Journal of Mathematical Modeling
Vol. 11, No. 2, 2023, pp. 207–228. Research Article

JMM

A mixed algorithm for smooth global optimization

Raouf Ziadi*, Abdelatif Bencherif-Madani

Laboratory of Fundamental and Numerical Mathematics (LMFN), Department of Mathematics, University Ferhat Abbas Setif 1, 19000 Setif, Algeria
Email(s): ziadi.raouf@gmail.com, lotfi_madani@yahoo.fr

Abstract. This paper presents a covering algorithm for solving bound-constrained global minimization problems with a differentiable cost function. In the proposed algorithm, we suggest to explore the feasible domain using a one-dimensional global search algorithm through a number of parametric curves that are relatively spread and simultaneously scan the search space. To accelerate the corresponding algorithm, we incorporate a multivariate quasi-Newton local search algorithm to spot the lowest regions. The proposed algorithm converges in a finite number of iterations to an ε-approximation of the global minimum. The performance of the algorithm is demonstrated through numerical experiments on some typical test functions.

Keywords: Global optimization, Alienor dimensionality reduction technique, $\alpha-$*dense* curve, one-dimensional global search algorithm, limited memory BFGS-B algorithm.
AMS Subject Classification 2010: 90C26, 90C30.

1 Introduction

In this paper we consider the following bound-constrained global optimization problem:

$$\min_{x\in D} f(x), \tag{P}$$

where $D=\{x\in\mathbb{R}^n | l\le x\le u\}$, $l,u\in\mathbb{R}^n$ and the objective function $f(x):\mathbb{R}^n\to\mathbb{R}$ is not necessarily convex but differentiable whose gradient function ∇f satisfies the Lipschitz condition with an a priory unknown Lipschitz constant M, $0<M<\infty$, i.e.,

$$\|\nabla f(x)-\nabla f(y)\|\le M\|x-y\|. \tag{1}$$

The problem (P) is of interest in many real-world problems, in particular in chemical and electrical applications [1, 28]. Many methods for solving problems of this kind have been proposed and are classified into deterministic and stochastic ones.

As is well known, the stochastic population-based algorithms such Spherical Search algorithm (SS) [15], Particle Swarm Optimization (PSO) [21, 24], Differential Evolution (DE) [22] and its improved variant Quantum-inspired Differential Evolution (QDE) [6], Harmony Search (HS) [11], etc. are practically the most efficient and the most used, for their simplicities and effectiveness. However, these methods have no guarantees to find a global

*Corresponding author
Received: 25 October 2022 / Revised: 21 November 2022 / Accepted: 24 November 2022
DOI: 10.22124/jmm.2022.23133.2061

optimal solution in a finite number of iterations and there are no theoretical results for their convergence to the global minimum. There are several parameters to be adjusted; if these parameters are not tuned appropriately, the solution obtained can be trapped into a local minimum. Moreover, most of them are computationally expensive. One of the reasons for this is the fact that the differentiability of the objective function is not efficiently exploited during the search. Due to the lack of guidance by a gradient during the searching process, their effectiveness is relatively inferior in terms of convergence speed for smooth problems. To enhance their performance for this class of problems, various combinations of evolutionary algorithms with gradient based local search procedures have been proposed [9,28]. On the other hand, the deterministic methods such as DIRECT algorithms [3], Branch and Bound algorithms [16] or the approach based on the introduction of an auxiliary function [20] etc., provide a theoretical guarantee of locating the global optimum within a prescribed precision ε, unfortunately these techniques suffer from computational difficulties when the dimension of the problem increases or the search space is relatively large.

In this paper, we present a deterministic covering algorithm by exploiting the first derivative's information. The proposed algorithm converges in a finite number of iterations to an ε-approximation of the global minimum. Our method is based on the use of the Alienor dimensionality reduction technique that transforms a multidimensional problem into a unidimensional one, using a space-filling curve, and then use a one-dimensional global optimization algorithm to approximate the global minimum. Despite the advantage of the latter, the large number of evaluations of f can make it inconvenient, particularly when the dimension of space is relatively high. This is mainly due to the use of a single curve that is supposed to closely approximate all points in the feasible domain D, for a good overview, see [18,25,26].

In order to overcome this drawback, we suggest to convert the problem (P) into a unidimensional one by running successively a number of $\alpha-dense$ (space-filling) curves that become progressively denser and simultaneously cover the search space. In the early phases of the algorithm, the curves are generated with a densification parameter α, which is relatively large, but they are sufficiently spread to cover a vast region, then the parameter α progressively decreases as the algorithm evolves. Throughout the generated $\alpha-dense$ curves, we explore the search space using a one-dimensional algorithm (with relatively large step-lengths) that is adapted to the current values of the first derivative and accelerates the search process.

Due to the lack of convexity, the objective function may exhibit violent variations with a large number of local minima, so we incorporate a quasi-Newton local search algorithm to speed-up our algorithm. Actually, one of the most effective quasi-Newton methods for solving large-scale bound differentiable optimization problems is the so called L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Schanno) [4] where 'B' stands for bound. It is considered as one of the most successful large-scale bound-constrained optimization methods that allows to explore promising regions with a moderate generation of evaluation points. The proposed algorithm will be called DRQN (Dimensionality-Reduction- Quasi-Newton algorithm).

We explore the search space D using the new accelerated one-dimensional global optimization search algorithm to spot the attraction zones of the local minima. When a new record obtained by the one-dimensional exploration algorithm is lower than the preceding one, it is taken care of by the L-BFGS-B local search algorithm as a starting point to select a lower region of D and make a descent towards a new local minimum. The last solution thus obtained gives a new speed up opportunity in the one-dimensional exploration algorithm (see Figures 4 - 7 below).

The advantage of our one-dimensional algorithm is that it generates points that are relatively distant in regions where f takes high values and denser in regions where f takes small values. Due to the use of the first derivative's information, the used one-dimensional algorithm generates much less points. Furthermore, it is well adapted to our case since the record obtained by the latter can be exploited by the L-BFGS-B local search algorithm to spot a new local minimum, and the last obtained record will be exploited by the proposed one-dimensional search algorithm to quicken the search for the global minimum over another curve (see the DRQN Algorithm 5.2). This procedure cannot be realized with other one-dimensional covering algorithms such as Branch and Bound type or the different extensions of Pyavskii-Shubert algorithm [8, 18], despite their efficiencies.

This paper is organized as follows. We describe the new approximated unidimensional problem and its properties in Section 2. Then, the one-dimensional exploration algorithm is presented in Section 3 and the L-BFGS-B algorithm used in the local search procedure in Section 4. In Section 5, we describe our proposed algorithm

DRQN for solving (*P*) and we prove its convergence to the global minimum. Numerical results are reported and some conclusions drawn.

2 The approximated unidimensional problem and its properties

We start this section by giving some notations, then we describe the new approximated unidimensional problem and its properties. Denote by f^* the global minimum of $f(\cdot)$ in $D \subset \mathbb{R}^n$, i.e., $\min_D f = f^* \in \mathbb{R}$, x^* is a global minimizer of the problem (*P*) and f_ε denotes the approximated global minimum. The $\nabla f(x)$ designates the gradient of f at the point x, μ stands for the Lebesgue measure on $\mathbb{R}^n$ and $d(\cdot,\cdot)$ for the Euclidean distance. For $x \in \mathbb{R}^n$, x^T stands for the transpose of x. $\|x\|$ is the Euclidean norm of x and $x^T y$ is the scalar product of x and y.

The Alienor dimensionality reduction technique has been elaborated at the beginning of the eighties by Cherruault and Guillez [5]. The basic idea is to use a parametric transformation allowing to simplify the above problem into a problem depending on a single variable by means of a continuous parametric $\alpha - dense$ curve.

For solving the initial problem (*P*), we first associate an $\alpha - dense$ search curve φ_α in D defined by:

$$\begin{aligned} \varphi_\alpha : [0, T_\alpha] &\rightarrow D \\ t &\rightarrow (\varphi_1^{(\alpha)}(t), \ldots, \varphi_n^{(\alpha)}(t)), \end{aligned}$$

where the variable t is termed the search variable. The minimization of the initial problem (*P*) is then approximated by a problem (*P'*) depending on the single variable t:

$$\min_{t \in [0, T_\alpha]} f_\varphi(t), \tag{P'}$$

where

$$f_\varphi(t) = f(\varphi_\alpha(t)) = f(\varphi_1^{(\alpha)}(t), \ldots, \varphi_n^{(\alpha)}(t)).$$

In other words, the objective function f which depends on several variables is approximated by a function of a single variable f_φ and the search space will be the interval $[0, T_\alpha]$.

Before describing the properties of the compound approximated function f_φ, let us first recall some definitions that will be needed below.

Definition 1. *We say that a subset S of D ($D \subset \mathbb{R}^n$) is $\alpha - dense$ in D, if for all $x \in D$, there exists a point $x' \in S$ such that $d(x, x') \leq \alpha$.*

Definition 2. *A curve $\varphi_\alpha : [0, T_\alpha] \rightarrow D, T_\alpha > 0$, is called $\alpha - dense$ in D, if for all $x \in D$, there exists $t \in [0, T_\alpha]$ such that $d(x, \varphi_\alpha(t)) \leq \alpha$, where $\varphi_\alpha(t) = (\varphi_1^{(\alpha)}(t), \ldots, \varphi_n^{(\alpha)}(t))$.*

The following result, established by Ziadi et al. [25], gives a method to generate an $\alpha - dense$ curve.

Theorem 1. *Let $\varphi_\alpha(t) = (\varphi_1^{(\alpha)}(t), \ldots, \varphi_n^{(\alpha)}(t)) : [0, T_\alpha] \rightarrow \prod_{i=1}^{n} [l_i, u_i]$ be a continuous function and let $\theta_1, \theta_2, \ldots, \theta_{n-1}, \alpha$ be strictly positive numbers such that:*

(a) $\varphi_n^{(\alpha)}$ is surjective;

(b) For any $i = 1, 2, \ldots, n-1$, $\varphi_i^{(\alpha)}$ reaches its bounds l_i and u_i in every closed interval of length θ_i;

(c) For any $i = 1, 2, \ldots, n-1$ and for any interval I of $[0, T_\alpha]$, we have

$$\mu(I) < \theta_i \Longrightarrow \mu\left(\varphi_{i+1}^{(\alpha)}(I)\right) < \frac{\alpha}{\sqrt{n-1}},$$

where $\mu(\cdot)$ is Lebesgue measure. Then, for $t \in [0, T_\alpha]$, the curve $\varphi_\alpha(t)$ is $\alpha - dense$ in $\prod_{i=1}^{n} [l_i, u_i]$.

Based on the above result, we give here an example of an $\alpha-dense$ curve that will be used in our numerical study.

Example 1. Consider the function $\varphi_\alpha(t):[0,\frac{\pi}{\theta_n}]\to D$ such that

$$\begin{aligned}
\varphi_1^{(\alpha)}(t) &= \frac{u_1+l_1}{2}-\frac{u_1-l_1}{2}\cos(\theta_1 t),\\
\varphi_2^{(\alpha)}(t) &= \frac{u_2+l_2}{2}-\frac{u_2-l_2}{2}\cos(\theta_2 t),\\
&\vdots\\
\varphi_n^{(\alpha)}(t) &= \frac{u_n+l_n}{2}-\frac{u_n-l_n}{2}\cos(\theta_n t),
\end{aligned}$$

where $\alpha>0$ and $\theta_1,\theta_2,...,\theta_n$ are parameters given by

$$\begin{aligned}
\theta_1 &= 1,\\
\theta_2 &= \frac{\alpha}{\pi(|l_2|+|u_2|)},\\
\theta_3 &= \frac{\alpha^2}{\pi^2(|l_2|+|u_2|)(|l_3|+|u_3|)},\\
&\vdots\\
\theta_n &= \frac{\alpha^{n-1}}{\pi^{n-1}(|l_2|+|u_2|)(|l_3|+|u_3|)\dots(|l_{n-1}|+|u_{n-1}|)}.
\end{aligned}$$

By Theorem 1, the parametrized curve $\varphi_\alpha(t)=(\varphi_1^{(\alpha)}(t),\dots,\varphi_n^{(\alpha)}(t))$ is $\alpha-dense$ in D. On the other hand, it is easy to show that the function φ_α is Lipschitzian with constant

$$\mathscr{L}_\varphi=\frac{1}{2}\left(\sum_{i=1}^{n}\theta_i^2(u_i-l_i)^2\right)^{\frac{1}{2}},$$

and by a straightforward calculation, the Lipschitz constant of its derivative $\frac{d\varphi_\alpha}{dt}$ is

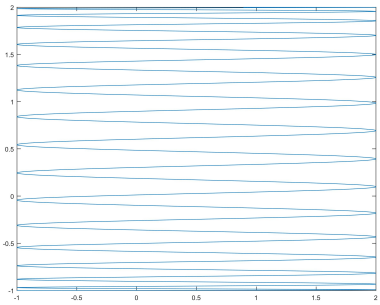
$$\mathscr{M}_\varphi=\frac{1}{2}\left(\sum_{i=1}^{n}\theta_i^4(u_i-l_i)^2\right)^{\frac{1}{2}}.$$

Figures 1 and 2 represent a densification of the square $[-1,2]^2$ and the cube $[-1,2]^3$ by the support of the given curve with different α's.

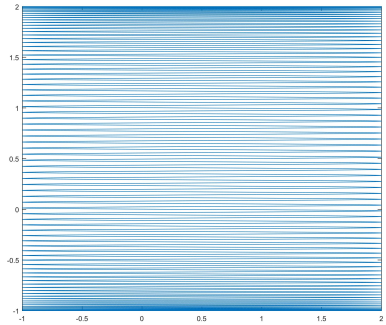
2.1 Properties of the approximated unidimensional problem (P')

Since the objective function f of the initial problem (P) is continuously differentiable, it follows that the compound univariate function f_φ is continuously differentiable and its derivative function satisfies the following Lipschitz condition with constant $\mathscr{M}=\mathscr{L}_\varphi^2\cdot M+L\cdot\mathscr{M}_\varphi$, where L is the Lipschitz constant of the objective function f, i.e.

$$\forall t_1,t_2\in\left[0,\frac{\pi}{\theta_n}\right],\left|\frac{df_\varphi}{dt}(t_1)-\frac{df_\varphi}{dt}(t_2)\right|\leq\mathscr{M}|t_1-t_2|. \tag{2}$$

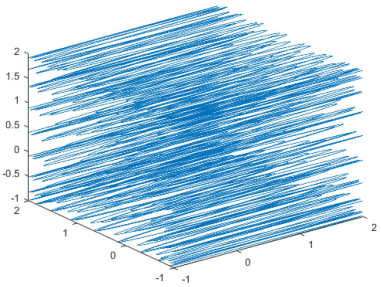


a: Support of the curve with $\alpha = 0.3$.

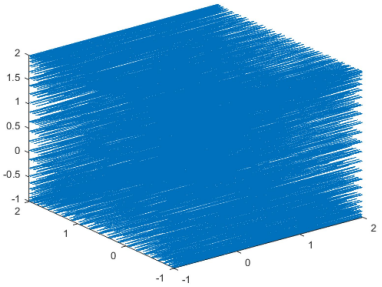


b: Support of the curve with $\alpha = 0.1$.

Figure 1: Support of the curve in the bidimensional case.



a: Support of the curve with $\alpha = 0.5$.



b: Support of the curve with $\alpha = 0.2$.

Figure 2: Support of the curve in the three-dimensional case.

Also, for all $t_1, t_2 \in \left[0, \frac{\pi}{\theta_n}\right]$, we have

$$\begin{aligned}
\left|\frac{df_\varphi}{dt}(t_1) - \frac{df_\varphi}{dt}(t_2)\right| &= \left|\nabla f(\varphi_\alpha(t_1))^T \cdot \frac{d\varphi_\alpha}{dt}(t_1) - \nabla f(\varphi_\alpha(t_2))^T \cdot \frac{d\varphi_\alpha}{dt}(t_2)\right| \\
&\leq \left\|\frac{d\varphi_\alpha}{dt}(t_1)\right\| \cdot \|\nabla f(\varphi_\alpha(t_1)) - \nabla f(\varphi_\alpha(t_2))\| + \|\nabla f(\varphi_\alpha(t_2))\| \cdot \left\|\frac{d\varphi_\alpha}{dt}(t_1) - \frac{d\varphi_\alpha}{dt}(t_2)\right\| \\
&\leq \mathscr{L}_\varphi \cdot M \|\varphi_\alpha(t_1) - \varphi_\alpha(t_2)\| + L \cdot \mathscr{M}_\varphi |t_1 - t_2| \\
&\leq \mathscr{L}_\varphi^2 \cdot M |t_1 - t_2| + L \cdot \mathscr{M}_\varphi |t_1 - t_2|.
\end{aligned}$$

It suffices to take $\mathscr{M} = \mathscr{L}_\varphi^2 \cdot M + L \cdot \mathscr{M}_\varphi$ as Lipschitz constant for $\frac{df_\varphi}{dt}$.

3 The one-dimensional search algorithm

In the literature there exist several methods for minimizing a univariate function over an interval $[a,b]$ and whose first derivative satisfies a Lipschitz condition. Most of these methods rely on the use of an a priory known value of the Lipschitz constant or an estimate of it during the search process. The majority of the proposed approaches aim to construct support functions that are close to the objective function [8, 12]. These approaches, by their very constructions, do not consider exploiting new information linked to the multidimensional local search. The idea we propose here is to use an algorithm which iteratively generates points following the scheme

$$t_{k+1} = t_k + r_k,$$

where r_k is the step-length which depends on the evaluation of t_k, on the record obtained at the step k and on the desired accuracy.

In the case where the function to be minimized has a Lipschitz gradient, Evtushenko [7] suggested a covering algorithm on an interval $[a,b]$. Since our approximate function f_φ has a Lipschitz gradient, by the Lagrange formula, for all $t_1,t_2 \in [a,b]$, we have

$$\begin{aligned} f_\varphi(t_2) &= f_\varphi(t_1) + \int_0^1 \frac{df_\varphi}{dt}(t_1 + t(t_2-t_1)) \cdot (t_2-t_1)dt \\ &= f_\varphi(t_1) + \int_0^1 \left(\frac{df_\varphi}{dt}\big(t_1 + t(t_2-t_1)\big) - \frac{df_\varphi}{dt}(t_1) \right) \cdot (t_2-t_1)dt + \frac{df_\varphi}{dt}(t_1) \cdot (t_2-t_1). \end{aligned}$$

On the other hand, we have for all $0 \le t \le 1$

$$\begin{aligned} \left(\frac{df_\varphi}{dt}\big(t_1 + t(t_2-t_1)\big) - \frac{df_\varphi}{dt}(t_1) \right) \cdot (t_2-t_1) &\ge -\left| \frac{df_\varphi}{dt}(t_1 + t\,(t_2-t_1)) - \frac{df_\varphi}{dt}(t_1) \right| . \left| t_2-t_1 \right| \\ &\ge -\mathscr{M}\, t\,(t_2-t_1)^2. \end{aligned}$$

Hence

$$\begin{aligned} f_\varphi(t_2) &\ge f_\varphi(t_1) - \mathscr{M}(t_2-t_1)^2 \int_0^1 t\,dt + \frac{df_\varphi}{dt}(t_1) \cdot (t_2-t_1) \\ &\ge f_\varphi(t_1) + \frac{df_\varphi}{dt}(t_1) \cdot (t_2-t_1) - \frac{\mathscr{M}}{2}(t_2-t_1)^2, \end{aligned} \tag{3}$$

from which

$$f_\varphi(t_1) + \frac{df_\varphi}{dt}(t_1) \cdot (t_2-t_1) + \frac{\mathscr{M}}{2}(t_2-t_1)^2 \ge f_\varphi(t_2) \ge f_\varphi(t_1) + \frac{df_\varphi}{dt}(t_1) \cdot (t_2-t_1) - \frac{\mathscr{M}}{2}(t_2-t_1)^2. \tag{4}$$

Let $\{t_j\}_{1\le j\le k}$ be the sequence of feasible points obtained when evaluating f_φ. It easy to check that if for a certain $\tilde{t} \in [a,b]$, there exists a $j \in \{1,...,k\}$ such that

$$\frac{\mathscr{M}}{2}(\tilde{t}-t_j)^2 - \frac{df_\varphi}{dt}(t_j) \cdot (\tilde{t}-t_j) \le f_\varphi(t_j) + \varepsilon - R_k^*,$$

then

$$R_k^* - \varepsilon \le f_\varphi(\tilde{t}),$$

where $R_k^* = \min\limits_{j=1,...,k} f_\varphi(t_j)$ is the record value obtained up to the step k. Therefore to have the possibility of reducing the feasible region, it is necessary to exclude the intervals with centres

$$c_j = t_j + \frac{1}{\mathscr{M}} \frac{df_\varphi}{dt}(t_j),$$

and width

$$r_j = \frac{1}{\mathscr{M}} \left(\left(\frac{df_\varphi}{dt}(t_j) \right)^2 + 2\mathscr{M}\big(f_\varphi(t_j) - R_k^* + \varepsilon\big) \right)^{\frac{1}{2}}.$$

In the Evtushenko algorithm, the evaluation points $\{t_k\}$ are generated following the scheme

$$\begin{cases} t_1 = a + \frac{\varepsilon}{\mathscr{M}} \\ t_{k+1} = t_k + \frac{1}{\mathscr{M}} \frac{df_\varphi}{dt}(t_k) + \frac{1}{\mathscr{M}} \left(\left(\frac{df_\varphi}{dt}(t_k) \right)^2 + 2\mathscr{M}\big(f_\varphi(t_k) - R_k^* + \varepsilon\big) \right)^{\frac{1}{2}} + \frac{\varepsilon}{\mathscr{M}}, \end{cases}$$

where the record value R_k^* is updated at each step k and the search process stops when a point $t_k > b$.

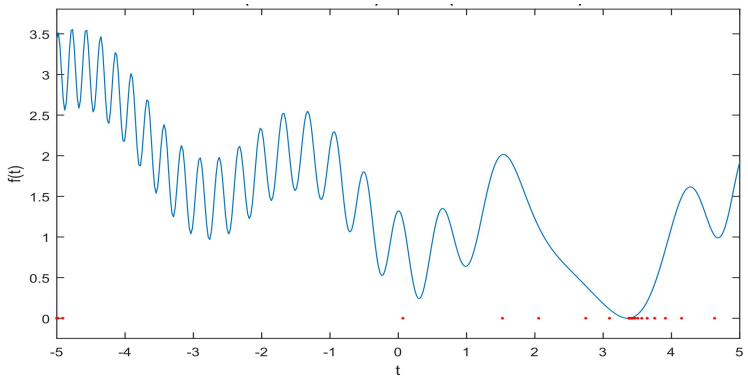


Figure 3: Trial points produced by Evtushenko's algorithm while minimizing the plotted function.

Remark 1. *For an ε-optimal solution of problem (P) in a finite number of iterations, the mixed DRQN algorithm must localize the global minimum of the approximated function f_φ, say f_φ^*, over $[0,\pi/\theta_n]$ with an accuracy $\varepsilon/2$ and the difference between f_φ^* and the global minimum f^* has to be $\varepsilon/2$ as well (see the proof of Theorem 2 below). Then, the exploration points $\{t_k\}$ are generated following the modified scheme*

$$\begin{cases} t_1 = \sqrt{\frac{\varepsilon}{\mathscr{M}}}, \\ t_{k+1} = t_k + \frac{1}{\mathscr{M}}\frac{df_\varphi}{dt}(t_k) + \frac{1}{\mathscr{M}}\left(\left(\frac{df_\varphi}{dt}(t_k)\right)^2 + 2\mathscr{M}\left(f_\varphi(t_k) - R_k^* + \frac{\varepsilon}{2}\right)\right)^{\frac{1}{2}} + \sqrt{\frac{\varepsilon}{\mathscr{M}}}, \end{cases} \tag{5}$$

and the search process stops when a point $t_k > \pi/\theta_n$.

The implementation of this algorithm has the advantage of not using complicated auxiliary calculations such as constructing minorant functions, which considerably reduces the calculation time. This choice avoids missing the global minimum of f_φ; moreover, the record R_k^* is exploited to quicken the search. To illustrate the effect of the acceleration, Figure 3 represents the points generated following the Evtushenko scheme (5) for minimizing the plotted function over the interval $[-5,5]$.

4 The L-BFGS-B local search algorithm

The L-BFGS-B algorithm (Limited Memory BFGS for Bound Constrained Optimization) [4] is a quasi-Newton type algorithm, it is an extension of L-BFGS algorithm (limited memory BFGS) to handle simple bounds. Due to its ability to deal with bounds on the variables, it is considered as one of the most successful large-scale bound-constrained optimization methods. It generates a sequence of points $\{x_j\}_{j\geq 0} \subset \mathbb{R}^n$ starting from an initial point $x_0 \in \mathbb{R}^n$ following the procedure

$$x_{j+1} = x_j + \lambda_j d_j, \tag{6}$$

where $\lambda_j > 0$ is a step-length which is determined by a line search procedure (usually chosen in such a way that it satisfies the Wolfe line search conditions) to ensure a sufficient decrease of f and d_j (the descent direction) is of the form

$$d_j = -H_j \nabla f(x_j), \tag{7}$$

where H_j is the inverse Hessian approximation matrix updated by the following formula

$$H_{j+1} = V_j^T H_j V_j + \rho_k s_j s_j^T, \tag{8}$$

with

$$y_j = \nabla f(x_{j+1}) - \nabla f(x_j),\ s_j = x_{j+1} - x_j,\ \rho_j = \frac{1}{y_j^T s_j},\ V_j = I - \rho_j y_j s_j^T.$$

The L-BFGS-B algorithm is based on the gradient projection method and uses a limited memory matrix H to approximate the inverse Hessian of the objective function by using the last $\tilde{m}$ (typically $\tilde{m} = 5$) correction pairs $\{s_i, y_i\}|_{i \in j-\tilde{m},\ldots,j-1}$. For a given iteration j, the matrix H_j that approximates the inverse Hessian at a point x_j is updated and the objective function is approximated by a quadratic model as

$$q_j(x) = f(x_j) + \nabla f(x_j)^T (x - x_j) + \frac{1}{2}(x - x_j)^T H_j (x - x_j).$$

At each iteration, the L-BFGS-B algorithm minimizes $q_j(x)$ subject to D, using the gradient projection strategy to determine a set of active constraints, followed by a minimization of $q_j(x)$ regarding the active bounds as equality constraints. The computation for the generalized Cauchy point and the subspace minimization are the most crucial phases at each iteration j (for more details see [4]). The objective of the Cauchy point computation is to minimize the quadratic approximation of the objective function $q_j(x)$, starting from the current point x_j, on the path defined by the projection of the steepest descent direction on the feasible domain. After the Cauchy point x^c is obtained, the quadratic function $q_j(x)$ is minimized over the free variables subject to their lower and upper bounds, i.e. the variables that are identified as inside the feasible design space, and then backtracked into the feasible design space to obtain $\tilde{x}$. The new search direction is computed as $d_j = \tilde{x}_j - x_j$ and a step-length λ_j is determined in such a way that it satisfies the strong-Wolfe conditions (*SW*) to compute the new design variable x_{j+1}

$$\begin{aligned} f(x_j + \lambda_j d_j) - f(x_j) &\leq c_1 \lambda_j \nabla f(x_j)^T d_j, \\ |\nabla f(x_j + \lambda_j d_j)^T d_j| &\leq c_2 |\nabla f(x_j)^T d_j|, \end{aligned} \qquad (SW)$$

where $0 < c_1 < 1/2$ and $c_1 < c_2 < 1$. The matrix H_{j+1} is then computed based on the new point x_{j+1} using the L-BFGS update formula and a new iteration is started. The algorithm stops when, for a point x_j, the norm of the projected gradient (in the sup-norm sense) onto the feasible design space is small, i.e $\|P_D(x_j - \nabla f(x_j)) - x_j\|_\infty \simeq 0$. Recently, new limited memory BFGS algorithms have been proposed for optimization problems, see [2, 14].

5 The proposed DRQN algorithm

5.1 Algorithm description

As we mentioned earlier, the principle behind our algorithm is to generate consecutively a number of $\alpha - dense$ curves that are relatively spread and cover simultaneously the feasible domain using relatively large step-lengths. In the earlier phases of the algorithm, the curves are generated with a densification parameter α which is relatively large but are sufficiently spread out to cover a vast region; the parameter α then decreases progressively with the evolution of the algorithm. The domain is explored through the generated $\alpha - dense$ curves using the one-dimensional search algorithm which localizes the local minima attraction zones. If a new record has thus been registered, the L-BFGS-B local search algorithm uses this record to select the lowest sub-region of D so to descend to a new local minimum. The last solution thus obtained is exploited by the one-dimensional exploration algorithm to quicken the search process through the generated $\alpha - dense$ curves (see Figures 4 - 7).

As is well known in Lipschitzian one-dimensional global optimization, when the Lipschitz constant (or an estimate of it) is relatively large, an important number of evaluation points is generated because the step-length becomes very small. To overcome this drawback, the explicit use of the Lipschitz constant is here circumvented. The idea is to use a sequence of controls over the gradient ∇f. That is, we find two increasing sequences $\{M_j\}$ and $\{L_j\}$ of positive constants that control the growth of M and L. Clearly there exists $\tilde{j} \in \mathbb{N}^*$ such that $M_{\tilde{j}} > M, L_{\tilde{j}} > L$ and for all $x, y \in D$ we have

$$\|\nabla f(x)-\nabla f(y)\| \leq M_{\tilde{j}}\|x-y\|, \tag{9}$$

and consequently, for all $t_1, t_2 \in [0, T_\alpha]$ we have

$$\left|\frac{df_\varphi}{dt}(t_1)-\frac{df_\varphi}{dt}(t_2)\right| \leq \mathscr{M}_{\tilde{j}}|t_1-t_2|. \tag{10}$$

This condition theoretically justifies the convergence of the algorithm in a finite number of iterations given the desired accuracy $\varepsilon > 0$ (see Theorem 2 below). At the step j, a well spread curve is generated which is $\alpha_j - dense$ depending on $\mathscr{M}_j$. The one-dimensional algorithm goes on to explore the objective function over this curve, it has also a variable step-length depending on M_j as well and also on the record obtained during the $j-1$ preceding steps. The local search intervenes when a new record point detected by the one-dimensional exploration algorithm is lower than the older one.

The first few terms of the sequences M_j and L_j are taken sufficiently small during the first phases of the algorithm so that the one-dimensional exploration algorithm can use relatively large step-lengths; but during the j iterations we increase the parameters M_j and L_j by setting $M_j = \xi \cdot M_{j-1}$ and $L_j = \xi \cdot L_{j-1}, \xi > 1$ to have a denser curve with smaller one-dimensional step-lengths. After a certain number of iterations, the set of the generated points will be somewhat dense in the regions where f takes small values and much less dense elsewhere (see Figure 3). This phenomenon is allowed by our variable exploration step-length which depends on the region where the feasible point lies and increases as f increases.

As is said in Remark 1, in order to have an ε-optimal solution of problem (P), the algorithm must localize the global minimum of the approximated function f_φ, noted f^*_φ, over $[0, \pi/\theta_n]$ with an accuracy $\varepsilon/2$ and the difference between f^*_φ and the global minimum f^* has to be $\varepsilon/2$ as well. Then from (5), at an iteration j, the search points generated by the modified one-dimensional algorithm are given as follows

$$\begin{cases} t_1 = \sqrt{\varepsilon/\mathscr{M}_j}, \\ t_{k+1} = t_k + \frac{1}{\mathscr{M}_j}\frac{df_{\varphi_j}}{dt}(t_k) + \frac{1}{\mathscr{M}_j}\left(\left(\frac{df_{\varphi_j}}{dt}(t_k)\right)^2 + 2\mathscr{M}_j\left(f_{\varphi_j}(t_k) - f_\varepsilon + \frac{\varepsilon}{2}\right)\right)^{\frac{1}{2}} + \sqrt{\frac{\varepsilon}{\mathscr{M}_j}}, \end{cases}$$

where $f_{\varphi_j}(t) = f\left(\varphi_{\alpha_j}(t)\right)$ in which φ_{α_j} is the curve of Example 1 having the densification parameter α_j, f_ε is the record value of the mixed algorithm DRQN and $\mathscr{M}_j = \mathscr{L}^2_{\varphi_j}.M_j + L_j.\mathscr{M}_{\varphi_j}$ is a term of an increasing sequence covering $\mathscr{M}$, see the algorithm below. The algorithm stops when we obtain a curve whose parameter densification α_j is lower than a threshold $\alpha_{\min}$.

5.2 The DRQN algorithm

Remark 2. *The first terms of the sequences $\{L_j\}_{j\in\mathbb{N}}$ and $\{M_j\}_{j\in\mathbb{N}}$ should be sufficiently small during the first steps of the algorithm, to have large enough step-lengths. The adjustment of these parameters depends essentially on the size of the search space D, a larger domain would demand smaller ξ, M_1 and L_1 for lesser evaluation points.*

To illustrate the minimization process by the DRQN algorithm, we present in Figures 4 - 7 respectively the graphs of the Bird, Drop-Wave, Modified Langerman and Schaffer 2 functions together with the work done by DRQN, through the trial points produced by the algorithm, to spot a global minimizer.

6 Convergence of the DRQN algorithm

The following theorem establishes the convergence of the DRQN algorithm without a stopping criterion.

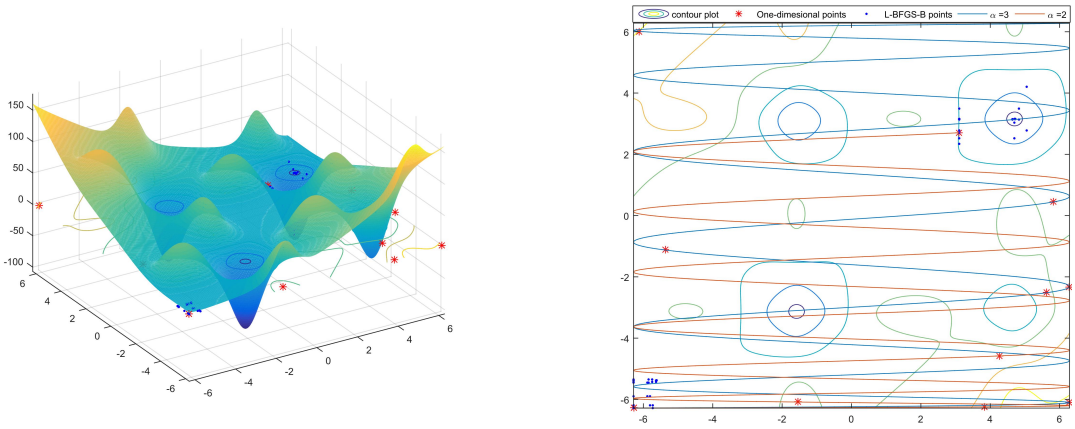


Figure 4: Minimization process of the Bird function: global minimizer $x^* = (4.7010, 3.1529)$ with global minimum $f(x^*) = -106.764537$.

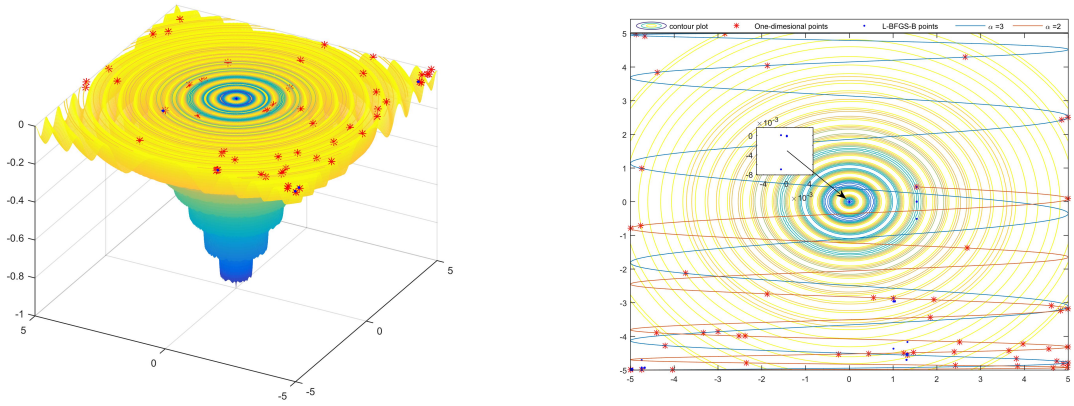


Figure 5: Minimization process of the Drop-Wave function: global minimizer $x^* = (0,0)$ with global minimum $f(x^*) = -1$.

Theorem 2. *The combined algorithm DRQN converges in a finite number of evaluation points towards the global minimum of problem (P) within the accuracy $\varepsilon > 0$.*

Proof. Let f^* be the global minimum of f on D and f_ε its approximate value obtained by the proposed algorithm. We denote by $f^*_{\varphi_j}$ the global minimum of the approximated function f_{φ_j} on $[0, T_{\alpha_j}]$.

Without a stopping criterion, it is expected that the DRQN algorithm generates a sequence of points $\{t_k\}_{k\in\mathbb{N}^*}$ that is everywhere dense in D. Indeed, as j increases, the DRQN algorithm generates the sequences $\{\alpha_j\}_{j\in\mathbb{N}^*}$, $\{M_j\}_{j\in\mathbb{N}^*}$, $\{L_j\}_{j\in\mathbb{N}^*}$ and $\{\mathscr{M}_j\}_{j\in\mathbb{N}^*}$; the sequences $\{M_j\}_{j\in\mathbb{N}^*}$ and $\{L_j\}_{j\in\mathbb{N}^*}$ are geometric with the first terms $M_1 > 0, L_1 > 0$ and comment ration $\xi > 1$ and tend to infinity, whereas the sequence $\{\alpha_j\}_{j\in\mathbb{N}^*}$ is geometric with the first term $\alpha_1 > 0$ and comment ration $0 < 1/\xi < 1$ and tends to zero. Therefore, from (9) there exists $\tilde{j} \in \mathbb{N}^*$ such that $\alpha_{\tilde{j}} \leq \sqrt{\varepsilon/M}$ and for all $x, y \in D$ we have

$$\begin{cases} \big|f(x) - f(y)\big| \leq L_{\tilde{j}}\|x-y\| \\ \|\nabla f(x) - \nabla f(y)\| \leq M_{\tilde{j}}\|x-y\| \end{cases}$$

Algorithm 1: Pseudo-code of the DRQN algorithm

Input: Objective function f, define the $\alpha-dense$ curve $\varphi_\alpha(.)$ and let $\mathscr{L}_\varphi$ and $\mathscr{M}_\varphi$ respectively the Lipschitz constants of the curve φ_α and $\frac{\varphi_\alpha}{dt}$, the accuracy of the computation of the global minimum ε, multiplicative factor $\xi > 1$, $\alpha_{\min}$ a densification parameter of the last generated curve to stop the algorithm.

Output: Global minimum f_ε and global minimizer x_ε

.

// Initialization

- Set $j = 1$ and initialize L_1, M_1
- Set $\alpha_1 = \sqrt{\varepsilon/M_1}$
- Set $f_\varepsilon = \min\{f(l), f(u)\}$ and $x_\varepsilon = \arg\min f_\varepsilon$, where l and u are search-space limits

// Main loop

while $\alpha_j > \alpha_{\min}$ **do**

- Generate $\alpha_j - dense$ curve φ_{α_j}, and let $\mathscr{L}_{\varphi_j}$ and $\mathscr{M}_{\varphi_j}$ be their Lipschitz constants
- Set $\mathscr{M}_j = \mathscr{L}^2_{\varphi_j}.M_j + L_j.\mathscr{M}_{\varphi_j}$
- Set $k = 1$ and $t_1 = \sqrt{\varepsilon/\mathscr{M}_j}$
- **while** $t_k < T_j$ **do**
  - Evaluate $f_{\varphi_j}(t_k)$
  - **if** $f_{\varphi_j}(t_k) < f_\varepsilon$ **then**
    // Apply the L-BFGS-B local search algorithm, starting from $x_0 = \varphi_{\alpha_j}(t_k)$ and let $\tilde{x}$ the obtained local minimizer
    - $(\tilde{x}, f(\tilde{x}))$=L-BFGS-B $(\varphi_{\alpha_j}(t_k))$
    - Put $x_\varepsilon = \tilde{x}, f_\varepsilon = f(x_\varepsilon)$ // Update the record value f_ε
  - Set $t_{k+1} = t_k + \frac{1}{\mathscr{M}_j}\frac{df_{\varphi_j}}{dt}(t_k) + \frac{1}{\mathscr{M}_j}\left(\left(\frac{df_{\varphi_j}}{dt}(t_k)\right)^2 + 2\mathscr{M}_j\left(f_{\varphi_j}(t_k) - f_\varepsilon + \frac{\varepsilon}{2}\right)\right)^{\frac{1}{2}} + \sqrt{\frac{\varepsilon}{\mathscr{M}_j}}$
  - $k = k+1$
- Set $M_{j+1} = \xi \cdot M_j$, $L_{j+1} = \xi \cdot L_j$ // Increase the Lipschitz constants of f and its derivative
- Set $\alpha_{j+1} = \alpha_j/\xi$ // Decrease the parameter densification α to generate a denser curve
- $j = j+1$.

// Get the solution

- **return** $(x_\varepsilon, f_\varepsilon)$

and consequently for all $t_1, t_2 \in [0, T_{\alpha_{\tilde{j}}}]$ we have

$$\left|\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_1) - \frac{df_{\varphi_{\tilde{j}}}}{dt}(t_2)\right| \leq \mathscr{M}_{\tilde{j}}|t_1 - t_2|.$$

Since the algorithm must localize the global minimum of the approximated function $f_{\varphi_{\tilde{j}}}$, noted $f^*_{\varphi_{\tilde{j}}}$, on $[0, T_{\alpha_{\tilde{j}}}]$ with accuracy $\varepsilon/2$ and the difference between $f^*_{\varphi_{\tilde{j}}}$ and the global minimum f^* has to be $\varepsilon/2$ as well, we now distinguish two parts (A) and (B) in the proof.

- (A): At the step $\tilde{j}$, the algorithm generates the curve $\varphi_{\alpha_{\tilde{j}}}(t) : t \in [0, T_{\alpha_{\tilde{j}}}]$ in D, $\alpha_{\tilde{j}} - dense$ with $\alpha_{\tilde{j}} = \xi^{1-\tilde{j}}\alpha_1$. Let $t_1, t_2, \ldots, t_N$ be the points generated by the algorithm during this step for minimizing the composed function $f_{\varphi_{\tilde{j}}}$. The main iteration during this step is

$$t_1 = \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}, \quad t_{k+1} = t_k + r_k,$$

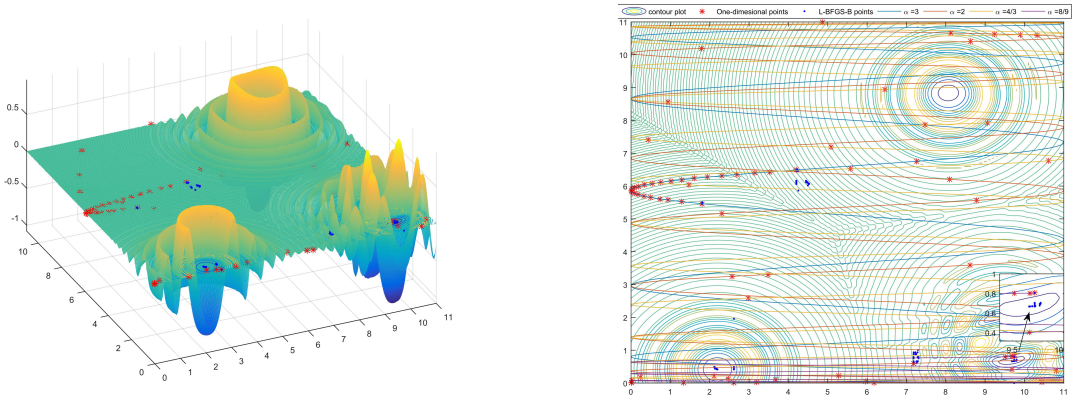


Figure 6: Minimization process of the Modified Langerman function: global minimizer $x^* = (9.68107, 0.66665)$ with global minimum $f(x^*) = -0.96500$.

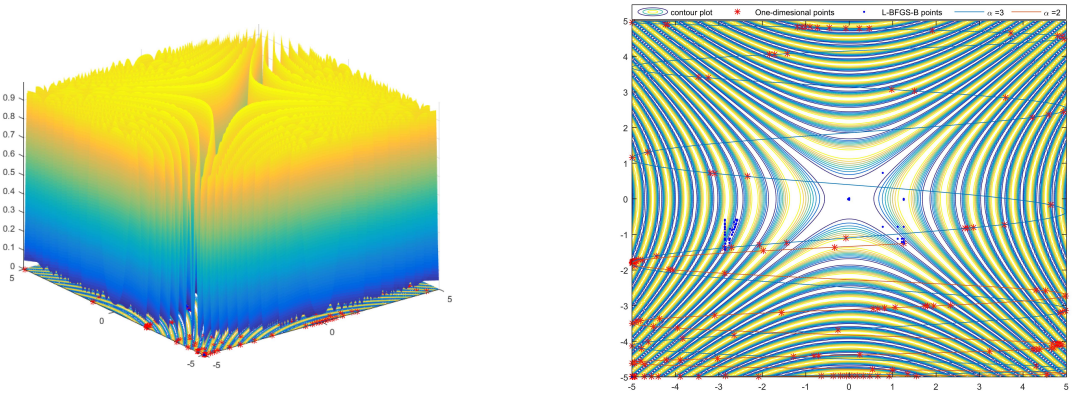


Figure 7: Minimization process of the Schaffer 2 function: global minimizer $x^* = (0,0)$ with global minimum $f(x^*) = 0$.

where

$$r_k = \frac{1}{\mathscr{M}_{\tilde{j}}}\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_k) + \frac{1}{\mathscr{M}_{\tilde{j}}}\left(\left(\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_k)\right)^2 + 2\mathscr{M}_{\tilde{j}}\left(f_{\varphi_{\tilde{j}}}(t_k) - f_\varepsilon + \varepsilon/2\right)\right)^{\frac{1}{2}} + \sqrt{\frac{\varepsilon}{\mathscr{M}_{\tilde{j}}}}.$$

Let $f^*_{\varphi_{\tilde{j}}}$ be the global minimum of the approximated function $f_{\varphi_{\tilde{j}}}$ on $[0, T_{\alpha_{\tilde{j}}}]$, there exists a $t^* \in [0, T_{\alpha_{\tilde{j}}}]$ such that $f^*_{\varphi_{\tilde{j}}} = f\left(\varphi_{\tilde{j}}(t^*)\right)$. There are overall only four possibilities depending on relation (4), let us list them in the following way.

- First case: $t^* \in \{0, T_{\alpha_{\tilde{j}}}\}$, then $f^*_{\varphi_{\tilde{j}}} = \min\left\{f_{\varphi_{\tilde{j}}}(0), f_{\varphi_{\tilde{j}}}(T_{\alpha_{\tilde{j}}})\right\} = \min\{f(l), f(u)\}$, hence $f_\varepsilon = f^*_{\varphi_{\tilde{j}}}$.

- Second case: $t^* \in (0, t_1]$ then $t_1 - t^* \leq \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}$. Since the gradient function of $f_{\varphi_{\tilde{j}}}$ is Lipschitzian, then from (4) we have

$$f_{\varphi_{\tilde{j}}}(t_1) - f_{\varphi_{\tilde{j}}}(t^*) \leq \frac{df_{\varphi_{\tilde{j}}}}{dt}(t^*) \cdot (t_1 - t^*) + \frac{\mathscr{M}_{\tilde{j}}}{2}(t_1 - t^*)^2.$$

Since t^* is the global minimum of $f_{\varphi_{\tilde{j}}}$ on $[0,T_{\alpha_{\tilde{j}}}]$, then $\frac{df_{\varphi_{\tilde{j}}}}{dt}(t^*)=0$, thus

$$f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le f_{\varphi_{\tilde{j}}}(t_1) - f_{\varphi_{\tilde{j}}}(t^*) \le \frac{\mathscr{M}_{\tilde{j}}}{2}(t_1-t^*)^2 \le \varepsilon/2.$$

- Third case: $t^* \in (t_1,t_N]$, where t_N is the last point generated by the algorithm at the step $\tilde{j}$ (with $t_{N+1} \ge T_{\alpha_{\tilde{j}}}$), then $\exists k \in 1,\dots,N-1$ such that $t_k < t^* \le t_{k+1}$, with $t_{k+1} = t_k + r_k$, we have two possibilities:

- (i): $t^* \in \left(t_k, t_k + r_k - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}\right]$. It follows that

$$t^* - t_k \le \frac{1}{\mathscr{M}_{\tilde{j}}}\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_k) + \frac{1}{\mathscr{M}_{\tilde{j}}}\left(\left(\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_k)\right)^2 + 2\mathscr{M}_{\tilde{j}}\left(f_{\varphi_{\tilde{j}}}(t_k) - f_\varepsilon + \varepsilon/2\right)\right)^{\frac{1}{2}}.$$

From relation (3), after straightforward calculations, we obtain

$$f_{\varphi_{\tilde{j}}}(t_k) - f_{\varphi_{\tilde{j}}}(t^*) \le -\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_k)\cdot(t^*-t_k) + \frac{\mathscr{M}_{\tilde{j}}}{2}(t^*-t_k)^2 \le f_{\varphi_{\tilde{j}}}(t_k) - f_\varepsilon + \varepsilon/2,$$

then $f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le \varepsilon/2$.

- (ii): $t^* \in \left(t_k + r_k - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}, t_{k+1}\right]$. It follows that $t_{k+1} - t^* \le \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}$. Since t^* is the global minimum of $f_{\varphi_{\tilde{j}}}$ on $[0,T_{\alpha_{\tilde{j}}}]$, as in the second case it follows that

$$f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le \frac{\mathscr{M}_{\tilde{j}}}{2}(t_{k+1}-t^*)^2 \le \varepsilon/2.$$

- Fourth case: $t^* \in (t_N, T_{\alpha_{\tilde{j}}})$, we have two possibilities:

- (i): $T_{\alpha_{\tilde{j}}} - t_N \le r_N - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}$. It follows that

$$t^* - t_N \le \frac{1}{\mathscr{M}_{\tilde{j}}}\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_N) + \frac{1}{\mathscr{M}_{\tilde{j}}}\left(\left(\frac{df_{\varphi_{\tilde{j}}}}{dt}(t_N)\right)^2 + 2\mathscr{M}_{\tilde{j}}\left(f_{\varphi_{\tilde{j}}}(t_N) - f_\varepsilon + \varepsilon/2\right)\right)^{\frac{1}{2}}.$$

Proceeding as in the third case-(i), we obtain $f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le \varepsilon/2$.

- (ii): $T_{\alpha_{\tilde{j}}} - t_N \ge r_N - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}$. It follows that

$$t^* \in \left(t_N, t_N + r_N - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}\right], \quad \text{or} \quad t^* \in \left(r_N - \sqrt{\varepsilon/\mathscr{M}_{\tilde{j}}}, T_{\alpha_{\tilde{j}}}\right).$$

Proceeding exactly as in the third case and keeping in mind that $T_{\alpha_{\tilde{j}}} \le t_{N+1}$ and $\varphi(T_{\alpha_{\tilde{j}}}) = u$, we also find $f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le \frac{\varepsilon}{2}$.

We deduce that in all cases, there exists $t^* \in [0,T_{\alpha_{\tilde{j}}}]$ such that $f_\varepsilon - f^*_{\varphi_{\tilde{j}}} \le \varepsilon/2$.

- (B): As the curve $\varphi_{\tilde{j}}$ is $\alpha_{\tilde{j}}-dense$ with $\alpha_{\tilde{j}} \le \sqrt{\varepsilon/M}$, there exists $\tilde{t} \in [0,T_{\alpha_{\tilde{j}}}]$ such that

$$d(\varphi_{\tilde{j}}(\tilde{t}), x^*) \le \alpha_{\tilde{j}} \le \sqrt{\varepsilon/M}.$$

Since $\nabla f(x^*) = 0$ and the gradient function of f has Lipschitz constant M, then from relation (4)

$$f(\varphi_{\tilde{j}}(\tilde{t})) - f^* \le \nabla f(x^*)^T.(\varphi_{\tilde{j}}(\tilde{t}) - x^*) + \frac{M}{2}\|\varphi_{\tilde{j}}(\tilde{t}) - x^*\|^2 \le \varepsilon/2.$$

It follows that

$$f^*_{\varphi_{\tilde{j}}} - f^* \le f(\varphi_{\tilde{j}}(\tilde{t})) - f^* \le \varepsilon/2.$$

From (A) and (B), we deduce that

$$f_\varepsilon - f^* = f_\varepsilon - f^*_{\varphi_{\tilde{j}}} + f^*_{\varphi_{\tilde{j}}} - f^* \le \varepsilon.$$

□

7 Computational experiments

In this section, we present some practical performances of the proposed method applied to a diverse set of typical test problems found in [23, 30]. All numerical experiments are implemented in the scientific software MATLAB version R2015a.

In all the experiments, the DRQN algorithm has been implemented taking $\varepsilon = 10^{-4}$, $L_1 = 10^{-4}$, $M_1 = 10^{-6}$ and $\xi = 2$ for the one-dimensional exploration algorithm, whereas for the local search algorithm L-BFGS-B, we have used the version of Byrd et al. [4], downloadable from https://github.com/bgranzow/L-BFGS-B, with the setting parameter $\tilde{m} = 5$ and $H_0 = I_n$ is set as the identity matrix (the other parameters are set to their default values). Moreover, in all the computations, the number of evaluations of the objective and gradient functions involved in the DRQN algorithm is calculated by the following formula

$$feval = Nf + n \times Ng, \tag{11}$$

where n is the dimension of the problem, Nf and Ng are respectively the number of evaluations of the objective and gradient functions f and ∇f.

To have an overall picture about the algorithm and its components, we present in Table 2 some detailed results obtained by the DRQN algorithm for finding the global minima of a certain number of test functions. This will give us a clue on the work done by each component of our algorithm. In this table, $feval$, $CPU(s)$ and num_curves are respectively the number of function evaluations, CPU times in seconds and number of generated $\alpha - dense$ curves, whereas $feval\_expl$, $feval\_loc$ and num_cal_ls are respectively the number of function evaluations generated by the one-dimensional exploration algorithm, the number of function evaluations generated by the L-BFGS-B local search algorithm and the number of calls for the L-BFGS-B local search algorithm.

In order to show the efficiency of the DRQN algorithm, we shall compare its performance with both deterministic and stochastic algorithms, by observing the number of evaluations and the time elapsed by each algorithm to obtain an approximate solution and the best results are styled in bold. In all the experiments every computation was terminated as successful when a recorded solution error satisfying

$$|f(x_k) - f^*| \leq 10^{-5}, \tag{12}$$

was reached within 5×10^5 function evaluations and whose calculation time does not exceed 100 seconds; otherwise, the computation was considered as a failure. The stopping criterion of the DRQN algorithm is replaced by the stopping condition (12) and the number of function evaluations is calculated by the formula (11).

In Table 3 the DRQN algorithm is compared with four deterministic algorithms: RTEHJ [29], MCS [13], AEGPS [27] and ACRS [3], where $feval$ and $CPU(s)$ are respectively the number of function evaluations and CPU-time in seconds. The Matlab implementations of the algorithms MCS and ACRS are respectively downloadable from https://www.mat.univie.ac.at/~neum/software/mcs/, http://www.glopt.net/softwares.html and have been used for all the experiments with default parameters setting, whereas for AEGPS and RTEHJ algorithms we have taken the same parameters as those in [27, 29].

From Table 3 it is clear that the proposed algorithm is practically more performent than the other deterministic algorithms since it has managed to be quicker in solving most problems, followed by the AEGPS algorithm. As is well known, the stochastic population-based algorithms are more efficient than the deterministic ones. Table 4 reports numerical results allowing to compare our method with seven stochastic algorithms (two of them are efficient and recent evolutionary algorithms: EO and COA) EO (Equilibrium Optimizer) [10], COA [17] (Coyote Optimization Algorithm), DE (Differential Evolution) [22], SPSO (Standard Particle Swarm Optimization) [24], HS (Harmony Search) [11] and G-CART (Classification and Regression Trees) [19] by observing the number of function evaluations and the time elapsed by each algorithm to obtain an approximate solution. The MATLAB implementations of the algorithms DE, SPSO, HS, EO, COA and G-CART are, respectively, downloadable from https://yarpiz.com/231/ypea107-differential-evolution, http://www.particleswarm.info/Programs, https://github.com/jkpir/COA, http://www.math.canterbury.ac.nz/~b.robertson/research.html, https://github.com/afshinfaramarzi/Equilibrium-Optimizer,

Table 1: Test problems.

Problem number	Dimension n	Problem name	Search region	global minimum
1	2	Schaffer 2 function	$[-10,10]^2$	0
2	2	Drop-Wave function	$[-10,10]^2$	-1
3	2	Shubert function	$[-10,10]^2$	-186.7309
4	4	Wood function	$[-30,30]^4$	0
5, 6, 7, 8, 9, 10	5, 10, 20, 30, 40, 50	Dixon and Price function	$[-30,30]^n$	0
11	2			0.01922
12	4			0.03844
13	10			0.096103
14	20	Cosine Mixture function	$[-30,30]^n$	0.192206
15	30			0.288309
16	40			0.384412
17	50			0.480515
18, 19, 20, 21, 22, 23	5, 10, 20, 30, 40, 50	Exponential function	$[-30,30]^n$	0
24, 25, 26, 27, 28, 29	4, 10, 20, 30, 40, 50	Griewank function	$[-30,30]^n$	0
30, 31, 32, 33, 34, 35	5, 10, 20, 30, 40, 50	Levy and Montalvo 1 function	$[-10,10]^n$	0
36, 37, 38, 39, 40, 41	5, 10, 20, 30, 40, 50	Levy and Montalvo 2 function	$[-10,10]^n$	0
42	2			-1.80130
43	5	Michalewicz function	$[0,\pi]^n$	-4.68765
44	8			-7.66375
45	10			-9.66015
46	2			-1.08093
47	5	Modified Langerman function	$[0,10]^n$	-0.96500
48	7			-0.51700
49	10			-0.96500
50	3		$[0,1]^3$	-3.86278
51	6	Hartamann function	$[0,1]^6$	-3.32237
52, 53, 54, 55, 56, 57, 58	5, 8, 10, 20, 30, 40, 50	Neumaier function	$[-n,n]^n$	$-n(n+4)(n-1)/6$
59, 60, 61, 62, 63, 64	5, 10, 20, 30, 40, 50	Sum Squares function	$[-30,30]^n$	0
65,66, 67, 68, 69, 70	5, 10, 20, 30, 40, 50	Zakharov function	$[-10,10]^n$	0
71, 72, 73, 74, 75, 76	4, 10, 20, 30, 40, 50	Rosenbrock function	$[-10,10]^n$	0
77, 78, 79, 80, 81, 82	5, 10, 20, 30, 40, 50	Rastrigin function	$[-30,30]^n$	0
83, 84, 85, 86, 87, 88, 89, 90	4, 8, 16, 20, 24, 28, 40, 50	Powell function	$[-30,30]^n$	0
91, 92, 93, 94, 95, 96	5, 10, 20, 30, 40, 50	Perm's function $(P)_{n,0.5}$	$[-n,n]^n$	0
97, 98, 99, 100, 101, 102	5, 10, 20, 30, 40, 50	Ackley function	$[-30,30]^n$	0
103	5			-195.8299
104	10	Styblinski-Tang function	$[-5,5]^n$	-391.6599
105	20			-783.3195
106	30			-1174.9797
107	4	Colville function	$[-10,10]^4$	0
108	4			$m=5,-10.1532$
109	4	Shekel function	$[0,10]^4$	$m=7,-10.4029$
110	4			$m=10,-10.5364$
111	2			-1
112	5			-2
113	8	Sum of different power function	$[-1,1]^n$	-4
114	10			-5
115	2			4.98151
116	5	Paviani function	$[2.001,9.99]^n$	9.73052
117	10			-45.77847
118	2			5
119	5			3413
120	8	Hyper-ellipsoid function	$[-5,5]^n$	17650828
121	10			10405071317
122	2			-1.4914
123	8	Sine envelope function	$[-10,10]^n$	-10.44047
124	10			-13.41403
125	2	Bird function	$[-2\pi,-2\pi]^2$	-106.764537

https://www.mathworks.com/matlabcentral/fileexchange/28850-harmony-search-algorithm, and have been used for all the experiments with a standard setting of associated parameters.

In these comparisons, all functions have been independently run twenty times by the stochastic algorithm under consideration; the mean number of the evaluations and the mean calculation time for each algorithm have been reported. If during the trials, a method has failed at least once, the number of failures was reported. For a given

Table 2: Performance of DRQN algorithm on some test problems.

Problem number	$feval$	$CPU(s)$	num_curves	$feval\_expl$	$feval\_loc$	num_cal_ls
5	317	0.1697	3	89	228	4
12	93	0.1014	2	30	63	2
17	1129	0.3068	3	432	697	3
20	11420	2.5007	4	8094	3326	8
31	1706	0.9028	3	101	1605	8
35	38368	17.8303	4	1205	37163	12
49	609583	70.6800	11	600625	8958	10
69	1001	0.2386	3	30	971	5
73	196443	33.2759	8	190075	6368	8
87	1914	0.4627	4	399	1515	4
90	1609	0.5491	3	132	1477	3
95	106487	28.3595	7	96095	10392	10
100	18285	3.7586	5	5309	12976	7

problem, the average number has not been calculated for the algorithm with at least 5 failures in 20 executions; for failures less than 5, the average of the twenty trials is calculated and, for each failure, the maximal number of evaluation points or the maximal CPU-time (depending on the failure case) is associated. The mean value is then displayed with an indication of the number of failures between parentheses.

From Table 4, the numerical results indicate that the proposed algorithm exhibits a better performance. In particular, the DRQN algorithm is fastest for about 41% of the test problems and it solves about 88% of the test problems successfully, followed by DE, EO and COA since they solve respectively about 79%, 75% and 74% of the test problems successfully, SPSO has the fifth best performance with 69% of the test problems; whereas HS and G-CART, solve respectively about 47% and 39% of the test problems successfully. These outcomes demonstrate that the DRQN is competitive and converges quickly towards the global minimum in the majority of the test problems. This is likely due to the fact that the actual adaptive exploitation of the gradient leads towards lower regions with a moderate number of generated points. However, the proposed algorithm produces a somewhat large number of evaluations in the case where the attraction region of the global minimizer is very narrow. All things considered, numerical comparisons indicate that our proposed method seems to be promising and competitive in practice.

8 Conclusions and future research

This paper deals with bound-constrained and non-convex global optimization problems where the objective function has a Lipschitz gradient. The proposed method is a combination of two procedures; the main procedure is to scan the feasible domain using a one-dimensional global search algorithm through a number of $\alpha - dense$ curves that are relatively spread and become progressively denser, thus covering simultaneously the search space. In order to quicken the exploration procedure, we have incorporated a quasi-Newton local search algorithm to spot the lowest regions. The proposed algorithm converges in a finite number of iterations to an ε-approximation of the global minimum. Preliminary numerical experiments indicate that the algorithm is promising and competitive in practice.

Concerning further developments, other one-dimensional covering algorithms could be exploited; another track is to incorporate local search procedures which are better performing than the L-BFGS-B algorithm.

References

[1] N. Andrei, *Nonlinear Optimization Applications Using the GAMS Technology*, New York: Springer, 2013.

Table 3: Number of function evaluations and CPU-time required by the DRQN algorithm and the other deterministic methods to reach the global minimum.

Problem number	DRQN $feval/CPU(s)$	RTEHJ $feval/CPU(s)$	MCS $feval/CPU(s)$	AEGPS $feval/CPU(s)$	ACRS $feval/CPU(s)$
1	301/0.2409	443/**0.0911**	**208**/1.4444	1609/0.9766	6529/1.7160
2	320/0.1812	**121/0.0662**	212/1.5526	137269/20.6520	50232/9.0013
3	**61**/0.1417	1102/**0.0862**	205/1.3965	681/0.4950	55692/8.5021
4	**465/0.1964**	16764/0.9173	822/2.1382	62519/9.7248	fail
5	**317/0.1697**	25677/1.2402	fail	1534/ 0.5240	fail
6	27929/5.3070	**26733/1.4802**	fail	47021/8.8143	fail
7	116183/24.9951	fail	fail	**53392/12.1933**	fail
8	**134738/32.2898**	fail	fail	187396/36.3999	fail
9	fail	fail	fail	fail	fail
10	fail	fail	fail	fail	fail
11	**70**/0.0971	335/**0.0233**	222/0.2931	7157/1.9032	7157/1.9032
12	**93/0.1014**	2292/0.4042	292/1.6050	594/0.3540	8409/2.3244
13	**127/0.1110**	284040/34.8516	860/2.6876	3390/ 0.8049	9823/3.4008
14	**198/0.1266**	fail	2678/7.6787	12435/2.1678	fail
15	**287/0.1341**	fail	5466/19.5829	26649/4.3514	fail
16	**472/0.1807**	fail	9142/29.1413	45629/7.1427	fail
17	**1129/0.3068**	fail	13765/47.4710	75609/11.8238	fail
18	**374/0.1919**	424/0.0307	1255/2.8560	955/0.4863	23097/4.4148
19	7631/1.2491	**3816/0.4067**	5011/9.1345	9535/1.7359	72331/12.8233
20	**11420/2.5007**	333116/83.3204	20199/44.6562	12965/2.2772	271243/71.9321
21	**11994/2.5795**	437387/87.0294	fail	26333/4.1875	fail
22	**22353/4.3259**	fail	fail	45502/6.9854	fail
23	68261/11.3260	fail	fail	**65665/10.2088**	fail
24	**7153/1.5798**	15764/1.9403	800/2.2542	13973/3.3009	300000/55.9388
25	**169/0.1158**	317967/48.3792	3200/6.4923	68557/13.3904	fail
26	**72/0.2061**	454949/72.8232	20000/57.9918	6644/ 1.3298	fail
27	**92**/0.2148	fail	fail	11107/**2.0672**	fail
28	**112/0.2248**	fail	fail	166072/ 2.9690	fail
29	**132/0.2368**	fail	fail	23037/4.1859	fail
30	499/0.4248	432/0.0309	**337**/1.8180	917/**0.4246**	6286/2.1060
31	1706/0.9028	9495/1.5515	**748**/2.4662	2613/ **0.7154**	9534/3.3072
32	6628/2.8975	228991/35.2568	**2944**/8.4843	6298/**1.3603**	17892/7.6128
33	14526/6.7263	135520/18.5295	**6479**/23.7376	23066/**4.0663**	fail
34	24304/12.1846	401757/69.1811	**11487**/50.3674	24527/**7.9458**	fail
35	38368/17.8303	478598/87.9847	**17895**/49.4480	66424/**11.2693**	fail
36	**102/0.1435**	298/**0.0191**	368/1.6647	1038/0.4507	49528/8.1277
37	1167/**0.6198**	211701/30.5712	**895**/2.3140	2972/0.7661	24274/6.6144
38	4522/**2.0778**	409774/88.72117	**2812**/5.3231	11117/2.1164	24118/7.4880
39	8571/**3.9566**	fail	**5339**/18.4640	23660/4.2189	51665/20.3269
40	14401/**6.6602**	fail	**8654**/35.8083	37889/6.6841	fail
41	**1075/0.6992**	fail	12583/73.0270	51652/8.7678	fail
42	155/0.1457	1052/**0.0964**	**146**/1.4823	183/0.2864	fail
43	**546/2.8703**	129047/42.0626	fail	18122/3.2506	fail
44	**38141/7.0891**	221113/42.0041	fail	fail	fail
45	fail	fail	fail	fail	fail
46	**142/0.1501**	4719/1.1473	fail	6380/1.8371	5250/1.8408
47	**7259/4.0403**	336951/90.9013	fail	25883/8.2362	fail
48	**22167/6.8306**	14858/13.3722	fail	fail	fail
49	fail	20948/16.8765	fail	fail	fail
50	2895/0.2959	635/0.0404	233/9.3358	416/0.4454	3741/0.4844
51	fail	fail	429/1.1742	fail	4376/0.6406
59	**79**/0.0969	812/**0.0661**	1250/2.4032	869/0.4087	3763/1.1544
60	**179/0.1100**	3397/0.2835	4879/7.7656	2677/0.6844	8143/2.7144
61	**524/0.1616**	21399/1.8735	20001/51.6496	6442/1.2603	fail
62	**999/0.2486**	52268/4.6691	45000/97.6603	27301/4.3689	fail
63	**1459/ 0.3188**	204131/18.7340	fail	55534/8.4395	fail
64	**1999/0.4095**	387088/35.2641	fail	80871/12.4093	fail

[2] S. Babaie-Kafaki, Z. Aminifard, S. Ghafoori, *Nonmonotone diagonally scaled limited-memory BFGS meth-*

Table 3: (Continued)

Problem number	DRQN $feval/CPU(s)$	RTEHJ $feval/CPU(s)$	MCS $feval/CPU(s)$	AEGPS $feval/CPU(s)$	ACRS $feval/CPU(s)$
65	**307**/0.1513	773/**0.0648**	1250/2.6738	3570/0.9253	3514/1.1700
66	**377/0.1495**	2638/0.2127	5001/7.5070	405965/54.8257	fail
67	**545/0.1752**	14269/1.2861	20000/26.6835	fail	fail
68	**1293/0.3662**	29821/2.6873	45000/88.9233	fail	fail
69	**1001/0.2386**	59573/5.7198	fail	fail	fail
70	**1289/0.3238**	100409/9.6512	fail	fail	fail
71	**707/0.2363**	48680/4.3381	1252/2.4560	55980/7.1216	fail
72	**1157/0.3110**	467463/39.4330	4104/4.4644	29136/8.1583	fail
73	196443/33.2759	fail	**12524/13.9042**	200335/36.9396	fail
74	418318/85.0012	fail	**13779/20.3061**	fail	fail
75	fail	fail	**21725/43.2405**	fail	fail
76	fail	fail	**31400/43.1281**	fail	fail
77	10316/1.8507	149261/38.0386	1256/2.3295	**893/0.4260**	fail
78	32184/6.9592	fail	5000/10.4717	**3459/0.8332**	fail
79	49088/11.9594	fail	20000/47.3173	**7407/1.4242**	fail
80	133980/32.0293	fail	fail	**12791/ 2.2121**	fail
81	238558/61.2592	fail	fail	**22740/4.5859**	fail
82	fail	fail	fail	**25524/4.9713**	fail
83	272/0.1443	**1172/0.1018**	800/2.0314	4405/1.0236	5245/ 1.7940
84	**475/0.1774**	8248/0.7401	3200/4.7220	12048/2.0606	4821/1.8408
85	**2015/0.4787**	60015/5.6749	12800/28.3240	32149/5.0656	fail
86	**1779/0.4250**	107655/10.6568	20200/30.1957	40196/6.0393	fail
87	**1914/0.4627**	198530/19.7912	28800/76.7368	48096/7.4592	fail
88	**1754/0.4126**	257874/25.9186	fail	56204/ 8.4244	fail
89	**10039/2.1806**	fail	fail	80140/13.0114	fail
90	**2005/0.4625**	fail	fail	100001/15.2575	fail
91	1609/0.5491	**867/0.0905**	545/2.0357	939/0.4205	3392/1.2948
92	**11051**/1.7870	6520/1.3023	1274/2.8843	2882/**0.8927**	fail
93	12565/**2.3604**	77664/45.1479	3363/8.0638	12553/3.9082	fail
94	57859/**12.0364**	fail	**6472**/16.7313	27118/13.0105	fail
95	106487/**28.3595**	fail	**12214**/42.1135	42301/ 30.3660	fail
96	fail	fail	**75762**/81.7165	fail	fail
97	6493/0.9757	111542/11.5234	1226/2.0866	**678/0.3666**	12173/2.9952
98	6513/0.9993	230954/24.6094	7.3376/7.3376	**3124/0.7492**	18254/4.3212
99	13312/2.1586	309522/33.9039	19980/21.5487	**10851/1.8860**	30244/13.4629
100	18285/3.7586	fail	42050/99.74658	**18123/2.9291**	40430/23.9930
101	28430/6.3012	fail	fail	**27806/4.4117**	fail
102	**24267/5.869**	fail	fail	36175/5.8309	fail
103	**5432/0.6984**	109230/9.0916	fail	fail	240286/29.3906
104	**132908/12.0909**	fail	fail	fail	fail
105	fail	fail	fail	fail	fail
106	fail	fail	**509/0.8558**	fail	fail
107	316/**0.1288**	427713/37.1732	**304**/1.1030	45637/5.8335	4186/0.5313
108	12099/**1.0093**	283691/25.4181	**498**/1.3183	439/0.2924	12380/1.5156
109	9899/**0.6606**	236329/20.7825	**319**/1.0157	156275/15.5351	9215/1.1250
110	8994/1.0954	22785/0.8825	**200/0.7338**	183937/19.7885	9128/1.0938
111	388/0.1267	**44/0.0047**	1250/1.3287	203/0.2781	2325/0.2813
112	1209/0.0695	**271/0.0153**	3200/5.3300	701/0.3340	fail
113	3878/0.1940	15313/**0.0620**	5000/5.4073	**978**/0.3816	fail
114	12094/0.9958	24542/2.0367	**160**/0.8198	1061/**0.3859**	fail
115	225/0.0343	**122/0.0316**	397/0.9214	203/0.339	3193/0.5781
116	9858/2.9805	**669/0.0375**	1293/1.3116	792/0.5269	4606/0.8438
117	15095/5.0059	3687/**0.1938**	**201**/0.6210	2753/0.6575	27905/5.3281
118	62/0.0857	**115/0.0096**	1250/1.2752	193/0.2611	3455/0.4688
119	**81**/0.0890	862/**0.0517**	3200/2.7699	543/0.3375	4666/0.6875
120	**201/0.1090**	2649/0.1583	5001/4.5211	894/0.3656	5143/ 0.9063
121	**205/0.1093**	4561/0.2741	5001/5.5722	1060/0.4079	7673/1.4844
122	169/0.1242	1036/**0.0685**	fail	**163**/0.3400	15968/1.9531
123	**1832/0.3171**	fail	fail	75441/11.7715	fail
124	fail	fail	fail	**89437/14.0512**	fail
125	**224/0.1312**	5063/0.2086	206/0.7312	204/0.4320	fail

Table 4: Number of function evaluations required by the DRQN algorithm and the other stochastic algorithms to reach the global minimum.

Problem number	DRQN $feval/CPU(s)$	DE $feval/CPU(s)$	SPSO $feval/CPU(s)$	HS $feval/CPU(s)$	G-CART $feval/CPU(s)$	EO $feval/CPU(s)$	COA $feval/CPU(s)$
1	**301**/0.2409	2640/0.3412	700/0.0716	71232/4.7149	322/0.6959	1422/**0.1405**	5037/0.2567
2	**320**/0.1812	6288/0.8093	2996/0.2991	fail(09)	4063/8.1148	1968/**0.1484**	12220/0.6202
3	**61/0.1417**	6264/0.8124	4932/0.4895	7436/0.4616	960/1.0887	4300/0.4979	8986/0.4670
4	**465/0.1964**	91160/12.4980	21272/3.0186	fail(20)	5225/15.7623	188597/17.8046	46454/3.1169
5	**317/0.1697**	12576/2.0571	5040/1.7474	90375/10.3851(1)	2736/9.2372	fail(20)	30117/1.7171
6	**27929/5.3070**	fail(10)	fail(16)	270541/48.5828(4)	fail(20)	fail(20)	106814/6.0952
7	**116183/24.9951**	fail(15)	fail(20)	fail(18)	fail(20)	fail(20)	458842/26.1955(4)
8	**134738/32.2898**	fail(11)	fail(20)	fail(20)	fail(20)	fail(20)	fail(12)
9	fail	fail(06)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
10	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
11	**70/0.0971**	1164/0.1744	2456/0.7459	9925/0.7448	779/0.8374	119/0.0984	4775/0.5957
12	**93/0.1014**	3568/0.5698	5248/1.8088	15270/1.5065	2230/5.4245	2553/0.2374	19465/ 2.3669
13	**127/0.1110**	9648/3.730	13200/2.1121	35044/4.9306	fail(12)	12255/1.3798	86950/11.2123
14	**198/0.1266**	42616/6.9917	58484/20.6384	341824/82.7057(4)	fail(19)	20843/2.0874	198805/25.8456
15	**287/0.1341**	107958/18.0329	fail(18)	fail(12)	fail(20)	24123/2.5333	312475/38.9567(1)
16	**472/0.1807**	219076/38.0083	fail(20)	fail(08)	fail(20)	360804/25.3255(2)	361900/47.4209
17	**1129/0.3068**	438592/81.7535	fail (20)	fail(11)	fail(20)	fail(06)	461700/63.35195(1)
18	**374/0.1919**	2012/0.3433	1596/0.2180	4516/0.4808	1251/3.9725	2487/0.2635	30475/1.1908
19	7631/1.2491	4492/0.8052	**2628**/0.5274	12912/1.8332	3138/14.5657	3877/**0.4201**	77890/3.0394
20	11420/2.5007	9536/1.7367	4800/1.6028	41080/9.5732	7712/33.0923	**4432/0.4444**	169765/6.7281
21	11994/2.5795	14588/2.6591	200017/93.1265	298260/91.6444	fail(15)	**5245/0.5929**	251260/10.0687
22	22353/4.3259	19936/3.6051	fail(06)	fail(05)	fail(20)	**5542/0.5397**	330025/13.4297
23	68261/11.3260	25804/4.7529	fail(20)	fail(09)	fail(20)	**5510/0.6366**	401305/16.8382
24	7153/1.5798	11664/1.9128	267708/41.5324	fail(12)	fail(06)	**7695/0.5040**	27082/1.8662
25	**169/0.1158**	17620/2.8547	350000/70.7129(3)	fail(11)	fail(18)	4360/0.4119	122345/8.4421
26	**72**/0.2061	12640/2.1517	fail(05)	168542/45.3009(3)	fail(20)	2145/**0.1981**	126065/9.0915
27	**92/0.2148**	18380/3.1482	308953/65.3254(4)	fail(20)	fail(20)	2455/0.2848	167945/12.3440
28	**112/0.2248**	23144/3.9049	114496/78.2541(2)	fail(20)	fail(20)	2605/0.2596	244351/18.3591(1)
29	**132**/0.368	27964/4.9080	485968/72.3255 (4)	fail(20)	fail(20)	2845/**0.3497**	239414/18.4144(04)
30	**499**/0.4248	2344/0.4420	2680/0.4717	3784/0.4402	2144/6.8838	1954/**0.2676**	18070/1.1773
31	**1706/0.9028**	5348/0.9580	3876/0.9780	9092/1.5253	4606/19.2140	7850/0.9805	55045/3.6133
32	**6628**/2.8975	11176/2.0511	6980/7.0903	28860/7.9062	8934/34.3338	15122/**2.2080**	119500/7.9855
33	14526/6.7263	17756/3.2792	**10140**/5.6123	fail(12)	fail(12)	22400/**2.3058**	176785/11.9776
34	24304/12.1846	24716/4.5897	**10808**/6.2293	fail(06)	fail(20)	29120/**4.3084**	233980/16.1046
35	38368/17.8303	32636/6.3958	**20640**/18.6353	fail(15)	fail(20)	37965/**3.8966**	289540/20.2766
36	**102**/0.1435	2416/0.4834	2704/0.4896	3164/0.3508	2212/8.6562	410/0.4330	1468/**0.1434**
37	**1167**/0.6198	5340/1.0977	4012/1.0617	9120/1.5242	5033/24.1036	11712/1.4556	3245/**0.5514**
38	**4522**/2.0778	11176/2.2628	6464/2.8278	28960/7.8704	20098/92.3578	16160/2.1580	5094/**0.8948**
39	**8571**/3.9566	17648/3.3914	10076/6.1021	fail(08)	fail(11)	25590/2.2314	14983/**1.4832**
40	**14401**/6.6602	24808/5.0135	15220/11.5545	fail(09)	fail(20)	34510/2.4687	18094/**1.9091**
41	**1075/0.6992**	32820/6.8681	21376/20.3530	fail(17)	fail(20)	41530/3.0970	21938/1.5113
42	**155**/0.1457	708/0.1127	1216/0.1512	1240/**0.0871**	275/0.3019	862/0.0954	2935/0.3607
43	**546**/2.8703	3148/**0.5283**	303504(4)/57.7091	5216/0.5466	fail(05)	fail(09)	25525/3.0734
44	38141/7.0891	55780/8.7762(1)	56528/10.0644	**26464/3.6214**	fail(08)	fail(16)	59530/7.7435
45	fail	59580/9.6003	fail(14)	64152/10.1642	fail(15)	fail(20)	82990/10.8918
46	**142/0.1501**	3916/0.7080	64040/11.0479	71216/6.9471(1)	968/3.758	5823/0.9283	fail(05)
47	**7259/4.0403**	139576/27.6813	54220/11.0580	323496/43.7390(4)	1186/46.4116	400560/42.3869(4)	218963/41.8466(4)
48	**22167/6.8306**	416516/81.3752(4)	103372(1)/29.2819	393362/71.1899(4)	17586/98.2965 (2)	fail(07)	fail(07)
49	fail	fail(20)	52772(1)/17.3358	fail(9)	fail(06)	fail(20)	265123/51.0071(4)
50	2895/0.2959	1184/0.1219	2025/0.3030	2056/0.3317	614/0.5701	1057/0.0370	5248/0.2278
51	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(12)
52	**1089/0.2104**	fail(15)	fail(08)	fail(20)	fail(20)	fail(20)	fail(20)
53	**110624/28.0399**	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
54	**258762/45.1470**	fail(19)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
55	**401105/85.9435**	fail(17)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
56	fail	fail(20)	fail(20)	fail(20)	fail(20))	fail(20)	fail(20)
57	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
58	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
59	**79/0.0969**	3144/0.5018	3368/0.5341	8784/0.9187	1861/6.4064	2715/0.2032	24100/1.2882
60	**179/0.1100**	6852/1.0677	6028/1.3914	21448/2.9967	4697/23.3924	2772/0.2725	77545/4.1505
61	**524/0.1616**	14068/2.2329	20852/8.2030	199410/58.4240	10668/51.2878	3335/0.2469	161305/8.7392
62	**999/0.2486**	22020/3.5101	52156/22.8968	fail(05)	fail(06)	3330/0.3138	241770/13.3362
63	**1459**/ 0.3188	29940/4.8412	100944/57.1542	fail(10)	fail(14)	3652/**0.3059**	320635/17.8487
64	**1999**/0.4095	38728/6.2800	170308/93.7677(1)	fail(18)	fail(20)	3962/**0.4016**	407920/23.1569
65	**307**/0.1513	9304/1.4173	4184/0.6962	11068/1.1398	983/3.8674	1345/**0.1422**	28968/2.0255
66	**377/0.1495**	43744/6.7221	9876/2.4963	20952/3.2574	3935/19.2049	3740/0.4020	119105/8.1767
67	**545/0.1752**	159940/25.0731	35752/14.6440	fail(12)	fail(12)	12175/ 1.2788	392037/26.9962
68	**1293/0.3662**	371912/61.5703	77124/43.5881	fail(08)	fail(18)	29098/3.4381	fail(09)
69	**1001/0.2386**	fail(10)	135896/98.4727	fail(20)	fail(20)	33220/4.0446	fail(12)
70	**1289/0.3238**	fail(18)	212652/93.8313(3)	fail(20)	fail(20)	77015/7.9305	fail(12)
71	**707/0.2363**	20820/3.3032	37148/5.4190	fail(12)	fail(12)	fail(20)	fail(20)
72	**1157/0.3110**	181320/28.5223	276584(2)/67.5560	fail(20)	fail(20)	fail(06)	281465/19.0103

Table 4: Continued.

Problem number	DRQN $feval/CPU(s)$	DE $feval/CPU(s)$	SPSO $feval/CPU(s)$	HS $feval/CPU(s)$	G-CART $feval/CPU(s)$	EO $feval/CPU(s)$	COA $feval/CPU(s)$
73	**196443/33.2759**	fail(08)	fail(20)	fail(20)	fail(20)	fail(11)	fail(20)
74	4**18318/85.0012**	fail(06)	fail(20)	fail(20)	fail(20)	fail(11)	fail(20)
75	fail	fail(11)	fail (20)	fail(20)	fail(20)	fail(20)	fail(20)
76	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)	fail(20)
77	10316/1.8507	5320/0.8668	176788/32.1008(4)	10364/0.9935	fail(05)	**2110/0.2490**	42237/2.4866
78	32184/6.9592	14116/2.3429	fail(06)	23636/3.3730	fail(20)	**3270/0.3835**	101705/5.7970
79	49088/11.9594	50600/8.4894	fail(20)	455048/89.1896(4)	fail(20)	**5113/0.6139**	212825/12.2736
80	133980/32.0293	142708/23.7796	fail(20)	fail(12)	fail(20)	**5053/0.6253**	308071/18.0668
81	238558/61.2592	360420/60.3150	fail(20)	fail(20)	fail(20)	**11343/4.5432**	412420/24.6229
82	fail	fail(20)	fail(20)	fail(20)	fail(20)	fail(05)	**486871/29.5983**
83	**272/0.1443**	8064/1.2501	4004/0.5319	37844/3.2007	1516/4.7361	2111/0.2346	16762/1.1745
84	**475/0.1774**	28348/4.6969	30976/5.3752	427400/74.3949(3)	12546/58.0914	3951/0.3130	66562/4.6974
85	**2015**/0.4787	158152/25.4620	140676/39.1219	fail(17)	fail(10)	5158/**0.4171**	314482/22.6776
86	**1779**/0.4250	317080/52.3128	230492/79.7261	fail(20)	fail(12)	5197/**0.4372**	465442/34.1212(4)
87	**1914**/0.4627	491152/80.9260	353980/96.0873	fail(20)	fail(20)	3815/**0.3131**	fail(12)
88	**1754/0.4126**	fail(10)	fail(20)	fail(20)	fail(20)	4986/0.5942	fail(20)
89	10039/2.1806	fail(20)	fail(08)	fail(20)	fail(20)	**5607/0.7565**	fail(20)
90	**2005/0.4625**	fail(17)	fail(14)	fail(20)	fail(20)	5453/0.7089	fail(20)
91	**1609**/0.5491	3036/0.5258	3292/**0.4883**	7096/0.8679	1932/6.9557	5115/0.8070	24408/3.0958
92	11051/1.7870	7456/**1.3319**	**7324**/1.7166	25920/4.8751	6208/32.2524	11330/3.9301	93117/13.3153
93	**12565/2.3604**	18388/5.8843	42476/20.2240	297420/89.3301(2)	fail(10)	18572/14.1780	258768/87.5680
94	**57859/12.0364**	102456/18.3595	132320/98.1886(2)	fail(05)	fail(12)	26070/35.5304	fail(12)
95	106487/**28.3595**	49820/39.6211	fail(09)	fail(11)	fail(20)	**5380**/69.6549	fail(17)
96	fail	**72772/85.9157**	fail(06)	fail(20)	fail(20)	fail(05)	fail(20)
97	6493/0.9757	**5044/0.6717**	5336/0.7293	24600/2.3562	6114/26.4751	1753/ 0.2079	42550/2.2356
98	6513/0.9993	9908/1.3178	7344/1.4622	34952/4.6243	6176/29.2344	**3442/ 0.4205**	107440/5.6991
99	13312/2.1586	19260/2.6075	12308/3.9865	fail(20)	fail(08)	**4052/0.4592**	229720/12.4291
100	18285/3.7586	28344/4.0764	fail(08)	fail(20)	fail(20)	**4452/0.4845**	327355/18.0816
101	28430/6.3012	38264/5.4343	fail(20)	fail(20)	fail(20)	**4887/0.6036**	412285/23.1954
102	24267/5.7869	48432/6.9138	fail(20)	fail(20)	fail(20)	**5194/ 0.6911**	491995/28.2281(2)
103	5432/**0.6984**	fail(10)	5860/1.0060	252736/16.6449(4)	**3475**/8.3538	300450/10.0423(3)	fail(12)
104	**132908**/12.0909	fail(10)	302560/ 84.9217(4)	159228/16.8822(3)	fail(14)	300309/**10.5323(4)**	fail(20)
105	fail	fail(15)	fail(11)	**360825/66.7224**(4)	fail(14)	fail(8)	fail(20)
106	fail	fail(20)	fail(20)	fail(17)	fail(20)	fail(14)	fail(20)
107	**316/0.1288**	14040/1.3116	4965/0.7962	fail(12)	2128/3.9982	21037/4.2304(2)	21418/0.8759
108	12099/1.0093	17220/1.6057	53640/6.8815(1)	158996/9.4665	**1803**/3.2096	5625/**0.1834**	18820/0.8066
109	9899/0.6606	29304/2.7425	**3930/0.6217**	fail(14)	fail(9)	19070/0.6691	90880/3.5973
110	8994/1.0954	100016/9.6665	**4175/0.6783**	fail(20)	17808/64.8135	106310/3.4598	37324/1.4100
111	388/0.1267	224/0.0279	375/0.1015	736/0.2079	610/0.2962	**180/0.0111**	652/0.0284
112	1209/0.0695	712/0.0806	1390/0.2385	1584/0.1141	1882/3.1155	**455/0.0236**	3463/0.1509
113	3878/0.1940	1216/0.1330	63865/15.7746	3288/0.3112	4279/9.7487	**860/0.0438**	7000/0.3192
114	12094/0.9958	**1424/0.1558**	22840/6.0558	4560/0.4991	5824/14.0280	1335/0.0675	8692/0.3895
115	**225/0.0343**	508/0.0705	875/0.1247	1720/0.0939	291/ 0.2005	650/0.0315	2272/0.1147
116	9858/2.9805	2096/**0.2438**	2705/0.4778	8632/0.6452	**1596**/3.3528	2228/0.1002	15904/0.7973
117	15095/5.0059	5604/0.6293	5305/1.2836	22644/2.5355	**4495**/13.5779	6710/**0.2953**	54388/2.6292
118	62/0.0857	624/0.0772	1215/0.1566	816/0.0477	400/1.3025	**360/0.0217**	2788/0.1493
119	**81**/0.0890	2272/0.2669	3130/0.5142	2958/0.2296	1877/4.7355	840/**0.0441**	18712/0.9746
120	**201**/0.1090	4136/0.5079	4604/0.9945	6100/0.6276	3853/13.4257	1230/**0.0623**	43108/2.2697
121	**205**/0.1093	5400/ 0.6523	5625/1.3057	8968/1.0345	5285/ 19.5989	1410/**0.0767**	60816/3.1984
122	**169**/0.1242	592/0.0724	332/0.0454	fail(9)	279/0.3894	325/**0.0173**	fail(10)
123	**1832**/0.3171	422200/50.5455(4)	170472/36.1353	50096/5.0071	fail(13)	11190/**0.5444**	85696/4.9491
124	fail	415376/47.0313	**227362/43.0603**	fail(20)	fail(20)	fail(14)	fail(11)
125	**224**/0.1312	2224/ 0.2202	2855/0.3663	14444/1.2814	645/0.4473	1410/**0.0475**	4720/0.1925

ods with application to compressive sensing based on a penalty model, Appl. Num. Math. **181** (2022) 618–629.

[3] P. Brachetti, M.D.F. Ciccoli, G.D. Pillo, S. Lucidi, *A new version of the Price's algorithm for global optimization*, J. Glob. Opt. **10** (1997) 165–184.

[4] R. H. Byrd, P. Lu, J. Nocedal, C. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput. **16** (1995) 1190–1208.

[5] Y. Cherruault, A. Guillez, *Une mthode pour la recherche du minimum global dune fonctionnelle*, CRAS Paris **296** (1983) 175–178.

[6] W. Deng, J. Xu, X.Z. Gao, H. Zhao, *An enhanced MSIQDE algorithm with novel multiple strategies for global optimization problems*, IEEE Trans. Syst. Man. Cybern. **52** (2020) 1578–1587.

[7] Y.G. Evtushenko, *Numerical Optimization Techniques*, Springer, Berlin, 1985.

[8] R. Ellaia, M.Z. Es-Sadek, H. Kasbioui, *Modified Piyavskii's global one-dimensional optimization of a differentiable function*, Appl. Math. **3** (2012) 1306–1320.

[9] M. EL-Alem, A. Aboutahoun, S. Mahdi, *Hybrid gradient simulated annealing algorithm for finding the global optimal of a nonlinear unconstrained optimization problem*, Soft Comput. **25** (2021) 2325–2350.

[10] A. Faramarzi, M. Heidarinejad, B. Stephens, S. Mirjalili, *Equilibrium optimizer: A novel optimization algorithm*, Knowledge-Based Syst. **191** (2020) 105190.

[11] Z.W. Geem, J.H. Kim, G.V. Loganathan, *A new heuristic optimization algorithm: harmony search*, Simulation **76** (2001) 60–68.

[12] D. Guettal, A. Ziadi, *Reducing transformation and global optimization*, Appl. Math. Comput. **218** (2012) 5848–5860.

[13] W. Huyer, A. Neumaier, *Global optimization by multilevel coordinate search*, J. Glob. Opt. **14** (1999) 331–355.

[14] N. Keskar, A. Wachter, *A limited-memory quasi-Newton algorithm for bound-constrained non-smooth optimization*, Opt. Meth. Soft. **34** (2019) 150–171.

[15] A. Kumar, R.K. Misra, D. Singh, S. Mishra, S. Das, *The spherical search algorithm for bound-constrained global optimization problems* Appl. Soft Comput. **85** (2019) 105734.

[16] M. Ouanes, H. A. Le Thi, T. P. Nguyen, A. Zidna, *New quadratic lower bound for multivariate functions in global optimization*, Math. Comput. Simul. **109** (2015) 197–211.

[17] J. Pierezan, L. S. Coelho, *Coyote Optimization Algorithm: A new metaheuristic for global optimization problems*, IEEE Congress on Evolutionary Computation, (2018) 2633–2640.

[18] M. Rahal, A. Ziadi, *A new extension of Piyavskii's method to Hlder functions of several variables*, Appl. math. Comput. **197** (2008) 478–488.

[19] B.L. Robertson, C.J. Price, M. Reale. *A CARTopt method for bound-constrained global optimization*, The ANZIAM Journal **55** (2013) 109–128.

[20] A. Sahiner, S.A. Ibrahem, *A new global optimization technique by auxiliary function method in a directional search*, Optim. Lett. **13** (2019) 309–323.

[21] M. Salahi, A. Jamalian, A. Taati, *Global minimization of multi-funnel functions using particle swarm optimization*, Neural. Comput. Appl. **23** (2013) 2101–2106.

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

[23] S. Surjanovic, D. Bingham, *Virtual Library of Simulation Experiments: Test Functions and Datasets*, Retrieved January 28, 2023, from http://www.sfu.ca/ ssurjano/index.html, (2013).

[24] M. Zambrano-Bigiarini, C. Maurice, R. Rodrigo, *Standard particle swarm optimisation: A baseline for future pso improvements*, IEEE Congress on Evolutionary Computation, 2013.

[25] A. Ziadi, Y. Cherruault, *Generation of α-dense curves and application to global optimization*, Kybernetes **29** (2000) 71–82.

[26] A. Ziadi, Y. Cherruault, G. Mora, *Global optimization: A new variant of the Alienor method*, Comput. Math. Appl. **41** (2001) 63–71.

[27] R. Ziadi, A. Bencherif-Madani, R. Ellaia, *Continuous global optimization through the generation of parametric curves*, Appl. Math. Comp. **282** (2016) 65–83.

[28] R. Ziadi, R. Ellaia, A. Bencherif-Madani, *Global optimization through a stochastic perturbation of the Polak-Ribière conjugate gradient method*, J. Comput. Appl. Math. **317** (2017) 672–684.

[29] R. Ziadi, A. Bencherif-Madani, *A covering method for continuous global optimization*, Int. Jour. Comput. Sci. Math. **13** (2021) 369–390.

[30] R. Ziadi, A. Bencherif-Madani, R. Ellaia, *A deterministic method for continuous global optimization using a dense curve*, Math. Comput. Simul. **178** (2020) 62–91.