Efficient global unconstrained black box optimization

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

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

September 14, 2018

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

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

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

2010 MSC Classification: primary 90C56

Contents

1 Introduction

We consider the unconstrained optimization problem of minimizing a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\texttt{inf}, \texttt{NaN}\}$ by an oracle that returns for a given $x \in \mathbb{R}^n$ the function value $f(x)$. This problem (see, e.g., [6, 27]) is usually called black box optimization (BBO) or derivative-free optimization (DFO). Neither gradients nor Lipschitz constants nor structural information about f are assumed to be available. A huge literature exists about the problem, and we only mention a few pointers to the literature.

The techniques for solving BBO problems fall into two classes, deterministic and stochastic methods. We mainly discuss the stochastic case; for deterministic methods see, e.g., the book by Conn et al. [6] and its many references.

Our goal in this paper is to describe a new, practically very efficient stochastic method for which good complexity results can be proved. Stochastic methods for BBO go back to [29] and were later discussed especially in the framework of evolutionary optimization [3, 16, 28]. An algorithm loosely related to our new one (but without complexity guarantees) is the Hit-and-Run algorithm by Bélisle [4]. A multistart clustering global optimization method called GLOBAL was presented by Csendes et al. [7] for bound constrained global BBO. The covariance matrix adaptation evolution strategy (CMA-ES) was introduced by Auger & Hansen [3]. Past software of our group on BBO includes the deterministic algorithms GRID [11, 12] and MCS [17] and the stochastic algorithms SnobFit [18] and VXQR [23].

1.1 Complexity

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

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

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

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

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

Complexity bounds limit the size of $N(\varepsilon)$. The appropriate asymptotic form for the expression $N(\varepsilon)$, found by Gratton et al. [14], Bergou, Gorbunov & Richtárik [5], and Nesterov & Spokoiny [21, 22], depends on the properties (smooth, smooth convex, or smooth strongly convex) of f; cf. Subsection 2.1 below. Standard assumptions for the complexity analysis of BBO algorithms are:

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

(A2) On the level set

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

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

Gratton et al. [14] studied direct search with probabilistic descent. If, in each poll step, one chooses a fixed number of directions uniformly independently distributed on the unit

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

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

Table 1: Complexity results for stochastic BBO in expectation (Bergou et al. [5])

sphere, they proved that with overwhelmingly high probability a complexity bound $\mathcal{O}(n\varepsilon^{-2})$ holds. Bergou et al. [5] and Nesterov & Spokoiny [22] generalized this result to give algorithms with complexity results for the nonconvex, convex and strongly convex case shown in Table 1. In each case, the bounds are better by a factor of n than the best known complexity results for deterministic algorithms (by Konečný & Richtárik [19]) given in Table 2. Of course, being a stochastic algorithm, the performance guarantee obtained by Bergou et al. is slightly weaker, only valid in expectation.

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

Table 2: Complexity results for deterministic BBO (Konečný & Richtárik [19])

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

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

1.2 Algorithms and data structures

In the present paper we describe and analyze a new stochastic method, the *Vienna stochastic black box optimization algorithm* (`VSBBO`). It gives the same order of complexity as the one

by Bergou et al. but with a guarantee that holds with probability arbitrarily close to 1; see Table 3. Numerical results in Section 5 show that, in practice, `VSBBO` matches the quality of other state-of-the-art algorithms for BBO, including those with good heuristics but without a complexity guarantee.

To be competitive with the state of the art – which means finding, with reasonable accuracy, a global minimizer in small and large dimensions, or at least in the majority of cases a point as good as competing algorithms – requires our algorithm to be quite complex. As a consequence, the algorithm manipulates many quantities, which for convenience are grouped into a number of separate data structures to which the subalgorithms may have access if needed. On the other hand, to be able to prove complexity results requires a detailed description of all steps. To present the algorithms in a concise way we compiled the variables of all data structures in Table 5, and the in-out dependence of the algorithms (described later in full detail) on the data structure in Table 4. Flow charts for `setScale`, `MLS`, `FDS`, and `VSBBO` can be found in Figures 7 and 6 of Appendix B.

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

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

`VSBBO` initally calls the algorithm `setScales` to estimate a good scaling of norms, step lengths, and related control parameters. Then it uses in each iteration the fixed decrease search algorithm `FDS`, aimed at repeatedly reducing the function value by an amount of at least Δ to update the best point. If no progress is made in a call to `FDS`, Δ is reduced by a factor of 4. Once Δ is below a minimum threshold, the algorithm stops. Both `setScales` and `FDS` work by making repeated calls to the multi-line search `MLS`. `MLS` polls in a number of suitable chosen directions (defined by `direction`) in a line search fashion a few objective function values each in the hope of reducing the objective by more than Δ. `direction` generates 4 kinds of direction vectors: heuristic directions, subspace directions, random directions, and cumulative directions, explained in more detail in Subsections 2.2 and 2.3. Finally, `updateXF` and `updateCum` are auxiliary routines for updating the data needed for calculating subspace steps and cumulative steps, respectively.

In the detailed descriptions of`VSBBO`, given below, we use a pseudo-Matlab notation. In particular, the notation $.*$ denotes componentwise multiplication, $==$ is the comparison operator for equality, and $A_{:k}$ denotes the kth column of a matrix A.

`fun` (function handle for the objective function f)
`point` (structure with information about points and function values)
m, X, F (list of the m best points so far and their function values) b, $x = X_{:b}$, $f = F_b$ (best point and its function value) x_{init}, f_{init} (initial point and its function value) `xr`, `fr` (newest point and its function value) `xm`, `fm` (point `xr` $- p$ and its function value) `xl`, `fl` (point at `xr` $- 2p$ and its function value)
`step` (structure with information about the step management)
s (scaling vector), p (random search direction), `dp` (scaled length of p) $\delta_{\min}$, $\delta_{\max}$ (minimum/maximum norm of trial steps) δ (norm of trial steps), Δ (threshold for good improvement) $\Delta_{\min}$, $\Delta_{\max}$ (minimal/maximal threshold for good improvement) q (cumulative step), r (cumulative gain)
`par` (structure with parameters modified during the search)
T (maximal number of directions in `MLS`) `nf` (number of function evaluations used), λ (approximate Lipschitz constant) `good` (sufficiently improved function value?), `ss` (are we in `setScale`?) `dir` (direction type), `state` (state of cumulative step) `hs`, `hss` (two parameters for heuristic direction)
`tune` (structure with fixed parameters for tuning the performance)
$m_{\max} \geq 3$ (maximum number of best points kept) $T_0 \geq m_{\max}$ (maximal number of multi-line searches in `setScale`) $H \geq 1$ (maximal number of heuristic directions in `MLS`) $S \geq 3$ (maximal number of subspace directions in `MLS`) $R \geq 1$ (number of random direction per subspace direction in `MLS`) $E \geq 1$ (maximal number of extrapolations in `MLS`) `scSub` (scale subspace direction?), `scCum` (scale cumulative direction?) `cum` (cumulative step type, 0=none, 1, or 2) a (bound for cumulative step size) $\Delta_{\min}$, $\Delta_{\max}$ (minimal/initial maximal threshold for good improvement) δ_{init} (initial norm of trial steps) $\gamma_1 > 0$ (factor for finding δ), $\gamma_2 > 0$ (factor for adjusting $\Delta_{\max}$) $\gamma_3 > 1$ (factor for extrapolation test), γ_4 (factor for finding initial λ) γ_5 (factor for subspace step), γ_6 (factor for adjusting $\delta_{\min}$) γ_7 (factor for adjusting $\delta_{\max}$), γ_8 (factor for adjusting heuristic direction)

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

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., [2, 24, 25]. In Table 5, there are 6 integral, 2 binary, 1 ternary, and 12 continuous tuning parameters, giving a total of 21 parameters for tuning our algorithm. A small amount of tuning was done by hand. Automatic tuning of `VSBBO` will be considered elsewhere.

2 Line search techniques for BBO

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

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

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

Cumulative directions are based on a simple separable model assumption.

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

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

2.1 Probing a direction

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

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

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

where γ depends on x and p and satisfies one of

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

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

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

In all three cases,

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

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

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

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

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

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

and one of the following holds:

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

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

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

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

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

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

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

Proposition 2.1 will play a key role in the construction of our multi-line search `MLS` detailed in Subsection 2.4:

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

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

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

At this stage we can compute the lower bound

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

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

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

This is used in the cumulative contributions discussed now.

2.2 A cumulative direction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2 Algorithm. (cumulative update)

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

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

2.3 Choice of search direction

The following algorithm generates 4 kinds of direction vectors: heuristic directions (`dir` = 1), subspace directions (`dir` = 2), random directions (`dir` = 3), and cumulative directions (`dir` = 4).

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

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

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

2.3 Algorithm. (direction generator)

Purpose: Create subspace, random, or cumulative direction
function [step, par] = **direction**(point, step, par, tune);
switch dir **case** 1 % heuristic direction nmax = max$(n, 100)$; hs = randi$([1$ nmax$], 1)$; p = rand$(n, 1) - 0.5$; hss = nmax$/($nmax$\gamma_8 +$ hs$)$; p = hss $* p$; scale = 1; **case** 2 % subspace direction α = rand$(m, 1) - 0.5$; $\alpha = \alpha\gamma_5/\|\alpha\|_2$; $p = \sum_{k=1}^{m} \alpha_k(X_{:k} - X_{:b})$; scale=scSub; **case** 3 % random direction p p = rand$(n, 1) - 0.5$; scale=1; **case** 4 % cumulative direction p $p = q$; scale=scCum; **end**; **if** scale % scale to $\|p\| = \delta$ **if** $\lambda == 0$, $\lambda = 1$; end; $\delta = \max\left(\delta_{\min}, \min\left(\sqrt{\gamma_1\Delta/\lambda}, \delta_{\max}\right)\right)$; $p = s. * p * (\delta/\|p\|)$; dp=$\delta$; **else** % evaluate $\|p\|$ dp=$\|p\|$; **end**;

The heuristic direction enhances the global search properties. It is a random direction but with random scaling, decreasing on average with the number of function evaluations used. The subspace direction generated by `direction` with `par.dir` $= 2$ requires to keep a matrix X whose columns are the m best points and a vector F with their function values. Initially $m = 1$, to be increased up to a maximum of $m_{\max}$; later we overwrite each time the worst point. The following algorithm updates both X and F:

2.4 Algorithm. subspace update

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

2.5 Proposition. *For the random search direction generated by* `direction` *with* `dir` $= 3$,

the output p of `direction` *satisfies* $\|p\| = \delta$ *and, with probability* $\geq \frac{1}{2}$,

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

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

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

2.4 A multi-line search

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

Schematically, `MLS` works as follows:

(i) At first, at most H iterations with heuristic directions are used.

(ii) Then, except in the final iteration, subspace directions alternate with R random directions.

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

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

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

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

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

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

where λ_0 is the initial value of λ.

2.6 Algorithm. A multi-line search (MLS)

Purpose: Improve function value along multiple directions

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

xm $= X_{:b}$; fm $= F(b)$; x_{init} = xm; f_{init} = fm; nE $= 0$; $r = 0$; $q = \text{zeros}(n, 1)$; $t = 0$;
state $= -1$; good $= 0$;
while $t < T$,
  switch state
  case -1 % new direction
    $t = t + 1$;
    if ss, sub $= (\text{mod}(t, R) == 0$ & $t < T)$; **else** sub $= 0$; **end**;
    if $t \leq H$, dir=1; % pick heuristic step
    elseif sub, dir=2; % pick subspace step
    elseif $t < T$, dir=3; % pick random step
    else % check for cumulative step
      if cum $== 1$, $q = x - x_{\text{init}}$; % first cumulative step
      elseif cum $== 2$ & $r \geq \Delta$, % second cumulative step
      end;
      if cum > 0 & all$(q == 0)$,
        dir=3; % pick random step
      else
        dir $= 4$; % pick cumulative step
        if cum $== 1$, state=1; fl $= f_{\text{init}}$; **end**; % use initial value
      end;
    end;
    [step, par] = direction(point, step, par, tune);
  case 0, % extrapolate at most T times
    nE $=$ nE $+ 1$;
    if nE $== 1$, fl=fm; **else** $p = 2p$; **end**;
    xm $= X_{:b}$; fm $= F(b)$;
  case 1, % opposite direction
    $p = -p$; fl=fr;
  end;
  xr $=$ xm $+ p$; fr $=$ fun(xr); nf $=$ nf $+ 1$; df $=$ fm $-$ fr;
  if df > 0,
    [point] = updateXF(point, tune); % update X and F
    ext $= (\text{df} > \gamma_3 \Delta)$; % large gain; extrapolate
    sext $= (\text{nE} < E$ & ext$)$; % sequence large gain; sequence extrapolate
    if sext, state $= 0$; **continue**; **end**;
    % update λ and cumulative direction
    $\lambda = \max(\lambda, |\text{fl} + \text{fr} - 2\text{fm}|/\text{dp}^2)$;
    if cum $== 2$, [step] = updateCum(point, par, tune); **end**;
    if $\sim$ ext, xm $= X_{:b}$; fm $= F(b)$; **end**;
  else
    opp=(state < 0 & df $< -\Delta$); % opposite gain expected
    if opp, state $= 1$; **continue**; **end**;
    % update λ and cumulative direction
    $\lambda = \max(\lambda, |\text{fl} + \text{fr} - 2\text{fm}|/\text{dp}^2)$;
    if cum $== 2$, [step] = updateCum(point, par, tune); **end**;
  end;
  gain $= f_{\text{init}} - F(b)$; good $= (\text{gain} > \Delta)$;
  if good, **return**; **end**;
  state $= -1$; % line search completed
end;

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

2.7 Theorem. *Assume that (A1) holds.*

(i) Algorithm 2.6 uses at most $2T + E$ function evaluations and does not increase f.

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

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

where c is the constant in Proposition 2.5 and

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

Proof. (i) In the while loop of `MLS`, a direction p is generated and at most two function values are computed, unless an extrapolation step is performed. In this case, at most E additional function values are computed during the extrapolation stage, and the loop is ended. As a result, `MLS` uses at most $2T + E$ function evaluations. Clearly, the function value of the best point does not increase.

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

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

for $t = 1, \ldots, T$. Let $\mathcal{R} := \{H < t < T \mid \texttt{sub}(t) = 0\}$, where $\texttt{sub}(t)$ is the value of `sub` in iteration t. For $t \in \mathcal{R}$, the search direction was generated by `direction` as a random direction, hence Proposition 2.5 implies that for $t \in \mathcal{R}$,

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

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

3 A stochastic descent algorithm for BBO

3.1 Setting the scales

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

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

3.1 Algorithm. Setting the scales (setScales)

Purpose: Estimate $\Delta_{\max}$, initial λ and s
function [`point`, `step`, `par`] = **setScales** (`fun`, `point`, `step`, `par`, `tune`);
% get point
$n = \text{length}(x)$; $f = \texttt{fun}(x)$; $X = x$; $F = f$; $b = 1$; $m = 1$;
% get step
$\Delta = \Delta_{\max}$; $\delta_{\max} = \delta_{\text{init}}$; $\delta = \delta_{\max}$; $\delta_{\min} = \delta_{\max}$;
% get par
$\lambda = 0$; $T = H + (S-1)(R+1) + 2$;
`ss` $= 0$; % ignore subspace direction in `direction`
% stage 1: select only heuristic/random/cumulative directions in `MLS`
for $t = 1 : T_0$,
[`point`, `par`] = `MLS`(`fun`, `point`, `step`, `par`, `tune`);
if $m == 1$, [`point`] = `updateXF`(`point`, `tune`); **end**;
end;
`ss` $= 1$; % add subspace direction to `direction`
% stage 2: estimate s, $\Delta_{\max}$ and initial λ
for $i = 1 : m$, $\texttt{dX}_{:i} = X_{:i} - X_{:b}$; **end**;
$s = \sup_{i=1:m} (\texttt{dX}_{:i})$; $s(s == 0) = 1$;
% get maximal desired gain $\Delta_{\max}$
$\texttt{dF} = \underset{i=1:m}{\text{median}} \lvert F_i - F_b \rvert$;
if `dF` $== 0$,
$\texttt{mdX} = \underset{i=1:m}{\text{mean}} \lVert \texttt{dX}_{:i} \rVert$;
if `mdX` $\neq 0$,
$\Delta_{\max} = \gamma_2 \sqrt{\texttt{mdX}}$; $\Delta = \Delta_{\max}$; $\lambda = \gamma_4 \sqrt{\texttt{mdX}/n}$;
else
if $\Delta_{\max} == 0$, $\lambda = 1/\sqrt{n}$; **else** $\lambda = \Delta_{\max}/\sqrt{n}$; **end**
end
else
$\Delta_{\max} = \gamma_2 \texttt{dF}$; $\Delta = \Delta_{\max}$; $\lambda = \gamma_4 \texttt{dF}/\sqrt{n}$;
end
$\delta_{\min} = \gamma_6 \texttt{hss}$; $\delta_{\max} = \gamma_7 \texttt{hss}$; $\delta = \delta_{\max}$;

3.2 Probing for fixed decrease

Based on the preceding results we present a fixed decrease search algorithm whose goal is to use repeated calls to the multi-line search `MLS` in order to improve the function value

each time by a fixed amount Δ.

3.2 Algorithm. Fixed Decrease Search (FDS)

Purpose: Repeatedly improve function value by $> \Delta$
function [point] = **FDS**(fun, point, step, par, tune);
while 1,
[point, par] = MLS(fun, point, step, par, tune);
if $\sim$ good, **break**; **end**;
end;

3.3 Theorem. *Assume that (A1) and (A2) hold. Then:*

(i) The number of function evaluations of FDS *is bounded by* $(2T+E)\dfrac{f_0-\widehat{f}}{\Delta}$, *where* f_0 *is the initial value of* f *and* $\widehat{f}$ *is the minimal function value.*

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

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

for at least one of the gradients g *encountered. Here* c *is the constant from Proposition 2.5 and, if* λ_0 *denotes the value of* λ *before the first execution of* FDS,

$$L' := \frac{L^2\gamma_1}{\lambda_0} + 4L + 4\frac{\lambda_0 + L}{\gamma_1}. \tag{19}$$

Proof. Because of (A2), $\widehat{f}$ is finite. Denote by f_k the result of the kth execution of MLS. By the definition of good, $f_k \leq f_{k-1} - \Delta$ for all but the last value of k. If a total of N iterations were performed, we conclude that $\widehat{f} \leq f_N \leq f_0 - N\Delta$, so that $N \leq \dfrac{f_0-\widehat{f}}{\Delta}$. Since each iteration uses at most $2T+E$ function evaluations, (i) follows.

(ii) By Theorem 2.7, $\|g\|_* \leq \sqrt{cn}\Gamma(\delta)$ holds with probability $\geq 1-\eta_1$. Thus it is sufficient to show that

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

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

Case 1: $\delta = \sqrt{\dfrac{\gamma_1\Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta + \frac{2\Delta}{\delta} = L\sqrt{\frac{\gamma_1\Delta}{\lambda}} + 2\sqrt{\frac{\lambda\Delta}{\gamma_1}} = \Lambda\sqrt{\Delta},$$

where

$$\Lambda := L\sqrt{\frac{\gamma_1}{\lambda}} + 2\sqrt{\frac{\lambda}{\gamma_1}}. \tag{21}$$

CASE 2: $\delta = \delta_{\min} \geq \sqrt{\frac{\gamma_1 \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta_{\min} + \frac{2\Delta}{\delta_{\min}} \leq L\delta_{\min} + 2\sqrt{\frac{\lambda\Delta}{\gamma_1}} \leq L\delta_{\min} + \Lambda\sqrt{\Delta}.$$

CASE 3: $\delta = \delta_{\max} \leq \sqrt{\frac{\gamma_1 \Delta}{\lambda}}$. In this case,

$$\Gamma(\delta) = L\delta_{\max} + \frac{2\Delta}{\delta_{\max}} \leq L\sqrt{\frac{\gamma_1\Delta}{\lambda}} + \frac{2\Delta}{\delta_{\max}} \leq \Lambda\sqrt{\Delta} + \frac{2\Delta}{\delta_{\max}}.$$

Thus in each case,

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

Now (20) follows since by (15) and (21), $\Lambda^2 = \frac{L^2\gamma_1}{\lambda} + 4L + \frac{4\lambda}{\gamma_1} \leq L'$. □

3.3 The VSBBO algorithm

We now have all ingredients to formulate VSBBO, the *Vienna stochastic black box optimization* algorithm. It uses in each iteration the fixed decrease search algorithm to update the best point. If no progress is made in the corresponding FDS call, Δ is reduced by a factor of 4. Once Δ is below a minimum threshold, the algorithm stops.

3.4 Algorithm. Vienna stochastic black box optimization (VSBBO)

Purpose: Solve stochastic black box optimization
function [x,f] = **VSBBO**(fun, x, tune)

[point, step, par] = setScales (fun, point, step, par, tune);
while 1,
  [point] = FDS(fun, point, step, par, tune);
  % gain of Δ unlikely with step of length δ
  if $\Delta \leq \Delta_{\min}$, **break**; **end**;
  % reduce Δ
  $\Delta = \Delta/4$;
end;

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

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

4 Complexity analysis of VSBBO

4.1 The general (nonconvex) case

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

$$n_k < (2T+E)\Big(T_0 + \frac{f_0 - \widehat{f}}{3\Delta_{\min}}\Big).$$

Proof. In `setScale`, we need $n_0 \le (2T+E)T_0$ function evalutions. By construction, the kth call to `FDS` uses $\Delta = 4^{1-k}\Delta_{\max}$, hence uses by Theorem 3.3(i) at most $(2T+E)\dfrac{f_0-\widehat{f}}{4^{1-k}\Delta_{\max}}$ function evalutions. Therefore, the total number of function evaluations up to iteration k is

$$\begin{aligned} n_k &\le (2T+E)T_0 + \sum_{j=1}^{k}(2T+E)\frac{f_0-\widehat{f}}{4^{1-k}\Delta_{\max}} \\ &= (2T+E)T_0 + (2T+E)(f_0-\widehat{f})\frac{4^k-1}{3\Delta_{\max}} < (2T+E)\Big(T_0 + \frac{f_0-\widehat{f}}{3\Delta_{\min}}\Big) \end{aligned}$$

since $\Delta_{\max} \ge 4^k\Delta_{\min}$. □

4.2 Theorem. *Assume that (A1) and (A2) hold. With c from Proposition 2.5, let $\zeta \ge 9c$, $0 < \eta < 1$, and let K be a positive integer. Then, for any $\varepsilon > 0$, if*

$$T \ge H + S + 1 + \log_2 \frac{K+1}{\eta}, \tag{22}$$

$$\max\Big(4^{-K}\Delta_{\max}, \frac{\varepsilon^2}{\zeta L' n}\Big) \le \Delta_{\min} \le \frac{\varepsilon^2}{9cL'n}, \tag{23}$$

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

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

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

Proof. By the rule of updating Δ in `VSBBO`, $\Delta_k = 4^{-k}\Delta_{\max} \le \Delta_{\min}$ for $k \ge K$. Hence at most K steps of `FDS` are performed. By (22), we have $\eta_1 = 2^{-(T-H-S-1)} \le \eta/(K+1)$. Thus by Theorem 3.3(ii), we have, with probability $\ge 1-(K+1)\eta_1 \ge 1-\eta$, for at least one of the gradients g encountered,

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

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

$$n_K \le (2T+E)\Big(T_0 + \frac{f_0 - \widehat{f}}{3\Delta_{\min}}\Big) \le (2T+E)\Big(T_0 + \frac{\zeta L' n}{3\varepsilon^2}(f_0 - \widehat{f})\Big) = \mathcal{O}\Big(n\varepsilon^{-2}\Big).$$

□

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

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

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

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

If the number of iterations is finite then $\mathcal{S}$ is finite. Let $k_0 \in \mathcal{S}$ be maximal. Then the updating rule for Δ_k implies that $\Delta_k = \Delta_{k_0}/4^{k-k_0} \to 0$ as $k \to \infty$. If the number of iterations is not finite then $\mathcal{S}$ and $\mathcal{U}$ are infinite, and again $\Delta_k \to 0$ as $k \to \infty$. Theorem 3.3(ii) now implies that $\inf_k \|g_k\|_* = 0$. □

4.2 The convex case

4.4 Theorem. *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. With c from Proposition 2.5, let $\zeta \ge 9c$, $0 < \eta < 1$, and let K be a positive integer. For any $\varepsilon > 0$, if (22)–(24) hold then Algorithm 3.4 finds after at most $\mathcal{O}\big(n\varepsilon^{-1}\big)$ function evaluations with probability $\ge 1-\eta$ a point x with*

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

where r_0 is given by (8).

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

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

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

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

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

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

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

$$0 \le f_k - \widehat{f} \le f_{k-1} - \widehat{f} - N_k \Delta_k \le \varepsilon r_0 - N_k \Delta_k = \varepsilon r_0 - N_k 4^k \Delta_{\min},$$

leading to

$$N_k \le 4^{-k} \frac{\varepsilon r_0}{\Delta_{\min}} \le 4^{-k} \frac{9 r_0 c L' n}{\varepsilon} \tag{30}$$

by (23). The bound for the number of function evaluations is now obtained from Theorem 4.2 and (30):

$$\begin{aligned} n_K &\le (2T + E)\Big(T_0 + \sum_{j=1}^{K} N_j\Big) \le (2T + E)\Big(T_0 + \frac{9 r_0 c L' n}{\varepsilon} \sum_{j=1}^{K} 4^{-j}\Big) \\ &\le (2T + E)\Big(T_0 + \frac{3 r_0 c L' n}{\varepsilon}\Big) = \mathcal{O}\Big(n \varepsilon^{-1}\Big). \end{aligned}$$

□

4.3 The strongly convex case

4.5 Theorem. *Let f be convex on $\mathcal{L}(x_0)$ and assume that (A1) and (A2) hold. Under the assumptions of Theorem 4.2, Algorithm 3.4 finds after at most $\mathcal{O}\Big(n \log_2 \varepsilon^{-1}\Big)$ function evaluations with probability $\ge 1-\eta$ a point x with*

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

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

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

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

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

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

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

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

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

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

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

so that

$$N_k \leq \frac{\varepsilon^2}{2\sigma\Delta_{\min}} \leq \frac{9cL'n}{2\sigma} \tag{35}$$

by (23). Now Theorem 4.2 implies

$$\begin{aligned} n_K &\leq (2T+E)\Big(T_0 + \sum_{j=1}^{K} N_j\Big) \leq (2T+E)\Big(T_0 + \frac{9cL'n}{2\sigma}K\Big) \\ &= (2T+E)\Big(T_0 + \frac{9cL'n}{\sigma}\log_2\Big(\frac{\Delta_{\max}}{\Delta_{\min}}\Big)^{\frac{1}{2}}\Big) \\ &= (2T+E)\Big(T_0 + \frac{9cL'n}{\sigma}\log_2\frac{3(cL'n\Delta_{\max})^{\frac{1}{2}}}{\varepsilon}\Big) = \mathcal{O}\Big(n\log_2\varepsilon^{-1}\Big). \end{aligned}$$

□

5 Numerical results

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

5.1 Test problems used

`VSBBO` was compared with other codes from the literature on all 549 unconstrained problems from the CUTEst [13] collection of test problems for optimization with up to 1000 variables, in case of variable dimension problems for all allowed dimensions in this range. To avoid guessing the solution of toy problems with a simple solution (such as all zero or all one), we shifted the arguments, for all $i = 1, \ldots, n$, by

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

where m ^

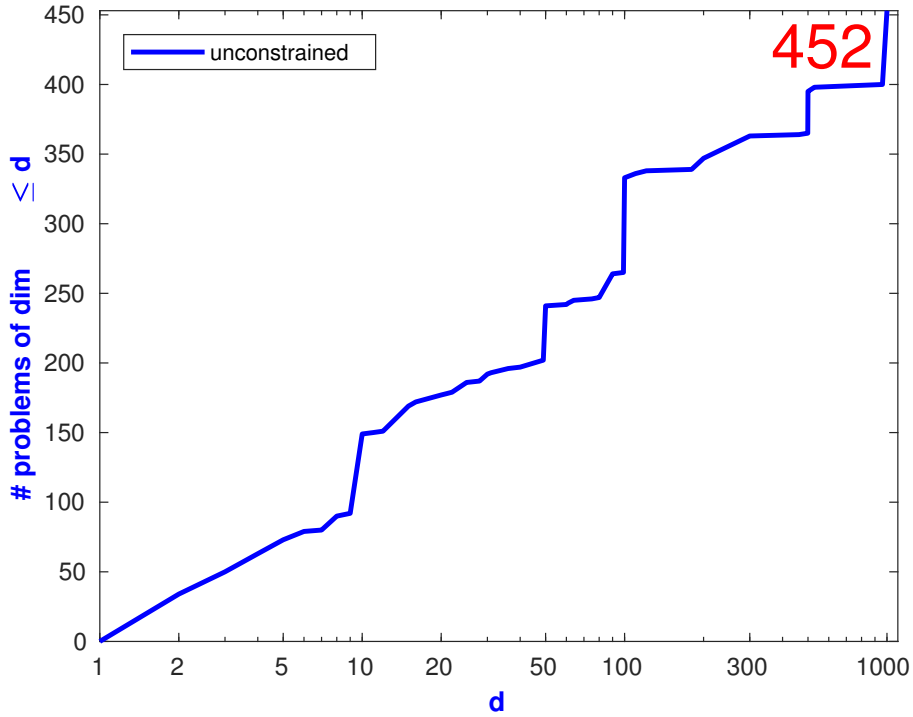


Figure 1: The number of problems with at most `d` variables solved by at least one solver. There were a total of 452 such problems; of these, 177 had dimension ≤ 20.

`nf` and `msec` denote the number of function evaluations and the time in milliseconds, respectively. We limited the budget available for each solver by allowing at most 500 seconds of run time and at most $2n^2+200n+5000$ function evaluations for a problem with n variables. A problem is considered solved if it reaches the target accuracy

$$q_f := (f - f_{\text{best}})/(f_{\text{init}} - f_{\text{best}}) \leq 0.05,$$

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

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

The quadratic growth formula was used to ensure that most of the original problems could be solved by some solver within the budget alloted. We excluded problem `CHNRSNBM:25` of dimension 25 since it was not solved within the alloted budget by any of the solver compared, leaving 452 problems. Figure 1 plots the number of problems with at most $\leq d$ variables versus d.

5.2 Default parameters for `VSBBO`

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

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

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

`mmax` $= 3$; $T_0 = 15$; $H = 10$; $S = 2$; $R = 10$; $E = 50$;
`scSub` $= 0$; `scCum` $= 0$; `cum` $= 2$;
$a = 1$; $\Delta_{\min} = 0$; $\Delta_{\max} = 0$; $\delta_{\text{init}} = 0.001$;
$\gamma_1 = 1$; $\gamma_2 = 0.01$; $\gamma_3 = 2$; $\gamma_4 = 0.001$;
$\gamma_5 = 1$; $\gamma_6 = 1$; $\gamma_7 = 10$; $\gamma_8 = 5000$;

5.3 Codes compared

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

- `RDSfs, RDSvs` and `PRDS`, obtained from the authors of Bergou et al. [5], are three versions of a stochastic direct search method with good complexity guarantees.
- `BFO`, obtained from

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

  is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli & Toint [25].
- `CMA-ES`, 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 [3]. We used `CMA-ES` with the following parameters:

```
oCMAES.MaxFunEvals = nfmax; oCMAES.DispFinal = 0;
oCMAES.DispModulo = 0; oCMAES.LogModulo = 0;
oCMAES.SaveVariables = 0; oCMAES.MaxIter = Inf;
```

- 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. [7]. We used GLOBAL with the following parameters:

oGLOBAL.MAXFNALL = nfmax; oGLOBAL.MAXFN = nfmax/10;
oGLOBAL.N100 = min(10n, nfmax/10); oGLOBAL.DISPLAY ='off';
oGLOBAL.METHOD ='unirandi'.

- MCS, obtained from

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

  is the deterministic global optimization by multilevel coordinate search by Huyer & Neumaier [17]. We used MCS with the following parameters:

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

- BC-DFO, obtained from Anke Troeltzsch (personal communication), is a deterministic model-based trust-region algorithm for derivative-free bound-constrained minimization by Gratton et al. [15].

- 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 [8, 9].

- fminseach, 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. [20]. We use fminseach with the options set by optimoptions as follows:

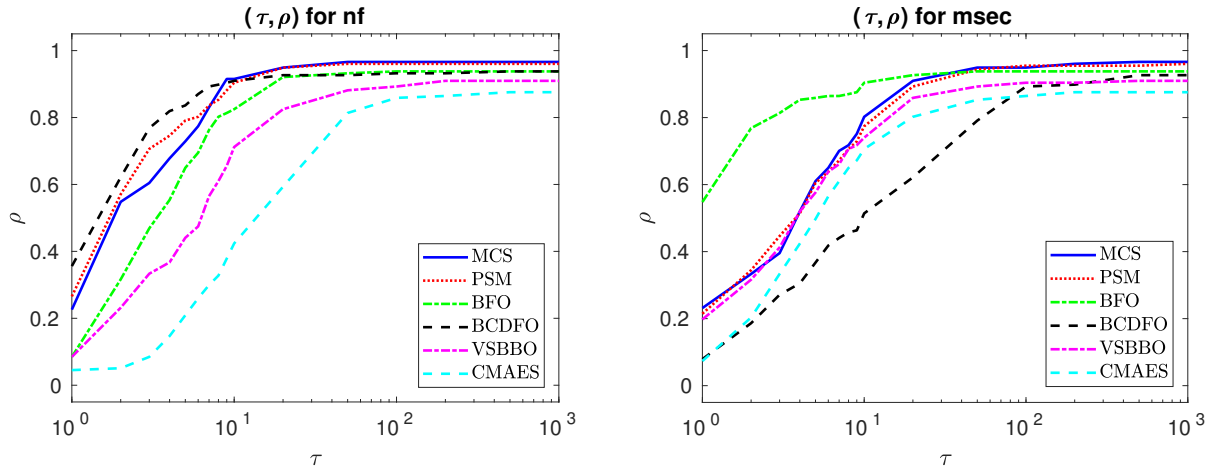
```
opts = optimset('Display','Iter', 'MaxIter', Inf,'MaxFunEvals', ...
       limits.nfmax,'TolX', 0, 'TolFun',0,'ObjectiveLimit',-1e-50);
```

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

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

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

Perfomance plots [10] for two cost measures `nf` (number of function evaluations needed to reach the target) and `msec` (time used in milliseconds) are shown in Figure 2.



a) b)

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

For a more refined statistics, we adapted the test environment described in [1], Azmi et al. to work with derivative-free solvers. For a given collection S of solvers, the strength of a solver $so \in S$ – relative to an ideal solver that matches on each problem the best solver – is measured, for any given cost measure c_s by the number, q_{so} defined by

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

called the **efficiency** of the solver so with respect to this cost measure. In the tables, efficiencies are given in percent. Larger efficiencies in the table imply a better average behavior; a zero efficiency indicates failure. All values are rounded (towards zero) to integers. Mean efficiencies are taken over the 452 problems tried by all solvers and solved by at least one of them, from a total of 453 problems. In the following tables, #100 and !100 count the number of times we have nf efficiency 100% or unique nf efficiency 100%. T_{mean} is defined by

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

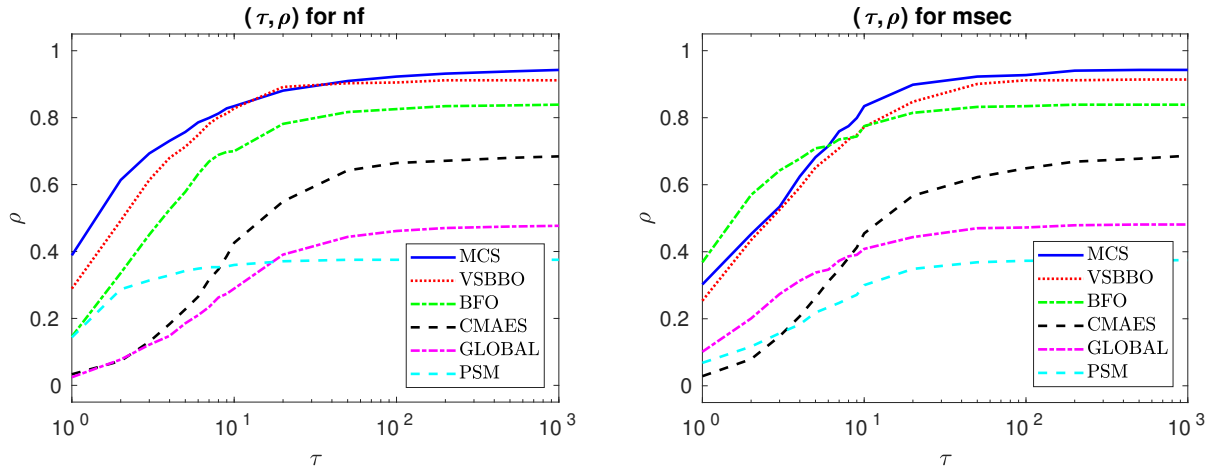
Failure reasons were reported in the anomaly columns:

- n indicates that `nf` $\geq 2n^2 + 200n + 5000$ `was reached`.
- t indicates that `sec` ≥ 500 `was reached`.
- f indicates that the `algorithm failed` for other reasons.

The resulting statistics table is

stopping test: $q_f \le$ `0.05`, `sec` $\le$ `500`, `nf` $\le 2n^2+200n+5000$										
177 of 177 problems solved									mean efficiency	
dim∈[1,20]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
MCS	mcs	171	35	31	187	0	0	6	50	28
PSM	psm	170	41	37	480	3	0	4	55	29
BFO	bfo	166	14	13	70	0	0	11	36	51
BCDFO	bcd	166	58	50	1779	0	0	11	59	15
VSBBO	vsbb	161	11	11	295	16	0	0	27	24
CMAES	cma	155	5	1	201	2	0	20	13	17
GLOBAL	glo	132	3	2	105	19	0	26	13	24
fminsearch	fmin	131	0	0	250	28	0	18	8	12
PRDS	prd	92	7	7	145	85	0	0	12	22
RDSfs	rfs	85	7	5	76	92	0	0	13	28
RDSvs	rvs	82	14	9	75	95	0	0	16	27

In the times, the (for some problems significant) setup time for CUTEST is not included. Although running times are reported, the comparison of times is not very reliable for several reasons:
(i) The times were obtained under different conditions (solver source code Fortran, C and Matlab).
(ii) In unsuccessful runs, the actual running time depends a lot on when and why the solver was stopped. Table entries use the maximal allowed time (500 sec) for each unsuccessful run.



a) b)

Figure 3: Large dimensions: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the percentage of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

5.5 Results for large dimensions ($n \leq 1000$)

We collected the same statistics for all problems of dimension ≤ 1000 (including the small dimensions). Perfomance plots [10] are shown in Figure 3.

The statistics table is now

stopping test: $q_f \leq$ `0.05`, `sec` $\leq$ `500`, `nf` $\leq 2n^2 + 200n + 5000$										
452 of 453 problems solved									mean efficiency	
dim∈[1,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
MCS	mcs	427	144	136	8303	0	0	26	54	39
VSBBO	vsbb	414	118	115	4834	35	0	4	49	36
BFO	bfo	380	53	50	5173	0	0	73	34	45
CMAES	cma	314	12	7	10184	44	0	95	12	11
GLOBAL	glo	218	10	8	976	65	0	170	10	17
PSM	psm	170	41	37	480	3	0	280	21	11
BCDFO	bcd	166	58	50	1779	0	0	287	23	5
PRDS	prd	151	17	16	1346	244	0	58	11	16
RDSfs	rfs	143	9	6	895	252	0	58	10	20
RDSvs	rvs	141	18	12	949	254	0	58	12	20
fminsearch	fmin	131	0	0	250	28	0	294	3	5

5.6 Results for the best two solvers (`MCS`, `VSBBO`)

In large dimensions, `VSBBO` stands out as the most robust stochastic solver, and `MCS` as the most robust deterministic solver.

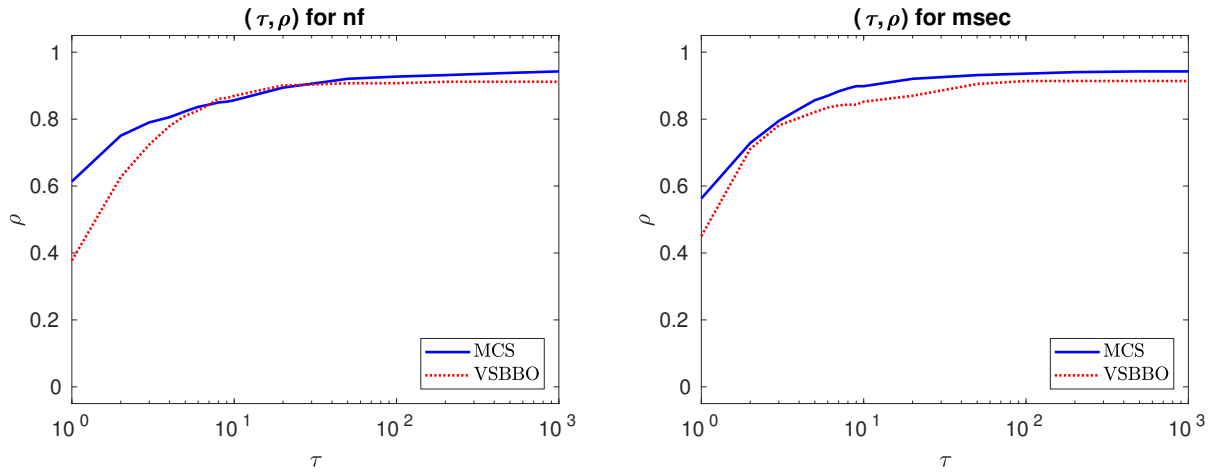


Figure 4: Large dimensions: Performance plots for (a) `nf`/(`best nf`) and (b) `msec`/(`best msec`). ρ designates the percentage of problems solved within a factor τ of the best solver. Problems solved by no solver are ignored.

The high quality of our new solver VSBBO is clearly visible. Its performance is close to the most robust solver MCS and almost complementary to it. Note that MCS was designed specifically for global optimization whereas VSBBO has a very simple globalization strategy only. Thus we may expect further improvements by a suitable combination of the techniques used in the design of these solvers, especially in small dimensions.

On the complete test set, the two most robust solvers MCS and VSBBO together solved within the given budget almost all 452 problemsolved by some solver. The 7 exceptions were the problems INDEF:10, STRATEC, CHNROSNB:25, CHNROSNB:50, CHNRSNBM:50, INDEF:50, FLETCHBV:100 (numbers after the colon indicate dimensions for variable-dimensional problems).

stopping test: $q_f \leq$ 0.05, sec $\leq$ 500, nf $\leq 2n^2 + 200n + 5000$										
445 of 453 problems solved									mean efficiency	
dim∈[1,1000]						# of anomalies			for cost measure	
solver		solved	#100	!100	T_{mean}	#n	#t	#f	nf	msec
MCS	mcs	427	278	274	8303	0	0	26	74	73
VSBBO	vsbb	414	171	167	4834	35	0	4	62	67

References

[1] B. Azmi, M. Kimiaei , A. Neumaier, LMBOPT - a limited memory method for bound-constrained optimization, in preprint (2018). [25]

[2] C. Audet and D. Orban, Finding optimal algorithmic parameters using derivativefree optimization, SIAM Journal on Optimization, 17 (2006), 642–664. [6]

[3] A. Auger and N. Hansen, A restart CMA evolution strategy with increasing population size. In: The 2005 IEEE congress on evolutionary computation, 2 (2005), 1769–1776. [3, 23]

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

[5] E.H. Bergou, E. Gorbunov and P. Richtárik, Random direct search method for minimizing nonconvex, convex and strongly convex functions, Manuscript (2018). [3, 4, 23]

[6] A.R. Conn, K. Scheinberg, and L.N. Vicente, Introduction to derivative-free optimization. SIAM, Philadelphia, PA, 2009. [2, 3]

[7] T. Csendes, L. Pál, J.O.H Sendin and J.R Banga, The GLOBAL optimization method revisited, Optim. Lett. 2 (2008), 445–454. [3, 24]

[8] A.L. Custódio, L.N. Vicente, Using sampling and simplex derivatives in pattern search methods, SIAM Journal on Optimization. 18 (2007), 537–555. [24]

[9] A.L. Custódio, H. Rocha, L.N. Vicente, Incorporating minimum Frobenius norm models in direct search, Computational Optimization and Applications. 46 (2010), 265–278. [24]

[10] Dolan E, Moré JJ, Benchmarking optimization software with performance profiles, Mathemathical Programming. 91 (2002), 201–213. [25, 27]

[11] C. Elster and A. Neumaier, A grid algorithm for bound constrained optimization of noisy functions, IMA J. Numer. Anal. 15 (1995), 585–608. [3]

[12] C. Elster and A. Neumaier, A trust region method for the optimization of noisy functions, Computing 58 (1997), 31–46. [3]

[13] N.I.M. Gould, D. Orban, Ph.L. Toint, CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Computational Optimization and Applications, 60(3) (2015), 545–557. [22]

[14] S. Gratton, C.W. Royer, L. N. Vicente and Z. Zhang, Direct search based on probablistic descent, SIAM J. Optimization 25 (2015), 1515–1541. [3]

[15] S. Gratton, Ph. L. Toint, and A. Troeltzsch, An active-set trust-region method for derivative-free nonlinear bound-constrained optimization, Opt. Methods and Software, 26(4–5) (2011) 875–896. [24]

[16] N. Hansen, The CMA evolution strategy: a comparing review, pp. 75–102 in: Towards a new evolutionary computation. Advances on estimation of distribution algorithms (J.A. Lozano, ed.), Springer, Berlin 2006. [3]

[17] W. Huyer and A. Neumaier, Global optimization by multilevel coordinate search, J. Global Optimization 14 (1999), 331–355. [3, 24]

[18] W. Huyer and A. Neumaier, SNOBFIT–stable noisy optimization by branch and Fit, ACM Trans. Math. Software 35, Article 9 (2008). [3]

[19] J. Konečný and P. Richtárik, Simple complexity analysis of simplified direct search, Manuscript (2014), `https://arxiv.org/abs/1410.0390`. [4]

[20] J.C. Lagarias, J.A. Reeds, M.H. Wright, P.E. Wright, Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. SIAM Journal of Optimization, 9(1) (1998) 112–147. [24]

[21] Y. Nesterov, Random gradient-free minimization of convex functions, CORE discussion paper #2011/1, unpublished (2011). [3]

[22] Y. Nesterov and V. Spokoiny, Random gradient-free minimization of convex functions, Found. Comput. Math. 17 (2017), 527-566. [3, 4]

[23] A. Neumaier, H. Fendl, H. Schilly and T. Leitner, Derivative-free unconstrained optimization based on QR factorizations, Soft Computing 15 (2011), 2287–2298. [3]

[24] M. Porcelli, Ph. L. Toint, A note on using performance and data profiles for training algorithms, (2017), `https://arxiv.org/abs/1711.09407` [6]

[25] M. Porcelli, Ph. L. Toint, BFO, a trainable derivative–free Brute Force Optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables, ACM Transactions on Mathematical Software 44–1 (2017), Article 6, 25 pages. [6, 23]

[26] I. Pinelis, A probabilistic angle inequality, MathOverflow (2018). `https://mathoverflow.net/q/298590` [30]

[27] L.M. Rios and N.V. Sahinidis, Derivative-free optimization: A review of algorithms and comparison of software implementations, Manuscript (2009). [2]

[28] R. Storn and K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optimization 11 (1997), 341–359. [3]

[29] P.J.M. Van Laarhoven and E.H.L. Aarts, Simulated annealing: theory and applications. Kluwer, Dordrecht (1987). [3]

A Appendix: Estimation of c

The following theorem was recently proved by Pinelis [26].

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

$$(p^Tp)(q^Tq) \le c_0 n (p^Tq)^2 \tag{36}$$

with probability $\ge 1/2$.

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

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

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

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

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

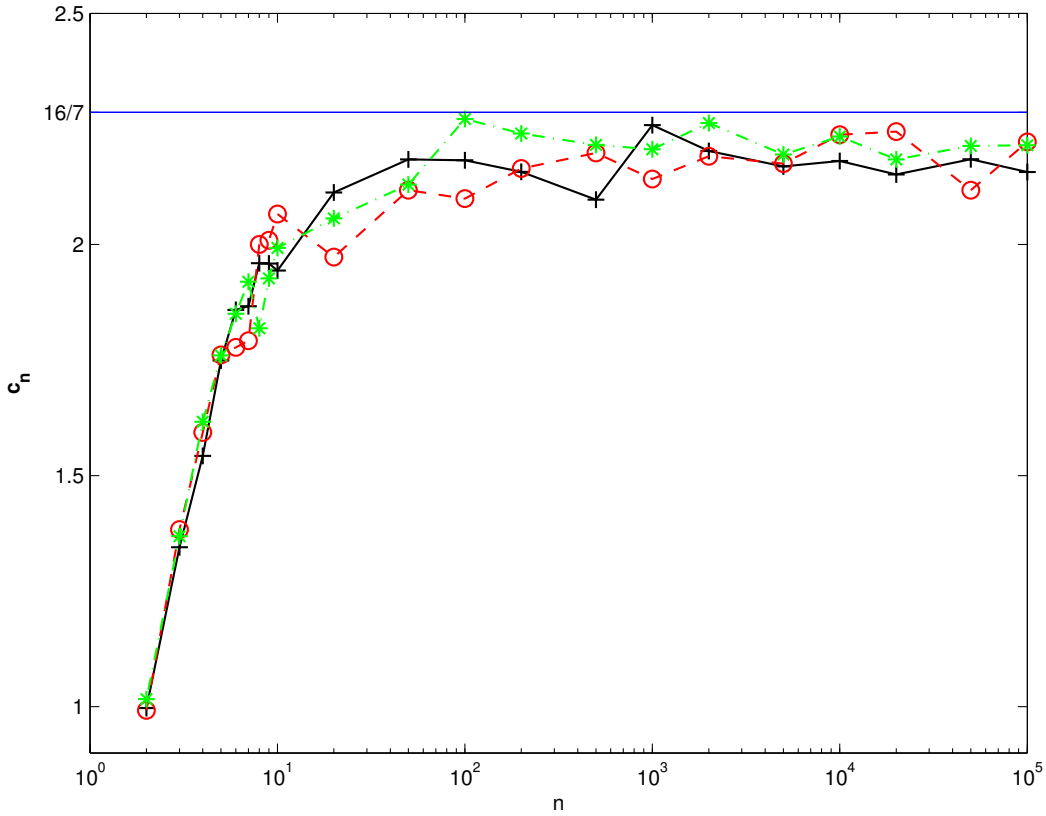


Figure 5: The plot of c_n versus the dimension n suggests that $c_0 \approx 16/7$.

A.2 Algorithm. (Estimating the Pinelis constant)

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

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

B Appendix: Flow charts

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

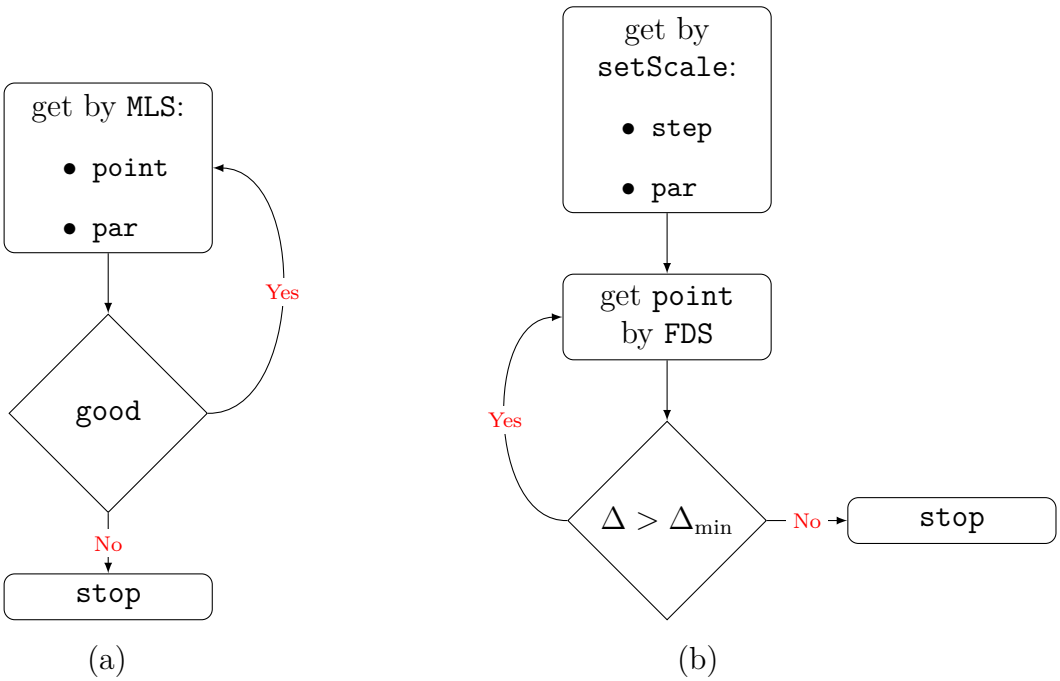


Figure 6: Flow charts for (a) `FDS`, and (b) `VSBBO`.

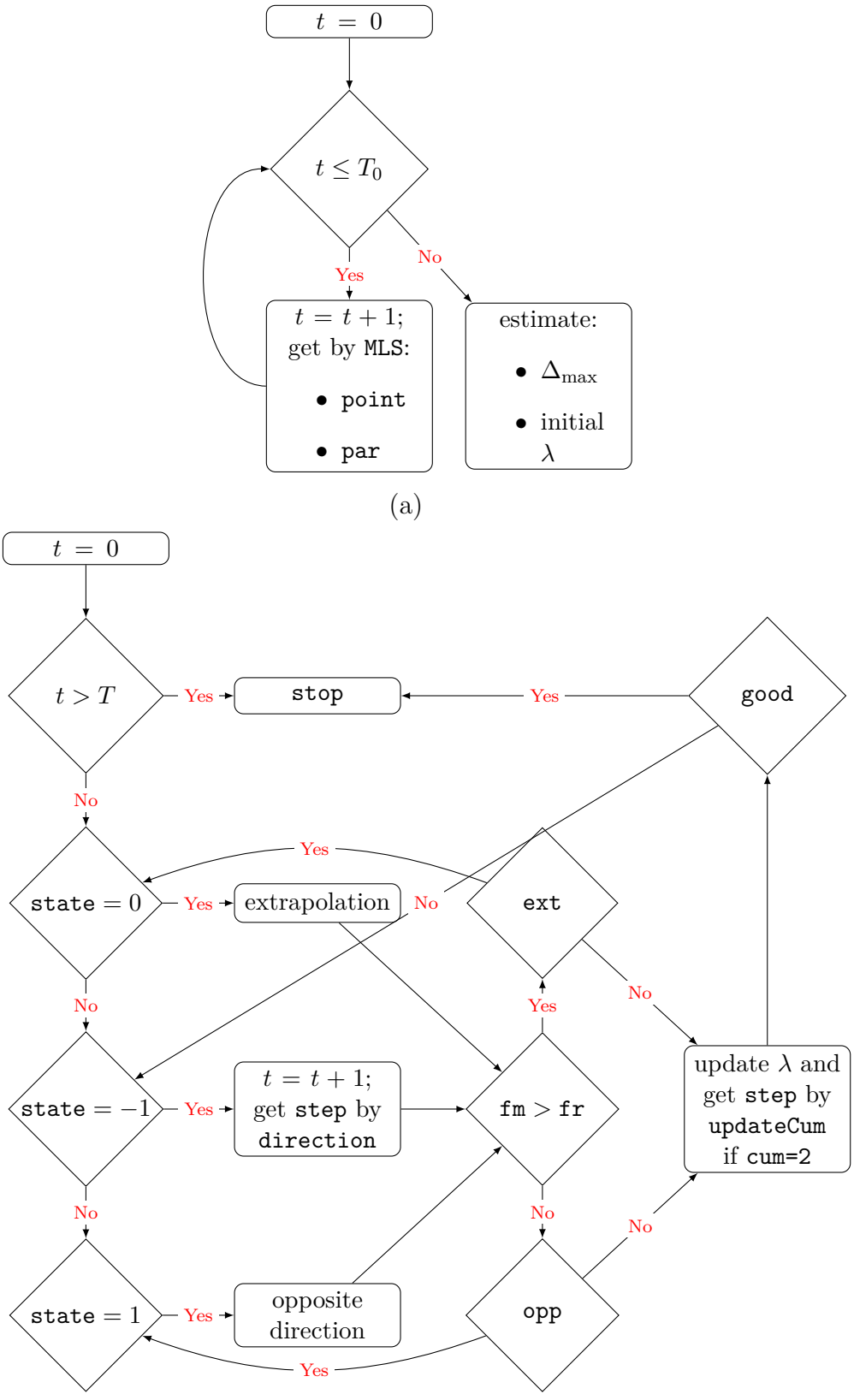


Figure 7: Flow charts for (a) `setScale`, (b) `MLS`.