfrac{n}{\varepsilon^2})$ and $R = \mathcal{O}(\log \eta^{-1})$. Finally, we conclude that $N_K = \mathcal{O}(nR\varepsilon^{-2}) = \mathcal{O}(n\varepsilon^{-2})$. □

Theorem 4 *Let $\{x^k\}$ ($k = 0, 1, 2, \cdots$) be the sequence generated by* **VRBBO-basic** *and let f be convex on $\mathcal{L}(x^0)$. Moreover, assume that (A1) and (A2) hold and* `nfmax` *is sufficiently large. Given $0 < \eta < 1$, for any $\varepsilon > 0$, if* (34)–(37) *hold then* **VRBBO-basic** *finds after at most $\mathcal{O}(n\varepsilon^{-1})$ function evaluations with probability $\geq 1 - \eta$ a point x satisfying* (38) *and*

$$f(x) - \widehat{f} = \mathcal{O}(\varepsilon r_0), \tag{40}$$

where r_0 is given by (11) *and $\widehat{f}$ is the global minimum value.*

Proof For $\ell \geq K$, from (33)–(35), (39) holds. Hence at most K steps of **FDS-basic** are performed. In fact in each step a point without sufficient gain is found satisfying (28) by Theorem 2(ii). As discussed earlier after Theorem 1, these points are unknown to us since the gradients and Lipschitz constants are unknown. The index set of these points is denoted by U whose size is K. The convex case is characterized by (8), so that

$$\widehat{f} \geq f_\ell + (g^\ell)^T(\widehat{x} - x^\ell) \quad \text{for all } \ell \geq 0.$$

From (11), we have with probability $\geq 1 - \eta$

$$\begin{aligned} f_{\ell-1} - f_\ell \leq f_{\ell-1} - \widehat{f} &\leq (g^{\ell-1})^T(x^{\ell-1} - \widehat{x}) \leq \|g^{\ell-1}\|_* \|x^{\ell-1} - \widehat{x}\| \\ &\leq r_0\sqrt{cn}\Big(L\delta_{\min} + \sqrt{L'\Delta_\ell} + \frac{2\Delta_\ell}{\delta_{\max}}\Big) \quad \text{for } \ell \in U. \end{aligned} \tag{41}$$

We consider the following three cases:
Case 1. The second term $\sqrt{L'\Delta_\ell}$ in (41) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell \leq f_{\ell-1} - \widehat{f} = \mathcal{O}(\sqrt{n\Delta_\ell}). \tag{42}$$

Put $U_1 := \{\ell \in U \mid (42) \text{ holds}\}$.
Case 2. The first term $\delta_{\min}$ in (41) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell \leq f_{\ell-1} - \widehat{f} = \mathcal{O}(\sqrt{n}\delta_{\min}) = \mathcal{O}(\varepsilon). \tag{43}$$

Put $U_2 := \{\ell \in U \mid (43) \text{ holds}\}$.
Case 3. The third term $\dfrac{2\Delta_\ell}{\delta_{\max}}$ in (41) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell \leq f_{\ell-1} - \widehat{f} = \mathcal{O}(\sqrt{n}\Delta_\ell). \tag{44}$$

Put $U_3 := \{\ell \in U \mid (44) \text{ holds}\}$. Then we conclude from (33) and (42)–(44) that

$$\begin{aligned} \sum_{\ell \in U} \frac{f_{\ell-1} - f_\ell}{\Delta_\ell} &= \sum_{\ell \in U_1} \frac{f_{\ell-1} - f_\ell}{\Delta_\ell} + \sum_{\ell \in U_2} \frac{f_{\ell-1} - f_\ell}{\Delta_\ell} + \sum_{\ell \in U_3} \frac{f_{\ell-1} - f_\ell}{\Delta_\ell} \\ &\leq \sum_{\ell \in U_1} \frac{\mathcal{O}(\sqrt{n\Delta_\ell})}{\Delta_\ell} + \mathcal{O}(\varepsilon) \sum_{\ell \in U_2} \Delta_\ell^{-1} + \sum_{\ell \in U_3} \frac{\mathcal{O}(\sqrt{n}\Delta_\ell)}{\Delta_\ell} \end{aligned}$$

$$\le \sum_{\ell\in U}\mathcal{O}(\sqrt{n}\Delta_\ell^{-1/2}) + \mathcal{O}(\varepsilon)\sum_{\ell\in U}\Delta_\ell^{-1} + \sum_{\ell\in U}\frac{\mathcal{O}(\sqrt{n}\Delta_\ell)}{\Delta_\ell}$$
$$\le \Delta_{\max}^{-1/2}\sum_{\ell\in U}\mathcal{O}(\sqrt{n}Q^{\frac{1}{2}(\ell-1)}) + \Delta_{\max}^{-1}\mathcal{O}(\varepsilon)\sum_{\ell\in U}Q^{\ell-1} + \mathcal{O}(\sqrt{n})K.$$

Hence (34) and (35) result in

$$\sum_{\ell\in U}\frac{f_{\ell-1}-f_\ell}{\Delta_\ell} = \mathcal{O}(n\varepsilon^{-1}) + \mathcal{O}(n\varepsilon^{-1}) + \mathcal{O}(\sqrt{n}\log(n\varepsilon^{-1})) = \mathcal{O}(n\varepsilon^{-1}),$$

so that we get in the same way as the proof of Theorem 2

$$N_K \le 1 + 2RK + (2R+1)\sum_{\ell\in U}\frac{f_{\ell-1}-f_\ell}{\gamma_{\min}\Delta_\ell} = \mathcal{O}(nR\varepsilon^{-1}) = \mathcal{O}(n\varepsilon^{-1}).$$

According Theorem 3, (38) holds at least for one of evaluated points. As a result, at least for one of evaluated points (40) holds with probability $\ge 1-\eta$ by applying (11) and (38) to (41). □

4.3 The strongly convex case

Theorem 5 *Let $\{x^k\}$ ($k = 0, 1, 2, \cdots$) be the sequence generated by* **VRBBO-basic** *and let f be σ-strongly convex on $\mathcal{L}(x^0)$. Moreover, assume that (A1) and (A2) hold and* `nfmax` *is sufficiently large. Under the assumptions of Theorem 3,* **VRBBO-basic** *finds after at most*

$$\mathcal{O}(n\log n\varepsilon^{-1})$$

function evaluations with probability $\ge 1-\eta$ a point x satisfying (38)*,*

$$f(x) - \widehat{f} = \mathcal{O}\Big(\frac{\varepsilon^2}{2\sigma}\Big), \tag{45}$$

and

$$\|x-\widehat{x}\| = \mathcal{O}\Big(\frac{\varepsilon}{\sigma^2}\Big). \tag{46}$$

Proof For $\ell \ge K$, from (33)–(35), (39) holds. Hence at most K steps of **FDS-basic** are performed. In fact in each step a point without sufficient gain is found satisfying (28) by Theorem 2(ii). As discussed earlier after Theorem 1, these points are unknown to us since the gradients and Lipschitz constants are unknown. The index set of these points is denoted by U whose size is K. The strongly convex case is characterized by (9), so that f has a global 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 (1), 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

$$\widehat{f} \ge f(x) - \frac{1}{2\sigma}\|g(x)\|_*^2. \tag{47}$$

The replacement of x by $x^{\ell-1}$ in (47) and (38) gives, with probability $\ge 1-\eta$,

$$f_{\ell-1} - f_\ell \le f_\ell - \widehat{f} \le \frac{\|g^{\ell-1}\|_*^2}{2\sigma} \le \frac{cn}{2\sigma}\Big(L\delta_{\min} + \sqrt{L'\mathit{\Delta}_\ell} + \frac{2\mathit{\Delta}_\ell}{\delta_{\max}}\Big)^2. \tag{48}$$

We consider the following three cases:
Case 1. The second term $\sqrt{L'\mathit{\Delta}_\ell}$ in (48) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell = \mathcal{O}(n\mathit{\Delta}_\ell). \tag{49}$$

Put $U_1 := \{\ell \in U \mid (49) \text{ holds}\}$.
Case 2. The first term $\delta_{\min}$ in (48) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell = \mathcal{O}(n\delta_{\min}^2) = \mathcal{O}(\varepsilon^2). \tag{50}$$

Put $U_2 := \{\ell \in U \mid (50) \text{ holds}\}$.
Case 3. The third term $\dfrac{2\mathit{\Delta}_\ell}{\delta_{\max}}$ in (48) dominates the others. Then for $\ell \in U$

$$f_{\ell-1} - f_\ell = \mathcal{O}(n\mathit{\Delta}_\ell^2). \tag{51}$$

Put $U_3 := \{\ell \in U \mid (51) \text{ holds}\}$. Then we conclude from (33) and (49)–(51) that

$$\begin{aligned}
\sum_{\ell\in U}\frac{f_{\ell-1}-f_\ell}{\mathit{\Delta}_\ell} &= \sum_{\ell\in U_1}\frac{f_{\ell-1}-f_\ell}{\mathit{\Delta}_\ell} + \sum_{\ell\in U_2}\frac{f_{\ell-1}-f_\ell}{\mathit{\Delta}_\ell} + \sum_{\ell\in U_3}\frac{f_{\ell-1}-f_\ell}{\mathit{\Delta}_\ell} \\
&\le \sum_{\ell\in U_1}\frac{\mathcal{O}(n\mathit{\Delta}_\ell)}{\mathit{\Delta}_\ell} + \mathcal{O}(\varepsilon^2)\sum_{\ell\in U_2}\mathit{\Delta}_\ell^{-1} + \sum_{\ell\in U_3}\frac{\mathcal{O}(n\mathit{\Delta}_\ell^2)}{\mathit{\Delta}_\ell} \\
&\le \sum_{\ell\in U}\frac{\mathcal{O}(n\mathit{\Delta}_\ell)}{\mathit{\Delta}_\ell} + \mathcal{O}(\varepsilon^2)\sum_{\ell\in U}\mathit{\Delta}_\ell^{-1} + \sum_{\ell\in U}\frac{\mathcal{O}(n\mathit{\Delta}_\ell^2)}{\mathit{\Delta}_\ell} \\
&\le \mathcal{O}(n)K + \frac{\mathcal{O}(\varepsilon^2)}{\mathit{\Delta}_{\max}}\sum_{\ell\in U}Q^{\ell-1} + \mathcal{O}(n)\mathit{\Delta}_{\max}\sum_{\ell=1}^{\infty}Q^{1-\ell}.
\end{aligned}$$

Hence (34) and (35) result in

$$\sum_{\ell\in U}\frac{f_{\ell-1}-f_\ell}{\mathit{\Delta}_\ell} = \mathcal{O}(n\log n\varepsilon^{-1}) + \mathcal{O}(\varepsilon^2)\mathcal{O}(n\varepsilon^{-2}) + \mathcal{O}(n) = \mathcal{O}(n\log n\varepsilon^{-1}),$$

and we then obtain in the same way as the proof of Theorem 2

$$N_K \le 1 + 2RK + (2R+1)\sum_{\ell\in U}\frac{f_{\ell-1}-f_\ell}{\gamma_{\min}\mathit{\Delta}_\ell} = \mathcal{O}(nR\log n\varepsilon^{-1}) = \mathcal{O}(n\log n\varepsilon^{-1}).$$

According to Theorem 3, (38) holds at least for one of evaluated points. As a result, at least for one of evaluated points (45) holds with probability $\geq 1-\eta$ by applying (38) to (45). (46) is obtained by applying (4), (45), and since $g(\widehat{x})=0$, i.e.,

$$\|x-\widehat{x}\|^2 \leq \frac{2}{\sigma}\Big(f(x)-\widehat{f}-g(\widehat{x})^T(x-\widehat{x})\Big)=\mathcal{O}\Big(\frac{\varepsilon}{\sigma^2}\Big).$$

□

5 Some new heuristic techniques

In this section, we describe several heuristic techniques that improve the basic version of Algorithm 4, **VRBBO**. While only convergence to a local minimizer is guaranteed, the **FDS** together with our heuristic techniques turn **VRBBO** into an efficient global solver. In fact **FDS** takes initially only for large Δ many large steps, hence has a global character.

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 how to approximate the gradient estimated by finite differences with step sizes extracted from the extrapolation steps. In Section 6, we combine Algorithm 4 with these heuristic techniques, resulting in the global solver **VRBBO**.

5.1 Cumulative directions

We consider two possibilities to accumulate past directional information into a cumulative search direction:
(i) With `xm` and `fm` defined in Section 2.3.1 the first cumulative direction is model independent, computed by

$$p=\texttt{xm}-x_{\text{init}}, \tag{52}$$

where x_{init} is the initial point of the current improved version of **MLS-basic**. 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

$$f\Big(\texttt{xm}+\sum_{i\in I}\alpha_i p_i\Big)\approx \texttt{fm}-\sum_{i\in I}\Psi_i(\alpha_i) \tag{53}$$

with quadratic univariate functions $\Psi_i(\alpha)$ 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 **MLS-basic**.

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

$$f(\mathtt{xm} + \alpha p) = \mathtt{fm} - \frac{\alpha}{2} d + \frac{\alpha^2}{2} h = \mathtt{fm} - \Psi(\alpha),$$

where $\Psi(\alpha) := \frac{\alpha}{2}(d - \alpha h)$. Let us recall the function values `fl`, `fm`, and `fr` satisfying either (19) or (22). If $\mathtt{fr} < \mathtt{fm}$, the last evaluated point was the best one, so

$$\mathtt{fr} \le \min(\mathtt{fl}, \mathtt{fm}).$$

In this case, (22) holds and we have

$$d := 4\mathtt{fm} - 3\mathtt{fr} - \mathtt{fl}, \tag{54}$$

and

$$h := \mathtt{fr} + \mathtt{fl} - 2\mathtt{fm}. \tag{55}$$

Otherwise, the last evaluated point was not the best one, so $\mathtt{fm} \le \min(\mathtt{fl}, \mathtt{fr})$. In this case, (19) holds and we compute d by

$$d := \mathtt{fl} - \mathtt{fr} \tag{56}$$

and h by (55).

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

$$\alpha := \begin{cases} a & \text{if } d \ge 0, \\ -a & \text{if } d < 0 \end{cases} \tag{57}$$

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

$$\alpha := \begin{cases} \min(a, d/2h) & \text{if } d \ge 0, \\ \max(-a, d/2h) & \text{if } d < 0. \end{cases} \tag{58}$$

Assuming the validity of the quadratic model (53), we find the model optimizer by additively accumulating the estimated steps αp and gains Ψ into a cumulative step q with anticipated gain r.

5.2 Random subspace direction

When sufficient gains are found, the trial points are accepted as the new best points and saved in X and their function values are saved in F. Denote by m_s the maximum number of points saved in X and by b the index of newest best point.

Throughout the paper, $A_{:k}$ denotes the kth column of a matrix A. 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

$$\alpha_{\text{rand}} := \text{rand}(m_s - 1, 1) - 0.5, \quad \alpha_{\text{rand}} := \frac{\alpha_{\text{rand}}}{\|\alpha_{\text{rand}}\|}, \quad p := \sum_{i=1, i \ne b}^{m_s} (\alpha_{\text{rand}})_i (X_{:i} - X_{:b}). \tag{59}$$

5.3 Choosing the initial $\varDelta$

First of all, we compute

$$\texttt{dF} := \operatorname*{median}_{i=1:m_s} |F_i - F_b|. \tag{60}$$

Then if dF is not zero we approximate the initial desired gain

$$\varDelta := \gamma_{\max} \min(\texttt{dF}, 1), \tag{61}$$

where $\gamma_{\max} > 0$ is a tuning parameter. Otherwise $\varDelta := \varDelta_{\max}$, where $\varDelta_{\max} > 0$ is the initial gain.

5.4 Choosing the initial λ

The initial value for λ is $\lambda_{\max}$ which is the tuning parameter, however, it is updated by (20) provided that the best point is updated by **extrapolationStep**. Our achievement is to approximate 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 compute λ by

$$\lambda := \begin{cases} \dfrac{\gamma_\lambda}{\sqrt{n}} & \text{if } \texttt{dF} = 0 \text{ and } \lambda_{\text{old}} = 0, \\ \lambda_{\text{old}} & \text{if } \texttt{dF} = 0 \text{ and } \lambda_{\text{old}} \neq 0, \\ \gamma_\lambda \sqrt{\dfrac{\texttt{dF}}{n}} & \text{otherwise,} \end{cases} \tag{62}$$

where dF is computed by (60).

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 (17). We compute

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

and estimate the scaling vector

$$s := \sup_{i=1:m_s} (\texttt{dX}_{:i}), \quad J = \{i \mid s_i = 0\}, \quad s_J = 1. \tag{63}$$

Finally, the formula (17) is rewritten as

$$p := s \circ p \quad \text{and} \quad p = p(\delta/\|p\|), \tag{64}$$

where $\circ$ denotes componentwise multiplication and δ is computed by (23).

5.6 Estimating the gradient

With `xm` and `fm` defined in Section 2.3.1, finite difference quasi-Newton methods approximate the gradient with components

$$\widetilde{g}_i := \frac{f(\texttt{xm} + \alpha_i e_i) - \texttt{fm}}{\alpha_i},$$

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

$$\alpha_i := \max\{1, \|\texttt{xm}\|_\infty\}\sqrt{\varepsilon_m}, \tag{65}$$

where ε_m is the machine precision; another choice for α is made now. After generating each coordinate search direction, we approximate each component of the 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 (65). The reason of this change is that we don't need to approximate the gradient by another algorithm due to the additional cost. Let describe how to compute the gradient. If **extrapolationStep** cannot find a sufficient gain in the tth iteration ($n_{\rm sg} = 0$), `fr` is computed and $\mathbf{a}(t)$ is unchanged. Given the old best point $\texttt{fm}_{\rm old}$, the tth component of the gradient is computed by

$$\widetilde{g}_t := (\texttt{fr} - \texttt{fm}_{\rm old})/\mathbf{a}(t); \tag{66}$$

otherwise, it is computed by

$$\widetilde{g}_t := (\texttt{fm} - \texttt{fm}_{\rm old})/\mathbf{a}(t), \tag{67}$$

where both $\texttt{fm}_{\rm old}$ and $\mathbf{a}(t)$ are updated by **extrapolationStep**. We will add later this computation to an improved version of **MLS-basic**.

6 The implemented version of VRBBO-basic

In this section, we discuss the implementation of **VRBBO-basic** with the improvements which are of a heuristic nature, very important for efficiency, and do not change the order of our complexity results. Thus **VRBBO** 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, **VRBBO** matches the quality of all state-of-the-art algorithms for unconstrained black box optimization problems. **VRBBO** is implemented in Matlab; the source code is obtainable from

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

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**, **FDS**, and **setScales**, which are described below.

Before we compute the direction, the type of direction needs to be identified by **identifyDir**. **VRBBO** 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 approximate the gradient by the finite difference approach.
• Limited memory quasi-Newton directions are computed by **lbfgsDir** (standard limited memory BFGS direction [36]). Due to rounding errors, the computed direction may not satisfy the angle condition (15); hence it needs to be modified by **enforceAngle** discussed in [39].
• **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.

These subalgorithms use `cum` (the cumulative step type), $\mathtt{ms}_{\max}$ (the maximum number of best points kept), $\mathtt{mq}_{\max}$ (the memory for L-BFGS approximation), $0 < \gamma_w < 1$ and $0 < \gamma_{\rm a} < 1$ (tiny parameters for the angle condition), `scSub` (random subspace direction scale?), and `scCum` (cumulative direction scale?) as the tuning parameters.

We denote by C the number of coordinate directions, by R the number of random directions, and by S the number of subspace directions in each repeated call to a multi-line search algorithm – an improved version of **MLS-basic**, called **MLS**; 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.

Denote by **FDS** the improved version of **FDS-basic**. Both **setScales** and **FDS** work by making repeated calls to **MLS**. **MLS** polls in several suitably chosen directions (implemented 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 $\varDelta$. Schematically, it works as follows:
• At first, at most C iterations with coordinate directions are used. They are used to approximate 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 finding a sufficient gain, a cumulative direction is used as final, Tth direction in the hope of finding a model-based gain.

MLS-basic 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.

VRBBO initially calls the algorithm **setScales** to estimate a good scaling of norms, step lengths, and related control parameters. Then it uses in each iteration **FDS**, aimed at repeatedly reducing the function value by an amount of at least a multiple of $\varDelta$ to update the best point. If no sufficient gain is found in a call to **FDS**, $\varDelta$ is reduced by a factor of Q. Once $\varDelta$ is below a minimum threshold or `nfmax` is reached, **VRBBO** stops.

An important question is the ordering of the search directions. In Section 7, it will be shown that after coordinate directions are used the use of subspace directions (limited memory quasi Newton and random subspace directions) is very preferable. Changing the ordering of other directions is not very effective on the efficiency of our algorithm.

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 $\geq 1 - 2^{2+C+S-T} = 1 - 2^{-R}$, where 5 kinds of directions are used. Given the tuning parameter `alg` $\in \{0, 1, 2, 3, 4, 5\}$ (algorithm type), we now discuss the factor of bounds depending on the number of search directions used in **MLS-basic**. We would have the following cases:

- In the first case (`alg` $= 0$), $T = R < n$ random directions are used. Then complexity results considered as Table 3 are valid. This variant of **VRBBO** is denoted by **VRBBO-basic1**.
- In the second case (`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 **VRBBO** is denoted by **VRBBO-basic2**.
- In the third case (`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 **VRBBO** is denoted by **VRBBO-C-Q**, ignoring the coordinate and limited memory quasi Newton directions.
- In the fourth case (`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 **VRBBO** is denoted by **VRBBO-Q**, ignoring the limited memory quasi Newton directions.
- In the fifth case (`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 **VRBBO** is denoted by **VRBBO-S**.
- In the sixth case (`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 **VRBBO** is the default version.

This defines six versions of **VRBBO**, 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 **VRBBO** are necessary for the 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 **VRBBO**

For our tests we used the following parameter choices:

Common tuning parameters: $E = 10$; $\delta_{\min} = 10^{-4}\sqrt{n}$; $\delta_{\max} = 0.1\sqrt{n}$; $\varDelta_{\min} = 0$; $\varDelta_{\max} = 10^{-3}$; $\gamma_\delta = 10^6$; $\gamma_e = 4$; $Q = 2$; $\lambda_{\max} = 1$; $\alpha_{\min} = 10^{-50}$; $\gamma_{\min} = 10^{-6}$;
VRBBO-basic1: $R = \mathtt{fix}(n/2) + 1$ and $\mathtt{alg} = 0$;
VRBBO-basic2: $R = n$ and $\mathtt{alg} = 1$;
VRBBO-C-Q: $\mathtt{ms}_{\max} = 5$; $R = \mathtt{fix}(n/2) + 1$; $S = \mathtt{fix}(n/5)$; $T_0 = 2 * \mathtt{ms}_{\max}$; $\mathtt{scCum} = 0$; $\mathtt{scSub} = 0$; $\gamma_{\max} = 10^{-3}$; $\mathtt{cum} = 1$; $\mathtt{alg} = 2$;
VRBBO-S: $C = n$; $R = \min(\mathtt{fix}(n/10) + 1, 20)$; $\mathtt{scCor} = 0$; $\mathtt{cum} = 1$; $\gamma_w = \varepsilon_m$; $\gamma_{\mathrm{a}} = 10^{-20}$; $\mathtt{mq}_{\max} = 5$; $\mathtt{alg} = 3$;
VRBBO-Q: $\mathtt{ms}_{\max} = 5$; $C = n$; $R = \min(\mathtt{fix}(n/10) + 1, 20)$; $S = \min(\mathtt{fix}(n/10), 5)$; $\mathtt{scCor} = 0$; $\mathtt{scSub} = 0$; $\mathtt{cum} = 1$; $\mathtt{ms}_{\max} = 5$; $T_0 = 2 * \mathtt{ms}_{\max}$; $\gamma_{\max} = 10^{-3}$; $\mathtt{alg} = 4$;
VRBBO: $\mathtt{ms}_{\max} = 5$; $\mathtt{mq}_{\max} = 5$; $T_0 = 2 * \mathtt{ms}_{\max}$; $C = n$; $S = \min(\mathtt{fix}(n/10) + 1, 5)$; $R = \min(\mathtt{fix}(n/10) + 1, 20)$; $\mathtt{scCum} = 0$; $\mathtt{scCor} = 0$; $\mathtt{scSub} = 0$; $\mathtt{cum} = 1$; $\gamma_{\max} = 10^{-3}$; $\gamma_w = \varepsilon_m$; $\gamma_{\mathrm{a}} = 10^{-20}$; $\mathtt{alg} = 5$;

Table 4: The values of the tuning parameters

Although the best theoretical complexity is obtained for

$$\varOmega(\log \eta^{-1}) \quad \text{for a given } 0 < \eta \le \tfrac{1}{2},$$

the best numerical result are obtained for much larger $R \sim n$.

$\varDelta_{\min} = 0$ implies that the algorithm stops due to `nfmax` or `secmax`. Here `secmax` is maximal time in seconds.

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., [7,52,53]. In Table 4, there are 7 integral, 2 binary, 2 ternary, and 14 continuous tuning parameters, giving a total of 25 parameters for tuning our algorithm. A small amount of tuning was done by hand. Automatic tuning of **VRBBO** will be considered elsewhere.

7.2 Test problems used

We compare 29 competitive solvers from the literature (discussed in Subsection 7.3) on all 549 unconstrained problems from the `CUTEst` [26] collection of test problems for optimization and the test problems by JAMIL & YANG [37] for global optimization with up to 5000 variables, in case of variable dimension problems for all allowed dimensions in this range. For problems in dimension $n \geq 21$, 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 $i = 1, \dots, n$.

As discussed earlier, `nfmax` denotes maximal number of function evaluations and `secmax` denotes maximal time in seconds. `nf` and `msec` denote the number of function evaluations and the time in seconds, respectively. We limited the budget available for each solver by allowing at most

$$\texttt{secmax} := \begin{cases} 360 & \text{if } 2 \leq n \leq 20, \\ 420 & \text{if } 21 \leq n \leq 100 \\ 720 & \text{if } 101 \leq n \leq 1000 \\ 1800 & \text{if } 1001 \leq n \leq 5000 \end{cases}$$

seconds of run time and at most

$$\texttt{nfmax} := \begin{cases} 100n, 500n, 1000n & \text{if } 2 \leq n \leq 20, \\ 100n, 500n, 1000n & \text{if } 21 \leq n \leq 100 \\ 100n, 500n & \text{if } 101 \leq n \leq 1000 \\ 100n, 500n & \text{if } 101 \leq n \leq 1000 \end{cases}$$

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 runs to ensure that the best solvers can solve most of the test problems. Both `nfmax` and `secmax` are input parameters for all solvers.

A problem with dimension n is considered solved by the solver *so* if the target accuracy satisfies

$$q_{so} := (f_{so} - f_{\text{opt}})/(f_{\text{init}} - f_{\text{opt}}) \leq \begin{cases} 10^{-4} & \text{if } 1 \leq n \leq 100, \\ 10^{-3} & \text{if } 101 \leq n \leq 5000, \end{cases}$$

where f_{init} is the function value of the starting point (common to all solvers), f_{opt} is the best point known to us, and f_{so} is the best point found by the solver *so*.

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 through 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. [13], **ASACG** presented by Hagar & Zhang [29] and **LMBOPT** implemented by Kimiaei et al. [39]. The condition $\|g^k\|_\infty \le 10^{-5}$ was satisfied for all test problems except those listed in Subsection 9.2.

For a more refined statistics, we use our test environment (Kimiaei & Neumaier [44]) for comparing optimization routines on the `CUTEst` test problem collection by Gould et al. [26]. A solver is said **efficient** when it has lowest relative cost of function evaluations and said **robust** when it has highest number of solved problems. Performance profiles Dolan & Moré [22] and data profiles by Moré & Wild [45] for the two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in seconds) are displayed to identify which solvers are competitive (efficient and robust) for small up to high dimensions. In fact, the efficiency and robustness of all solvers are identified by the performance/data profiles.

7.3 Codes compared

We compare **VRBBO** 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.

• **SNOBFIT**, obtained from

`http://www.mat.univie.ac.at/~neum/software/snobfit/snobfit_v2.1.tar.gz`

is a combination of a branching strategy to enhance the chance of finding a global minimum with a sequential quadratic programming method based on fitted quadratic models to have good local properties by Huyer & Neumaier [35].

• **NOMAD** (version 3.9.1) , obtained from

`https://www.gerad.ca/nomad`

is a Mesh Adaptive Direct Search algorithm (MADS) [1,2,5,6,42]. **NOMAD1** uses the following option set

`opts` = `nomadset`('`max_eval`',`nfmax`,'max_iterations',2*`nfmax`,
'min_mesh_size','1e-008','initial_mesh_size','10')

while **NOMAD2** uses the following option set

`opts` = `nomadset`('`max_eval`',`nfmax`,'max_iterations',2*`nfmax`,
'min_mesh_size','1e-008','initial_mesh_size','10','model_search','0').

• **UOBYQA**, **NEWUOA**, and **BOBYQA** obtained from https://www.pdfo.net/docs.html

are model-based solvers by Powell [54,55,56].

• **STP-fs, STP-vs**, and **PSTP**, obtained from the authors of Bergou et al. [11], 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 [52].

• **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 [8]. We used **CMAES** with the tuning parameters

$\texttt{oCMAES.MaxFunEvals} = \texttt{nfmax}$, $\texttt{oCMAES.DispFinal} = 0$, $\texttt{oCMAES.DispModulo} = 0$,
$\texttt{oCMAES.LogModulo} = 0$, $\texttt{oCMAES.SaveVariables} = 0$, $\texttt{oCMAES.MaxIter} = \texttt{nfmax}$,
$\texttt{oCMAES.Restarts} = 7$.

• **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. [16]. We used **GLOBAL** with the tuning parameters

$\texttt{oGLOBAL.MAXFNALL} = \texttt{nfmax}$, $\texttt{oGLOBAL.MAXFN} = \texttt{nfmax}/5$, $\texttt{oGLOBAL.DISPLAY} =$‘$\texttt{off}$’,
$\texttt{oGLOBAL.N100} = 300$, $\texttt{oGLOBAL.METHOD} =$‘$\texttt{unirandi}$’, and $\texttt{oGLOBAL.NG0} = 2$.

• **DE**, obtained from

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

is the stochastic differential evolution algorithm by Storn & Price [59].

• **MCS**, obtained from

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

is the deterministic global optimization by multilevel coordinate search by Huyer & Neumaier [34]. We used **MCS** with the tuning parameters

$\texttt{iinit} = 1$, $\texttt{nfMCS} = \texttt{nfmax}$, $\texttt{smax} = 5n + 10$, $\texttt{stop} = 3n$, $\texttt{local} = 50$, $\texttt{gamma} = \texttt{eps}$,
$\texttt{hess} = \texttt{ones}(n, n)$, and $\texttt{prt} = 0$.

• **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. [28].

• **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 [19, 18].

• **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. [40]. 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. [20, 21]. 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 [61].

• **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 [38]. 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 [32].

• **GLODS**, obtained from

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

is Global and Local Optimization using Direct Search by CUSTÓDIO & MADEIRA [17].

• **ACRS**, obtained from

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

is a global optimization algorithm presented by BRACHETTI et al. [12].

• **SDBOX**, obtained from

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

is a derivative-free algorithm for bound constrained optimization problems by LUCIDI & SCIANDRONE [43].

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

VRBBO 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).

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

We have a self-testing and tuning for our solver in terms of the tuning parameter `alg` shown in Figure 2. As can be seen from the data and performance profiles, two competitive versions of our solver are **VRBBO** and **VRBBO-Q**. In fact, **VRBBO** is somewhat more robust than **VRBBO-Q** while **VRBBO-Q** is somewhat more efficient than **VRBBO**. As a result, **VRBBO** is recommended.

Results on `CUTEst`. Subfigures of Figure 3 display three comparisons among all solvers in low to high budgets (`nfmax` $\in \{100n, 500n, 1000n\}$) on `CUTEst`. The name of solvers is given in the horizontal axis of these subfigures, sorted by decreasing the number of solved problems in decreasing order. As can be seen from these subfigures, in low to high budgets **UOBYQA** is more robust and efficient than the others. By increasing the budget, the efficiency and robustness of **VRBBO** are increased, so that it is the fourth rank robust solver with low function evaluations cost in comparison with the other line search, direction search, Nelder–Mead solvers, etc., except in comparison with a few model based solvers.

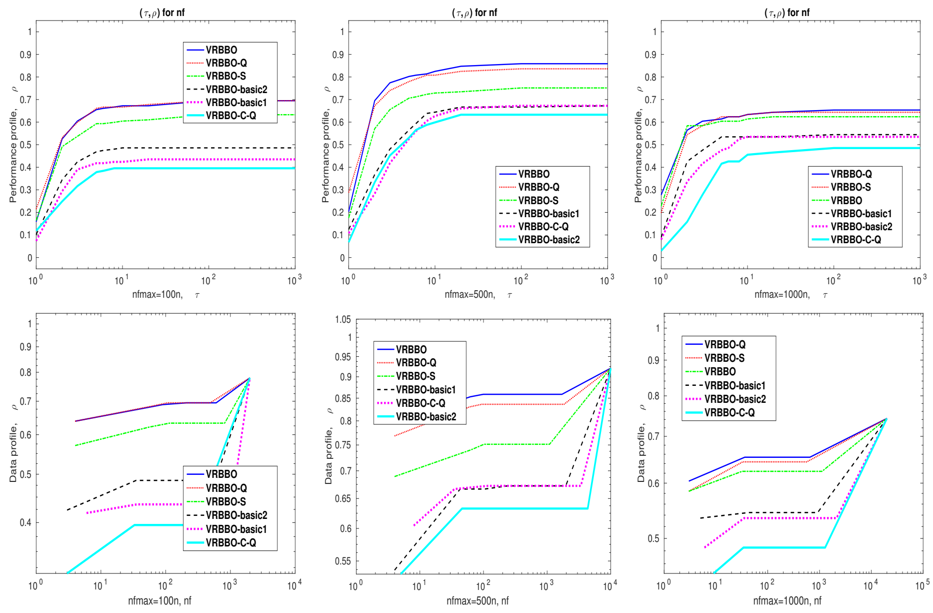
To have more accurate comparisons among the six most robust or most efficient solvers, the data and performance profiles are displayed in Figure 4 whose subfigures confirm that **UOBYQA** is more robust and efficient than the others in low to up budgets.

In summary, for small scale problems from the `CUTEst` collection, the model-based solvers are recommended and **VRBBO** is recommended in high budget since it is somewhat more robust than a few known model-based solvers (**NEWUOA**, **BCDFO**, and **BOBYQA**) although these solvers are somewhat more efficient than **VRBBO**.

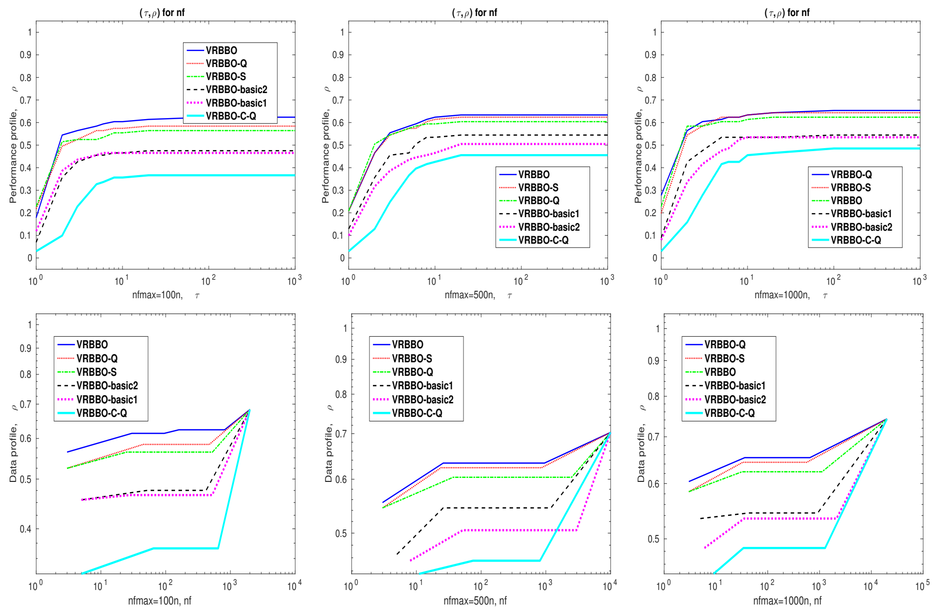
Results on `GlobalTest`. Subfigures of Figure 3 display three comparisons among all solvers in low to high budgets (`nfmax` $\in \{100n, 500n, 1000n\}$) on `GlobalTest`. The name of solvers is given in the horizontal axis of these subfigures, sorted by decreasing the number of solved problems in decreasing order. As can be seen from these subfigures:
• In low budget **BOBYQA** and **VRBBO** are the first and second rank robust solvers while **BOBYQA** and **NEWUOA** are the first and second rank efficient solvers.
• In medium and large budgets **MCS** and **GLOUS** are the first and second rank robust. In fact, by increasing the budget the global solvers have better behavior. In this case, **VRBBO** and **VRBBO-Q** are comparable with the global solvers.

To have more accurate comparisons among the four most robust or most efficient solvers, the data and performance profiles are displayed in Figure 4 whose subfigures confirm that **MCS** and **GLOUS** are more robust and efficient than the others in medium and large budgets, respectively.

In summary, the global solvers are more robust than local solvers on the `GlobalTest` collection while our findings show that **VRBBO** is comparable with the global solvers in terms of the efficiency and the robustness and is recommended for finding the global minimum.

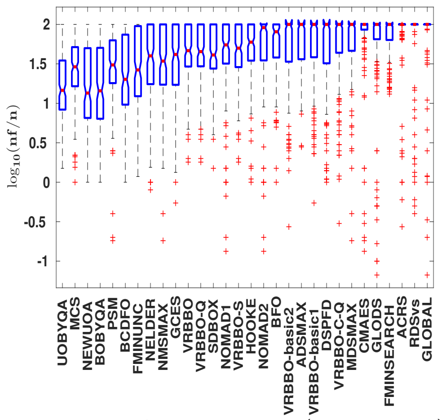


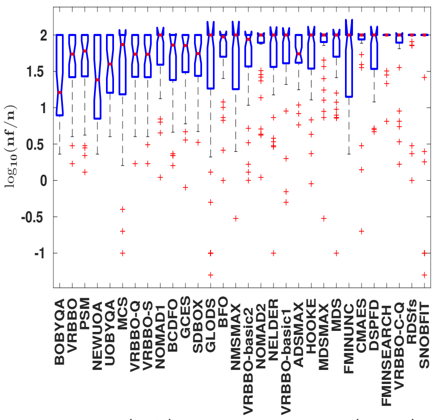
Results for small dimensions 2–20 on `CUTEst`



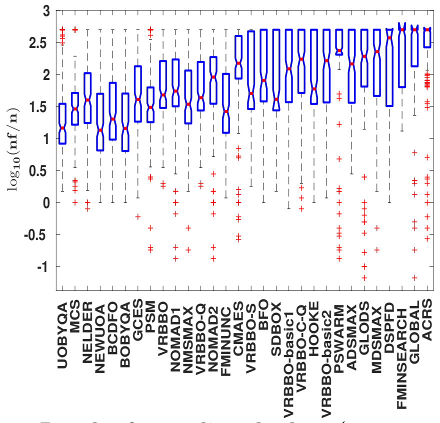
Results for small dimensions 2–20 on `GlobalTest`

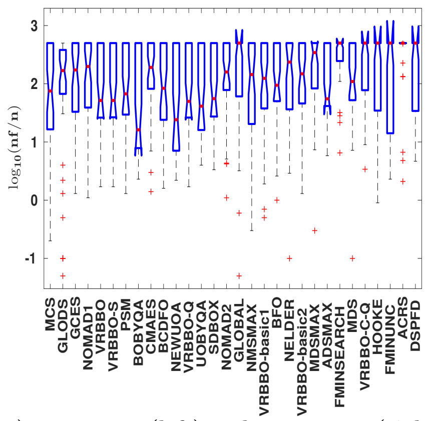
Fig. 2: For performance profiles ρ denotes the fraction of problems solved within a factor τ of the best solver. For data profiles ρ denotes the fraction of problems solved within the number of function evaluations and time in seconds used by the best solver. Problems solved by no solver are ignored.



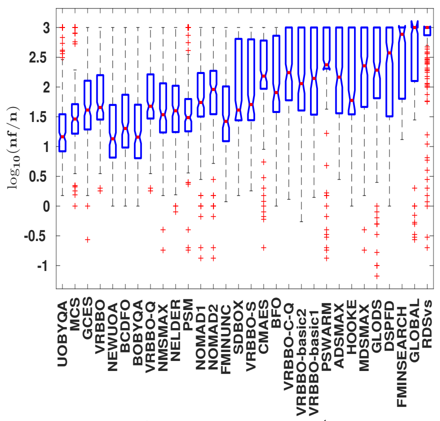


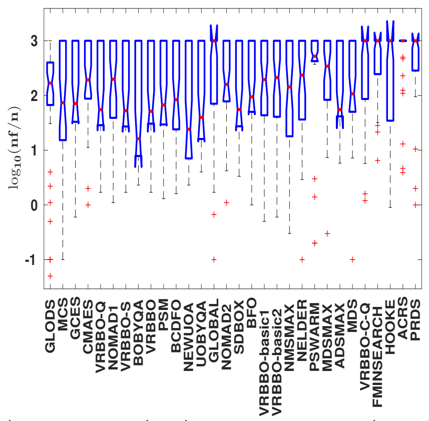
Results for low budget ($\texttt{nfmax} = 100n$) **on:** CUTEst **(left) and** GlobalTest **(right)**





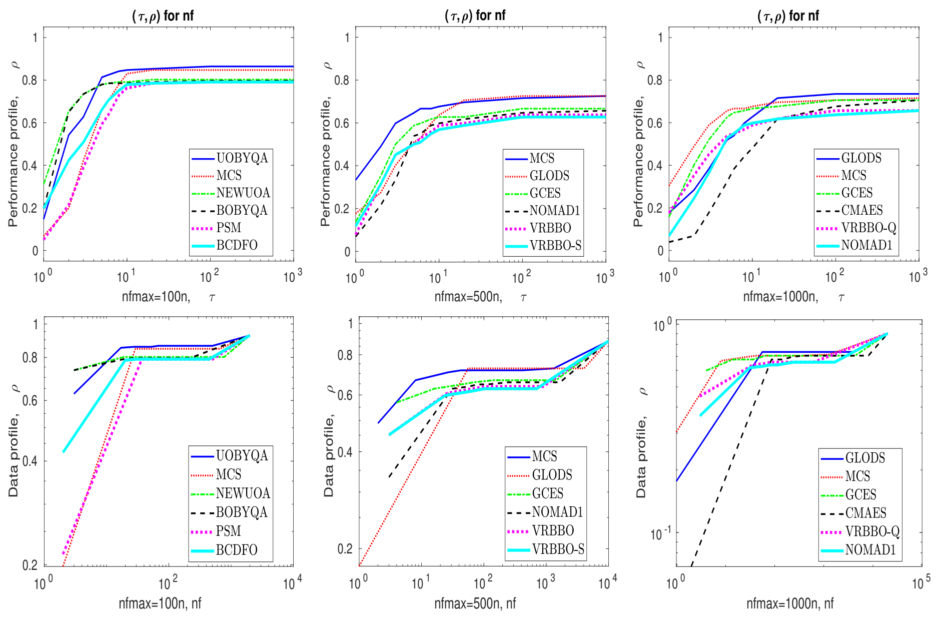
Results for medium budget ($\texttt{nfmax} = 500n$) **on:** CUTEst **(left) and** GlobalTest **(right)**



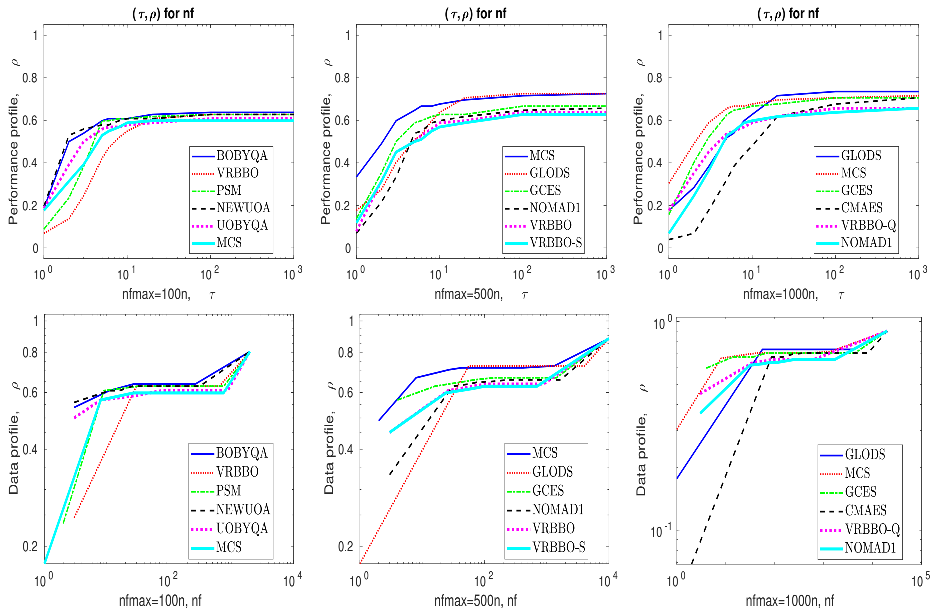


Results for large budget ($\texttt{nfmax} = 1000n$) **on:** CUTEst **(left) and** GlobalTest **(right)**

Fig. 3: Boxplots for small dimensions 2–20 in terms of `nf`.



Results for small dimensions 2–20 on `CUTEst`



Results for small dimensions 2–20 on `GlobalTest`

Fig. 4: Details as in Figure 2.

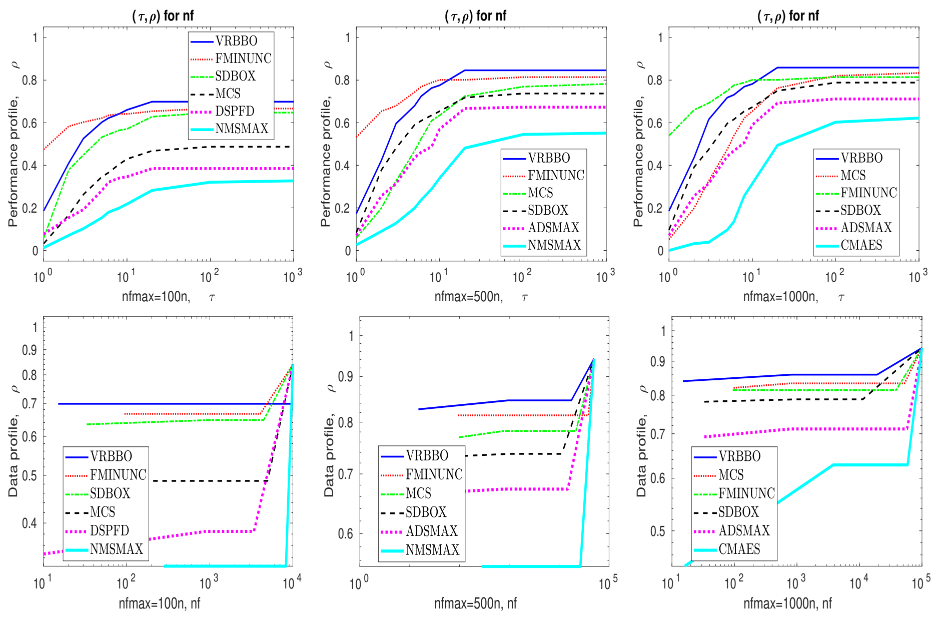
7.5 Results for medium dimensions ($21 \leq n \leq 100$)

For medium to very large scale problems, we ignored the model-based solvers in our comparison, because they needed $\frac{1}{2}n(n+3)$ sample points to construct fully quadratic models. There were a few too slow solvers even if `secmax` was expanded they could not solve most of the problems. Hence we compared only either fast, efficient or robust solvers and plotted the data and performance profiles for most robust solvers on `CUTEst` and `GlobalTest` for medium up to very large problems.

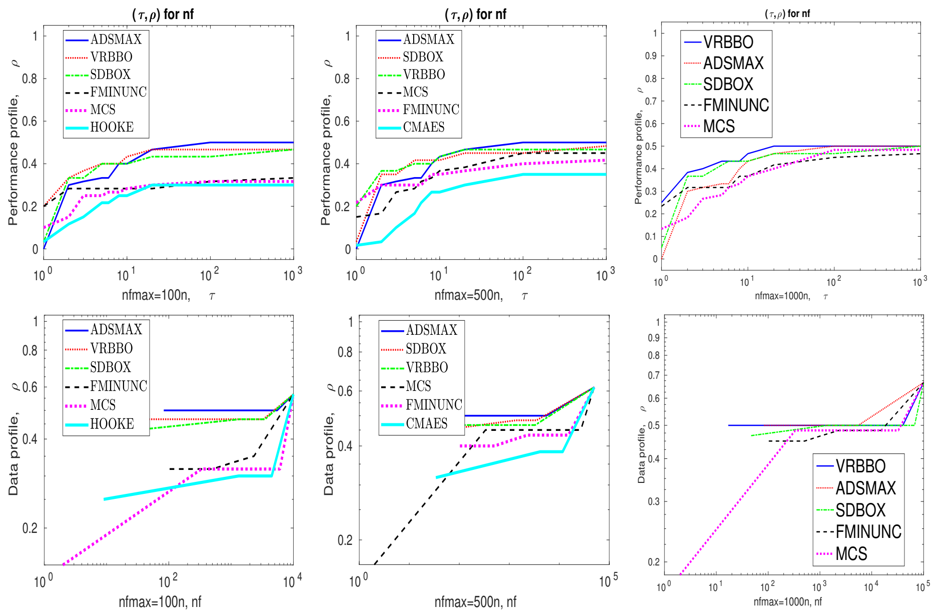
Results on `CUTEst`. We conclude from the performance profiles and the data profiles shown in Figure 5 that **VRBBO** and **VRBBO-Q** are much more efficient and robust than other solvers.

Results on `GlobalTest`. We conclude from the performance profiles and the data profiles shown in Figure 5 that:
• In low budget **ADSMAX** and **VRBBO** are the first and second rank robust solvers while **VRBBO** is much more efficient than **ADSMAX**.
• In medium budget **ADSMAX**, **SDBOX**, and **VRBBO** are, respectively, more robust than the others.
• In large budget, **VRBBO** is much more efficient and robust than others.

In summary, although **VRBBO** is comparable with the global solvers for small scale problems from `GlobalTest` it is much more efficient and robust than them for medium scale problems from `GlobalTest`.



Results for medium dimensions 21–100 on `CUTEst`



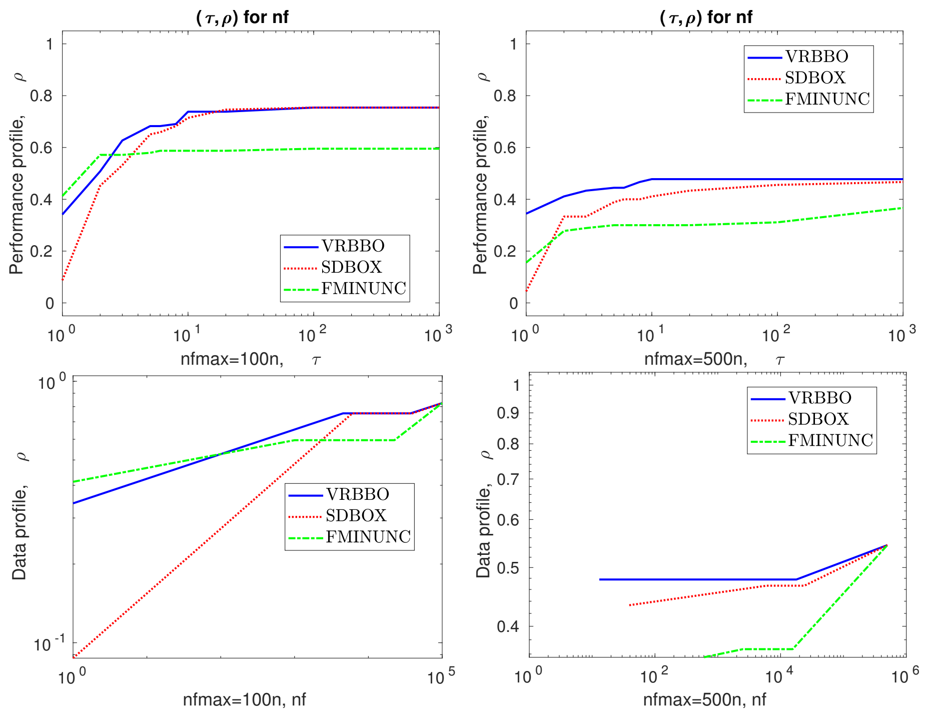
Results for medium dimensions 21–100 on `GlobalTest`

Fig. 5: Details as in Figure 2.

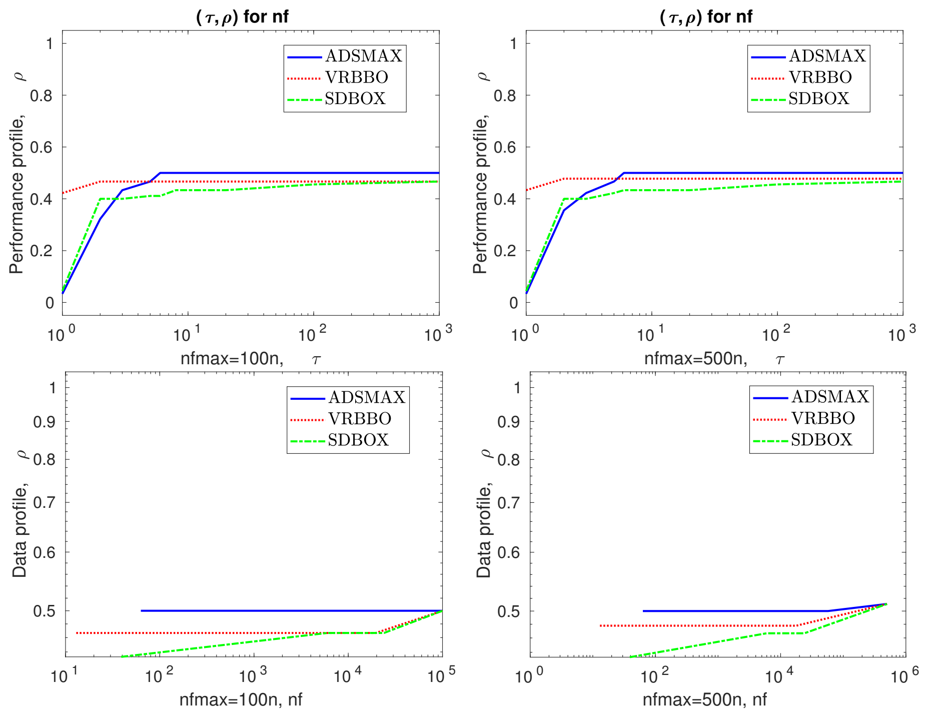
7.6 Results for large dimensions ($101 \le n \le 1000$)

Results on `CUTEst`. As discussed in Section 7.5, **VRBBO**, **FMINUNC**, and **SDBOX** were three rank robust and efficient solvers on `CUTEst`. Figure 6 displays the performance profiles and data profiles to compare these solvers, showing that **VRBBO** is much more robust than the others and is recommended for large scale problems from `CUTEst`.

Results on `GlobalTest`. As discussed in Section 7.5, **VRBBO**, **VRBBO-Q**, **ADSMAX**, and **SDBOX** were four rank robust and efficient solvers on `GlobalTest`. We conclude from the performance profiles and data profiles shown in Figure 6 that **VRBBO** is the first rank efficient and second rank robust solver for large scale problems from `GlobalTest`.



Results for large dimensions 101–1000 on `CUTEst`



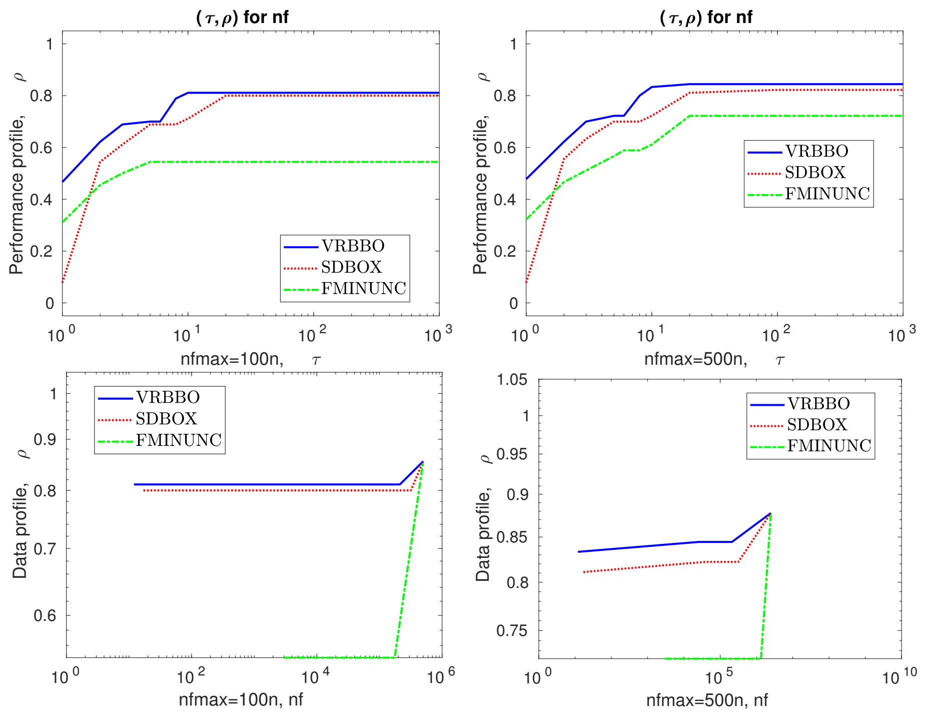
Results for large dimensions 101–1000 on `GlobalTest`

Fig. 6: Details as in Figure 2.

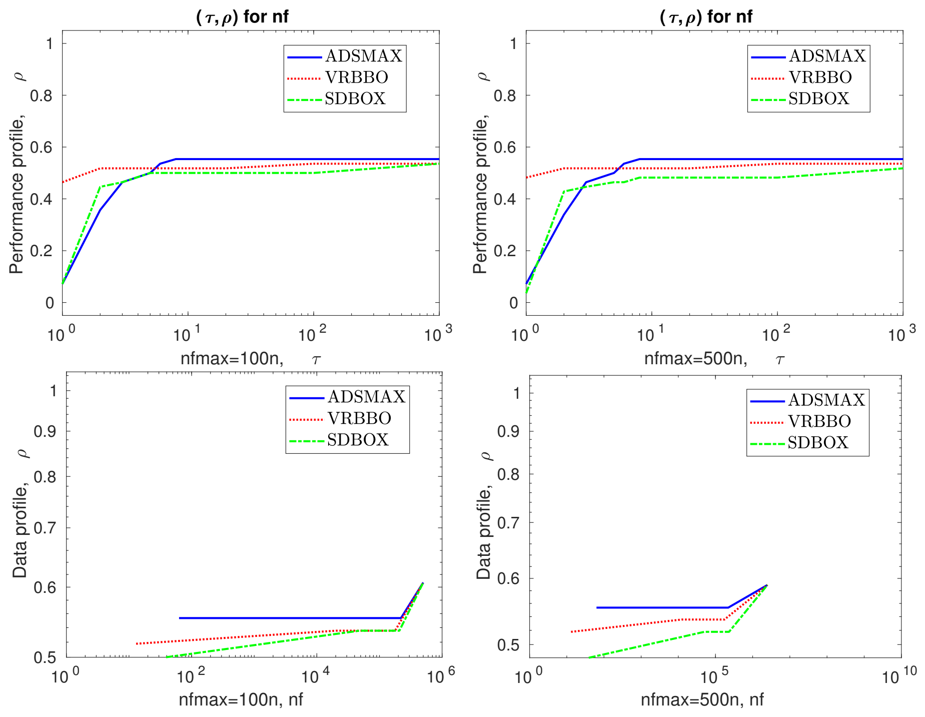
7.7 Results for very large dimensions ($1001 \leq n \leq 5000$)

Results on `CUTEst`. We conclude from the performance profiles and data profiles shown in Figure 7 that **VRBBO** is much more efficient and robust than the others for large scale problems from `CUTEst`.

Results on `GlobalTest`. We conclude from the performance profiles and data profiles shown in Figure 7 that **VRBBO** is the first rank efficient and second rank robust solver for large scale problems from `GlobalTest`.



Results very large dimensions 1001–5000 on `CUTEst`



Results for very large dimensions 1001–5000 on `GlobalTest`

Fig. 7: Details as in Figure 2.

8 Conclusion

We constructed an efficient randomized algorithm for unconstrained black box optimization problems. For the basic version of **VRBBO** with only random directions, the complexity bound for the nonconvex case, with probability arbitrarily close to 1, matches that found by Gratton et al. [27] for another algorithm. We also proved complexity bounds for **VRBBO** for the convex and strongly convex cases, with probability arbitrarily close to 1, essentially matching the bounds found by Bergou et al. [11], only valid in expectation.

An improved version of our algorithm has additional heuristic techniques that do not affect the order of the complexity results and which turn **VRBBO** into an efficient global solver although our theory guarantees only local minimizers. This version even was 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.

Two competitive versions of our algorithm were **VRBBO** and **VRBBO-Q** in low to high budgets on `CUTEst` and `GlobalTest` due to using all various directions and additional heuristic techniques. As a consequence of our extensive numerical results, **UOBYQA** is our recommendation for small scale problems in low to high budgets on `CUTESt` and **MCS** and **GLODS** on `GlobalTest` while **VRBBO** is our recommendation for medium up to very large scale problems in low to high budgets on `CUTESt` and `GlobalTest`.

9 Tools for VRBBO

9.1 Estimation of c

The following theorem was recently proved by Pinelis [50].

Theorem 6 *There is a universal constant $c_0 \leq 50$ 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*

$$(p^Tp)(q^Tq) \leq c_0 n(p^Tq)^2 \tag{68}$$

with probability $\geq 1/2$.

Pinelis also proved the lower bound $0.73 < c_0$ for the best Pinelis 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

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

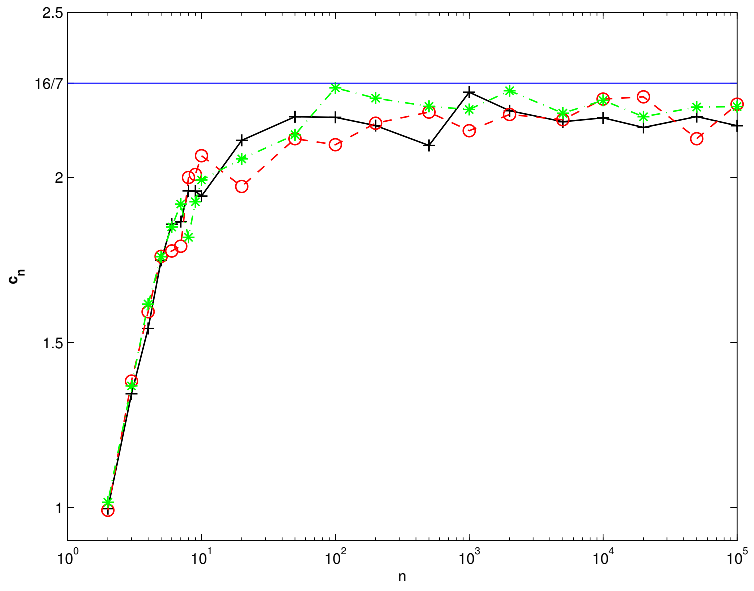


Fig. 8: The plot of c_n versus the dimension n suggests that $c_0 \approx 16/7$.

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 PinConst, estimating the Pinelis constant

```
 1: **Purpose**: Estimate $c_0$ satisfying (68) with probability $\geq 1/2$
 2: **Input**: $N$ (the total number of gradient evaluations), $D$ (vector of dimensions used)
 3: **Output**: $c_0$
 4: Set $M = |D|$.
 5: **for** $i = 1, \cdots, M$ **do**
 6:     **for** $k = 1, \cdots, N$ **do**
 7:         Generate random $g^k$ and $p^k$ with length $D_i$.
 8:         Compute $\texttt{gain}(k) = \frac{\|g^k\|_2 \|p^k\|_2}{|(g^k)^T p^k|}$.
 9:     **end for**
10:     Compute $\texttt{medgain}(i) = \texttt{median}(\texttt{gain})$ and $c(i) = \frac{(\texttt{medgain}(i))^2}{D_i}$.
11: **end for**
12: $c_0 = \max(c)$.
```

9.2 A list of test problems with f_{opt}

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

$$\|g^k\|_\infty \le 10^{-5}.$$

problem	dim	f_{opt}	$\|g\|_\infty$	$\|g\|_2$
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. M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at `https://www.gerad.ca/nomad/`.
2. C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization* **17** (2006), 188–217.
3. C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing (2017).
4. C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing (2017).
5. C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing (2017).
6. C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD (2009).
7. C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. Optim* **17** (January 2006), 642–664.
8. A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*. IEEE (2005).
9. A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim* **24** (January 2014), 1238–1264.
10. C. J. P. Bélisle, H. E. Romeijn, and R. L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Math. Oper. Res.* **18** (May 1993), 255–266.
11. E.H. Bergou, E. Gorbunov, and P. Richtárik. Stochastic three points method for unconstrained smooth minimization. *CoRR* **abs/1902.03591** (2019).
12. P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the prices algorithm for global optimization. *J. Glob. Optim.* **10** (1997), 165–184.
13. O. Burdakov, L. Gong, S. Zikrin, and Y. Yuan. On efficiently combining limited-memory and trust-region techniques. *Math. Program. Comput.* **9** (June 2016), 101–134.
14. J. Konečný and P. Richtárik. Simple complexity analysis of simplified direct search. *CoRR* **abs/1410.0390** (2014).
15. A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics (January 2009).
16. T. Csendes, L. Pál, J. O. H. Sendín, and J. R. Banga. The GLOBAL optimization method revisited. *Optim. Lett.* **2** (November 2007), 445–454.
17. A. L. Custódio and J. F. A. Madeira. GLODS: Global and local optimization using direct search. *J. Glob. Optim.* **62** (August 2014), 1–28.
18. A. L. Custódio, H. Rocha, and L. N. Vicente. Incorporating minimum frobenius norm models in direct search. *Comput. Optim. Appl.* **46** (August 2009), 265–278.
19. A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM J. Optim* **18** (2007), 537–555.
20. M. Dodangeh and L. N. Vicente. Worst case complexity of direct search under convexity. *Math. Program.* **155** (November 2014), 307–332.
21. M. Dodangeh, L. N. Vicente, and Z. Zhang. On the optimal order of worst case complexity of direct search. *Optim. Lett.* **10** (June 2015), 699–708.
22. E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (January 2002), 201–213.
23. C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.* **15** (1995), 585–608.
24. C. Elster and A. Neumaier. A method of trust region type for minimizing noisy functions. *Computing* **58** (March 1997), 31–46.
25. Yu. G. Evtushenko. Numerical methods for finding global extrema (case of a non-uniform mesh). *USSR USSR Comput. Math. & Math. Phys.* **11** (January 1971), 38–54.
26. N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* **60** (August 2014), 545–557.

27. S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim* **25** (January 2015), 1515–1541.
28. S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.* **26** (October 2011), 873–894.
29. W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM J. Optim* **17** (January 2006), 526–557.
30. E. R. Hansen. *Global Optimization Using Interval Analysis.* M. Dekker, New York, NY (1992).
31. N. Hansen. The CMA evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, pp. 75–102. Springer Berlin Heidelberg (2006).
32. N. J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.* **14** (April 1993), 317–333.
33. J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Optim* **2** (June 1973), 88–105.
34. W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *J. Glob. Optim.* **14** (1999), 331–355.
35. W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.* **35** (July 2008), 1–25.
36. S. Wright J. Nocedal. *Numerical Optimization.* Springer New York (2006).
37. M. Jamil and X. S. Yang. A literature survey of benchmark functions for global optimisation problems. *Int. J. Math. Model. Numer. Optim.* **4** (2013), 150.
38. C. T. Kelley. *Iterative Methods for Optimization.* Society for Industrial and Applied Mathematics (January 1999).
39. M. Kimiaei, A. Neumaier, and B. Azmi. LMBOPT – a limited memory method for bound-constrained optimization. `http://www.optimization-online.org/DB_HTML/2020/11/8089.html` (2020).
40. J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM J. Optim* **9** (January 1998), 112–147.
41. J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.* **28** (May 2019), 287–404.
42. S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software* **37** (2011), 1–15.
43. S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.* **21** (2002), 119–142.
44. A. Neumaier M. Kimiaei. Testing and tuning optimization algorithm (2020). in preparation.
45. J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* **20** (January 2009), 172–191.
46. J. Müller and J. D. Woodbury. GOSAC: global optimization with surrogate approximation of constraints. *Journal of Global Optimization* **69** (January 2017), 117–136.
47. Y. Nesterov. Random gradient-free minimization of convex functions. CORE discussion paper #2011/1 (2011).
48. Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Found. Comput. Math.* **17** (November 2015), 527–566.
49. A. Neumaier, H. Fendl, H. Schilly, and Thomas Leitner. VXQR: derivative-free unconstrained optimization based on QR factorizations. *Soft Comput.* **15** (September 2010), 2287–2298.
50. I. Pinelis. A probabilistic angle inequality. MathOverflow.
51. Boris T Polyak. Introduction to optimization. 1987. *Optimization Software, Inc, New York.*
52. M. Porcelli and 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. Softw.* **44** (July 2017), 1–25.

53. M. Porcelli and Ph. L. Toint. A note on using performance and data profiles for training algorithms. *ACM Transactions on Mathematical Software* **45** (June 2019), 1–10.
54. M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* **92** (May 2002), 555–582.
55. M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA. J. Numer. Anal.* **28** (February 2008), 649–664.
56. M. J. D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives (2009).
57. L. A. Rastrigin. *Statisticheskie metody poiska.* Nauka, (1968).
58. L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.* **56** (July 2012), 1247–1293.
59. Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* **11** (1997), 341–359.
60. P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications.* Springer Netherlands (1987).
61. A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization* **39** (February 2007), 197–219.
62. L. N. Vicente. Worst case complexity of direct search. *EURO J. Comput. Optim.* **1** (December 2012), 143–153.
63. A. A. Zhigljavsky. *Theory of Global Random Search.* Springer Netherlands (1991).