GIBO: Global Integral-Based Optimization

Sebastien Labbé, Andrea del Prete

To cite this version:

Sebastien Labbé, Andrea del Prete. GIBO: Global Integral-Based Optimization. 2023. hal-04273212

HAL Id: hal-04273212
https://hal.science/hal-04273212

Preprint submitted on 8 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

GIBO: Global Integral-Based Optimization

Sebastien Labbe
École normale supérieure - PSL

Andrea Del Prete
University of Trento

Abstract

Numerical optimization has been the workhorse powering the success of many machine learning and artificial intelligence tools over the last decade. However, current state-of-the-art algorithms for solving unconstrained non-convex optimization problems in high-dimensional spaces, either suffer from the curse of dimensionality as they rely on sampling, or get stuck in local minima as they rely on gradient-based optimization. We present a new graduated optimization method based on the optimization of the integral of the cost function over a region, which is incrementally shrunk towards a single point, recovering the original problem. We focus on the optimization of polynomial functions, for which the integral over simple regions (e.g. hyperboxes) can be computed efficiently. We show that this algorithm is guaranteed to converge to the global optimum in the simple case of a scalar decision variable. While this theoretical result does not extend to the multi-dimensional case, we empirically show that our approach outperforms state-of-the-art algorithms (BFGS and CMA-ES) in high dimensions (up to 72 decision variables) when tested on sparse polynomial functions with a high number of local minima.

1 Introduction

Many problems in the fields of science and engineering can be cast as the minimization/maximization of a scalar cost function with respect to its variables. When this function is convex and differentiable, efficient gradient-based optimization algorithms [Rud16] can be used to find the global minimizer/maximizer. However, when this function is highly non-convex, gradient-based optimization can fail to find a satisfying solution, and global optimization techniques are necessary. For instance, this is the case in protein structure prediction [KB19], global cluster structure optimization [Har11], and the training of deep neural networks [KB14].

The global optimization of non-convex functions in high dimensional spaces is still an open problem. Current algorithms can be divided in two categories. Stochastic algorithm (e.g., evolutionary algorithms [BS93], Bayesian optimization [SLA12]) are able to explore the decision space, but do not scale well because of the curse of dimensionality. They indeed need to carry out a dense sampling of the whole decision space, which scales exponentially with the number of decision variables [Han23]. Deterministic global optimization algorithms exist, and can sometimes provide strong optimality guarantees [JPS93; HN99; Las01]. However, they are typically limited to a specific class of functions (e.g., polynomials, or Lipschitz functions) and they also scale badly with the number of decision variables.

Our goal is thus to develop an algorithm that could overcome these limitations. To do so, we rely on the key idea to exploit the integral of the cost function, which, contrary to the gradient, provides non-local information. This fact is well-known in the literature, and it is at the core of methods such as Gaussian Smoothing [GS22], Randomized Smoothing, Gaussian Homotopy Continuation [Iwa+22] and Graduated Optimization. However, such methods try to compute the convolution between the function to optimize and a smoothing kernel (typically a Gaussian kernel), which cannot be computed analytically, and therefore needs to be approximated using sampling. Therefore, they suffer from the curse of dimensionality, and tend to be inefficient in high dimensions.

To overcome this issue, we restrict ourselves to functions that are analytically integrable, which include polynomials as a particularly interesting case. Moreover, rather than computing the convolution with a Gaussian kernel, we compute the convolution with a kernel that is unitary over a simple set (e.g. hyperbox) and zero elsewhere. This results in a computationally efficient optimization method, which we called Global Integral-Based Optimization (GIBO).

The method is implemented in Python, and evaluated on

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume TBD. Copyright 2024 by the author(s).

several non-convex unconstrained optimization problems, comparing it with other global optimization methods. Our results show that GIBO has great capabilities of avoiding local minima, outperforming the other algorithms in high dimensions (up to 72 variables).

2 Method

2.1 Definitions

We are interested in solving unconstrained optimization problems of the form:

$$\underset{x}{\text{minimize}}\, f(x) \tag{1}$$

where $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function, generally nonconvex, and $n \in \mathbb{N}_+$ is the dimension of the pre-image of f. We introduce now the notation used throughout the paper:

- $A \subset \mathbb{R}^n$ is a connected and compact set over which we integrate.
- $s \in \mathbb{R}_+$ denotes the target hypervolume (or size) of the set A.
- $S(\cdot)$ is a function that takes a set A and returns its hypervolume (or, less formally, size).
- When A is an axis-aligned hyper-rectangle, we parametrize it with its center $c \in \mathbb{R}^n$ and its half width $w \in \mathbb{R}^n_+$. Thus, the set corresponding to a pair (c, w) is
$$A_{cw} = \{y \in \mathbb{R}^n | c - w \leq y \leq c + w\}.$$
- $S(w) = \prod_{i=1}^n (2w_i)$ is a shortcut for the size $S(A_{cw})$.

We now describe the GIBO algorithm, which tries to find the minimum of a given function f over a given set A.

2.2 One-dimensional case

Before introducing our algorithm for the general multivariate case, we develop its core idea in the 1D case, which gives us insight into its working principles.

Given a smooth non-convex function $f(\cdot) : \mathbb{R} \to \mathbb{R}$ and the optimization problem :

$$\underset{x}{\text{minimize}}\, f(x) \tag{2}$$

GIBO's key idea is to minimize the integral of f over an interval of fixed width $w \in \mathbb{R}_+$, centered in $c \in \mathbb{R}$:

$$\underset{c}{\text{minimize}} \int_{c-w}^{c+w} f(x)\, \mathrm{d}\, x \tag{3}$$

Algorithm 1 GIBO-1D

```
1: Input(f, c, w, w_min, α_w)
2: while w > w_min do
3:     c ← c − α_c ∂_c I(c, w)
4:     w ← w − α_w
```

We first solve the problem for a large interval (i.e. a large value of w) and then we slowly decrease it, each time warm-starting the minimization with the result of the previous optimization, until we reach $w \approx 0$. Therefore, GIBO minimizes the following two quantities in an alternate manner:

$$I(c, w) \stackrel{\text{def}}{=} \int_{c-w}^{c+w} f(x)\, \mathrm{d}\, x$$

and

$$S(A_{c,w}) = S(w) = 2w.$$

Algorithm 1 shows the GIBO algorithm in its simplest form. The efficiency of this algorithm depends greatly on the choice of the step sizes α_c and α_w. One can find α_c using a line search procedure, but finding the optimal α_w is more complex.

2.3 Choosing α_w

To investigate the choice of α_w, we follow the classic approach [SBC16] of analysing an Ordinary Differential Equation (ODE) that is the limit of the 1D GIBO algorithm for infinitesimal values of α_c, α_w:

$$\begin{cases} \dot{c}(t) = -\partial_c I(c(t), w(t)) \\ \dot{w}(t) = -\alpha_w \end{cases} \tag{4}$$

The first equation describes a gradient descent on the cost function of (3), whereas the second equation describes the decrease of the interval width w. In the following we omit the time dependency to ease the notation. We begin by analyzing how the system behaves if $\dot{w}$ were proportional to w. We set $\alpha_w = \epsilon w$, with ϵ being a positive constant.

$$\begin{cases} \dot{c} = -\partial_c I(c, w) = f(c - w) - f(c + w) \\ \dot{w} = -\epsilon w \end{cases} \tag{5}$$

Given this system we can deduce that if $f(c+w) < f(c-w)$ then $\dot{c} > 0$ and therefore c moves towards $c + w$. Similarly, if $f(c - w) < f(c + w)$ then $\dot{c} < 0$ and therefore c moves towards $c - w$. In any case c moves towards the interval extreme associated to the smaller value of f, while w moves towards zero. It seems therefore reasonable that c should converge to the global minimum of f. However, we show that this only happens if w does not decrease too fast. More in detail, we prove that, if ϵ is sufficiently small, once $c + w$ or $c - w$ is at the global minimum, then c keeps moving towards it.

Theorem 1. *Let $c^\star$ be the unique global minimizer of the function f. Assume the initial values of c, w are such that $c(0) - w(0) \leq c^* \leq c(0) + w(0)$, and assume that ε satisfies the following bound:*

$$\begin{aligned} \varepsilon < 2\frac{f(c^\star + y) - f(c^\star)}{|y|} \\ \forall y \in [c(0) - w(0) - c^*, c(0) + w(0) - c^*] \end{aligned} \tag{6}$$

Then, letting $c(t)$ and $w(t)$ evolve according to the ODE (5) *we have that:*

$$\begin{aligned} \lim_{t\to\infty} c(t) = c^\star \\ \lim_{t\to\infty} w(t) = 0 \end{aligned} \tag{7}$$

Proof. It is straightforward to see that $(c^*, 0)$ is an equilibrium for the system (5). To prove the convergence of the algorithm to this equilibrium point, we prove the asymptotic stability of the ODE (5) using Lyapunov's direct method [Lya92], which has many analogies with optimization theory [PS17]. We use the following candidate Lyapunov function:

$$V(c, w) = |c + w - c^\star| + |c - w - c^\star|$$

This function satisfies the two basic conditions of being always positive, except at $(c^*, 0)$, where it is null:

$$\begin{cases} V(c, w) > 0 \quad \forall (c, w) \neq (c^*, 0) \\ V(c^\star, 0) = 0 \end{cases}$$

To show that V is indeed a Lyapunov function we need to prove that

$$\dot{V} = \partial_c V \cdot \dot{c} + \partial_w V \cdot \dot{w} < 0 \qquad \forall (c, w) \neq (c^*, 0)$$

Because of the non-differentiability of the absolute value, we must separately analyze three cases.

1. If $c - w < c^\star < c + w$, then $\dot{V} = (1-1)\dot{c} + 2(-\varepsilon w) = -2\varepsilon w < 0$.
2. If $c^\star = c + w$, then $\partial_c V$ depends on the sign of $\dot{c} + \dot{w} = \underbrace{f(c - w) - f(c^\star)}_{>0} - \varepsilon w$. If ε is small enough to satisfy (6) we have $\dot{c} + \dot{w} > 0$, which implies that $\dot{V} = -2\varepsilon < 0$.
3. If $c^\star = c - w$, then $\partial_c V$ depends on the sign of $\dot{c} - \dot{w} = \underbrace{f(c^\star) - f(c + w)}_{<0} + \varepsilon w$. If ε is small enough to satisfy (6) we have $\dot{c} - \dot{w} < 0$, which implies that $\dot{V} = -2\varepsilon < 0$.

Therefore, in all three cases we have $\dot{V} < 0$, as long as ε is sufficiently small to satisfy (6). This guarantees that c and w converge to the minimizer of V, i.e. $c \to c^\star$ and $w \to 0$. □

This proof shows that $c^\star$ never leaves $[c-w, c+w]$ because every time $c^\star$ touches one extreme, that extreme moves away. Moreover, from the analysis of $\dot{V}$ in the case where $c^\star = c + w$, we obtain the following bound on ε.

$$\begin{aligned} f(c - w) - f(c^\star) > \varepsilon w \\ f(c^\star - 2w) - f(c^\star) > \varepsilon w \\ \varepsilon < 2\frac{f(c^\star - 2w) - f(c^\star)}{2w} \end{aligned} \tag{8}$$

Similarly, from the case where $c^\star = c - w$, we obtain:

$$\varepsilon < 2\frac{f(c^\star + 2w) - f(c^\star)}{2w} \tag{9}$$

These bounds, which are equivalent to the condition (6), tell us that ε must be upper bounded by the (normalized) difference between $f(c^\star)$ and any other values of f. From this we can infer that:

- if there are local minima that are very distant from $c^\star$, but with value very close to $f(c^\star)$, then we need to set ε very small to ensure $\dot{V} < 0$;
- if there exist multiple global minima, then the upper bound on ε is zero, therefore the above analysis breaks down.

In conclusion, setting $\dot{w} = -\varepsilon w$ is troublesome for two main reasons. First, it requires information on f to ensure that ε is small enough. Second, it can lead to very slow convergence if the needed ϵ is very small. We thus look for an alternative strategy to decrease w.

2.3.1 Tying $\dot{w}$ and $\dot{c}$

As we just saw, using a $\dot{w}$ proportional to w can lead to slow convergence. Relying on the convergence proof presented above, we can ask what is the largest $\dot{w}$ that we can use while always satisfying the following three inequalities:

1. if $c - w < c^\star < c + w$: $\dot{V} = 2\dot{w} < 0$
2. if $c^\star = c + w$: if $\dot{c} + \dot{w} \geq 0 \Rightarrow \dot{V} = 2\dot{w} < 0$
3. if $c^\star = c - w$: if $\dot{c} - \dot{w} \leq 0 \Rightarrow \dot{V} = 2\dot{w} < 0$

We thus obtain the following constraints:

$$\begin{cases} \dot{w} < 0 \\ \dot{c} + \dot{w} > 0 \iff \overbrace{f(c^\star) - f(c - w)}^{-\dot{c} \leq 0} < \dot{w} < 0 \\ \dot{c} - \dot{w} < 0 \iff \underbrace{f(c^\star) - f(c + w)}_{\dot{c} \leq 0} < \dot{w} < 0 \end{cases}$$

Algorithm 2 GIBO-1D

1: **Input**$(f, c, w, w_{min}, \beta, \varepsilon)$
2: **while** $w > w_{min}$ **do**
3: $\quad c_{dot} \leftarrow \alpha_c \cdot \partial_c I(c, w)$
4: $\quad c \leftarrow c - c_{dot}$
5: $\quad w \leftarrow w - \beta \cdot abs(c_{dot}) - \varepsilon w$

We can infer that $\dot{w}$ should be bounded in $[-|\dot{c}|, 0]$. In our case we decide to use $\dot{w} = -\beta|\dot{c}|$ with $0 < \beta < 1$. The advantage of this choice of $\dot{w}$ is that it does not require any prior information on f to tune its hyper-parameters. The only issue with this version is that the ODE could converge to points that are only local minima, because as soon as $\dot{c} = 0$ we also have $\dot{w} = 0$. This however can be simply avoided by adding a small linear term in the update rule:

$$\dot{w} = -\beta|\dot{c}| - \varepsilon w$$

This time, ε can be set arbitrarily small without affecting the convergence speed of the algorithm, which would be anyway ensured by the first term. This new update rule for w gives us Algorithm 2.

2.4 Multi-dimensional case

Based on the 1-D algorithm we now explain its extension to the multi-variate case. First of all, we must choose a parametrization for the set over which the integral is computed. While in the 1D case this choice was trivial, in the multi-dimensional case we have countless options. Ideally, we would like a parametrization that allows to represent any bounded connected set, but that is simply not possible in practice. More realistically, we could choose a complex parametrization (e.g., a deep neural network) that allows to represent complex set shapes. However, by doing so, computing the integral of f over such a complex set would not be efficient. Given our objective to obtain an efficiently computable integral, we choose a simple parametrization, which is however sufficiently expressive to give good results in practice, an axis-aligned hyper-rectangle:

$$A_{cw} = \{y \in \mathbb{R}^n | c - w \le y \le c + w\},$$

parametrized by its center $c \in \mathbb{R}^n$ and half-width $w \in \mathbb{R}^n_+$. Thus, the two values that GIBO minimizes become:

$$I(c, w) = \int_{A_{cw}} f(x)\, \mathrm{d}^n x$$

and

$$S(A_{cw}) = S(w) = \prod_{i=1}^{n} (2w_i).$$

While in the 1D case the minimization of the integral was only changing the variable c, in the multi-dimensional case it must change both c and w. This is because we can now change w without changing the size of A_{cw}. Therefore, the minimization problem becomes:

$$\begin{aligned} \underset{c,w}{\text{minimize}} \quad & I(c, w) \\ \text{subject to} \quad & S(w) = s, \end{aligned} \tag{10}$$

with s being the set size that we wish to maintain. Instead of dealing with a constrained optimization problem, we have found that it works better to relax the constraint and add to the cost a quadratic penalty on $S(w) - s$ with a large weight ζ:

$$I_p(c, w, s) = I(c, w) + \zeta \cdot (s - S(w))^2$$

2.4.1 Efficient Integral Evaluation

Finally, we need to discuss the computation of the integral. Even for analytically integral functions, in general the evaluation of the integral over an axis-aligned hyper-rectangle requires 2^n evaluations of the primitive function. To understand this, let us analyze an example of a generic 3D function $f(x, y, z)$, and assume we want to compute:

$$\int_{\underline{z}}^{\bar{z}} \int_{\underline{y}}^{\bar{y}} \int_{\underline{x}}^{\bar{x}} f(x, y, z)\, \mathrm{d}\, x \ \mathrm{d}\, y \ \mathrm{d}\, z \tag{11}$$

Let us define the following primitive functions, which we assume we can compute analytically:

$$\begin{aligned} f_x(x, y, z) &\triangleq \int f(x, y, z)\, \mathrm{d}\, x \\ f_y(x, y, z) &\triangleq \int f_x(x, y, z)\, \mathrm{d}\, y \\ f_z(x, y, z) &\triangleq \int f_y(x, y, z)\, \mathrm{d}\, z \end{aligned} \tag{12}$$

The integral (11) can be expressed as:

$$\begin{aligned} &\int_{\underline{z}}^{\bar{z}} \int_{\underline{y}}^{\bar{y}} \left(f_x(\bar{x}, y, z) - f_x(\underline{x}, y, z) \right) \mathrm{d}\, y \ \mathrm{d}\, z = \\ &\int_{\underline{z}}^{\bar{z}} \left(f_y(\bar{x}, \bar{y}, z) - f_y(\underline{x}, \bar{y}, z) - f_y(\bar{x}, \underline{y}, z) + f_y(\underline{x}, \underline{y}, z) \right) \mathrm{d}\, z = \\ &f_z(\bar{x}, \bar{y}, \bar{z}) - f_z(\underline{x}, \bar{y}, \bar{z}) - f_z(\bar{x}, \underline{y}, \bar{z}) + f_z(\underline{x}, \underline{y}, \bar{z}) - \\ &f_z(\bar{x}, \bar{y}, \underline{z}) + f_z(\underline{x}, \bar{y}, \underline{z}) + f_z(\bar{x}, \underline{y}, \underline{z}) - f_z(\underline{x}, \underline{y}, \underline{z}) \end{aligned} \tag{13}$$

Therefore, evaluating the 3D integral (11) requires $2^3 = 8$ evaluations of the primitive function f_z. This would make the algorithm inefficient for large values of n. For this reason, we restricted our focus on functions with this form:

$$f(x) = \sum_i a_i \prod_j f_{i,j}(x_j),$$

where each $f_{i,j}$ is an analytically integrable function that depends only on a single variable x_j. Note that all polynomials can be represented under this form, where the functions

Algorithm 3 GIBO-implementation-nD

1: **Input**$(f, c, w, s_{min}, \beta, \gamma)$
2: $s \leftarrow S(w)$
3: **while** $s > s_{min}$ **do**
4: $\quad c_{dot} \leftarrow \alpha_c \cdot \partial_c I(c, w)$
5: $\quad c \leftarrow c - c_{dot}$
6: $\quad w \leftarrow w - \beta \cdot abs(c_{dot})$
7: $\quad s \leftarrow \min(S(w), s/\gamma)$
8: $\quad w \leftarrow \text{argmin}_x I_p(c, x, s)$

$f_{i,j}(x_j)$ are simply powers of x_j. These functions allow for an efficient evaluation of their integral, where the analytical primitive of each function $f_{i,j}$ needs to be evaluated only twice:

$$I(c, w) = \sum_i a_i \prod_j \int_{c_j - w_j}^{c_j + w_j} f_{i,j}(y) dy$$

2.4.2 Overview of the algorithm

Putting it all together, we summarize the multi-dimensional version of GIBO in Algorithm 3.

Lines 4, 5, 6 are almost the same as in the 1d algorithm. The only difference is that the corrective term $-\epsilon w$, which ensured a non-zero decrease of w, is here replaced by line 7, which ensures that s decreases by at least a γ factor at each iteration. Lines 6 and 7 play the role of decreasing s and changing w proportionally to c_{dot}. Then, in line 8 we try to minimize the cost by changing the set's shape without changing its size. To achieve this, we perform gradient descent with the function $I_p(c, \cdot, s)$, warm-started with the current value of w.

In this algorithm we decided to decouple the minimization with respect to w and c in order to first use c_{dot} to decrease the size of w, and then to optimize the set's shape with respect to the new size. To speed up the algorithm, the minimization in line 8 can be limited to a certain number of iterations of gradient descent. In our tests, we used 2 steps of gradient descent.

We also set a lower bound for the width w in order to better spread out the optimization on all the variables and to stop GIBO from optimizing some variables really well to meet the area size constraint.

The values that we chose for the hyper-parameters are:

- α_c is chosen by doing a line search along $\partial_c I(c, w)$.
- β represents the speed at which w decreases, and we chose values in $[0.5, 0.99]$.
- ζ represents the weight of the constraint penalty, and we chose values in $[10^5, 10^9]$.

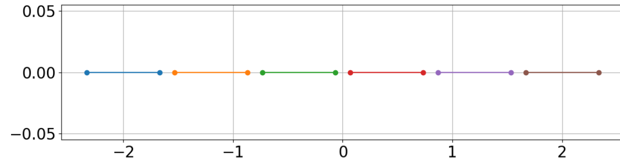


Figure 1: Sampling regions for the $a_{i,j}$ variables.

- γ represents the minimum rate of decrease for s, and we used values in $[1.001, 1.05]$.

3 Results

We designed our tests to see how well GIBO performed when trying to minimize non-convex polynomial functions, compared to other two approaches: repetitive gradient descent (RGD) from random initial points, and the Covariance Matrix Adaptation Evolution strategy (CMA-ES). In order to test the algorithms in high dimensions, we focused on sparse polynomial functions, which are common in many real-world applications, such as optimal control problems.

3.1 Implementation Details

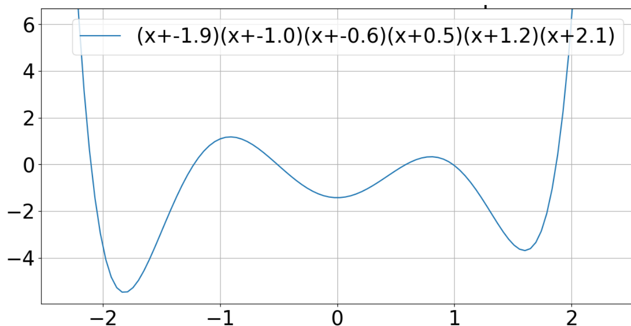
We implemented the algorithm in python [XX] and used 2 outside function calls per loop. The gradient descent step which updates the rectangle's center c_{dot} is computed using the `minimize` function from the *scipy* library, with the Sequential Least SQuares Programming (SLSQP) method. The $argmin_x I_p$ step is computed using the same `scipy.minimize` function with the quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno (BFGS).

The repetitive gradient descnet (RGD) approach was implemented by running the `scipy.minimize` function with BFGS, repetitively starting from uniformly chosen random points within the bounds.

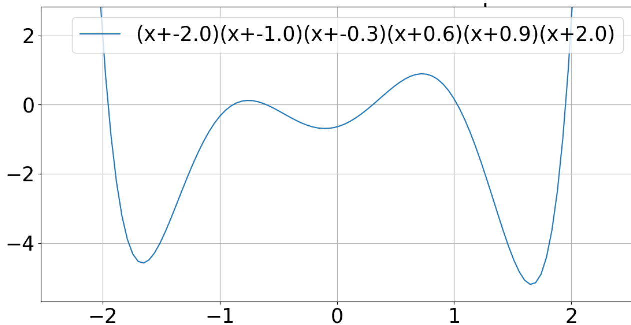
CMA-ES was implemented using the fast-cma-es library [Wol22]. The `advretry.minimize` functions was used.

3.2 Test functions

The cost functions are of the form $f(x) = p(x) + c(x)$. The first term is defined as $p(x) = \Sigma_{i=1}^{n} \Pi_{j=1}^{6} (x_i - a_{i,j})$, with $a_{i,j}$ chosen uniformly at random in an interval of width $\frac{2}{3}$, centered around $-2 + (j-1)\frac{4}{5}$ (see Figure 1 and 2). This term is simply the sum of n 1D 6-th order polynomial, defined in a way to obtain as many local minima as possible. The second term is instead defined as $c(x) = \Sigma_{i=1}^{n-1} b_i x_i x_{i+1}$. This term introduces coupling between neighbor decision variables. For these functions, the global minimum is contained in the hyper-rectangle $A_{c=0,w=2.2}$.



(a) Example 1.



(b) Example 2.

Figure 2: Example functions for p(x) (values rounded to 1 decimal place).

3.3 Test Description

Each function was optimized using 3 different algorithms. Given a generated function f and its set $A_{0,2.2}$:

1. We first ran GIBO on f, $A_{0,2.2}$ to completion.
2. We then ran RGD multiple times, each time starting from a random point in $A_{0,2.2}$.
3. We then ran CMA-ES multiple times, using $A_{0,2.2}$ as bounds.

All the algorithm were given the same amount of time as GIBO and they were all given a single core to run on an intel core i7 8th generation with 32Gb of RAM. For each function we also ran a forth test, simply used to normalize the results. We randomly sampled points in $A_{0,2.2}$ for the same amount of time as GIBO and used the average and minimum value of f at the sampled points to normalize the results of the other algorithms. In particular, a score of 0 means that the algorithm has found a solution equal to the average value found by random sampling. A score of -1 instead means that the algorithm has found the same minimum found by random sampling. Lower scores mean that the algorithm performed better than random sampling.

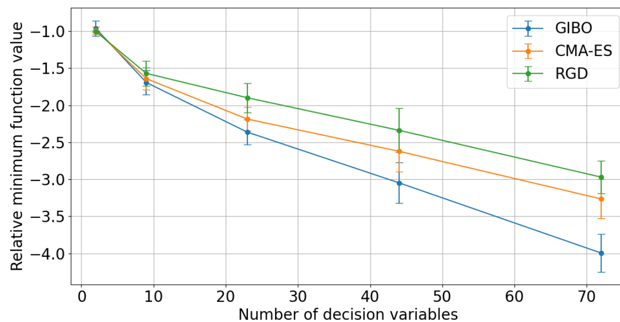


Figure 3: Average performance of the algorithms, as a function of the number of decision variables.

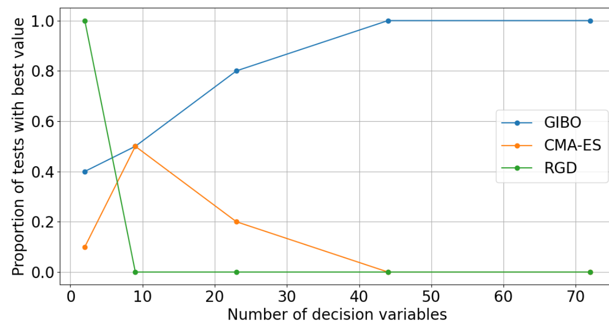


Figure 4: Percentage of tests where each algorithm outperformed the others.

3.4 Discussion of the Results

Figure 3 shows the average performance and its standard deviation after running each algorithm 10 times on randomly generated functions of different sizes. Figure 4 shows the number of times each algorithm found the best minimum out of all three algorithms (when they obtain equal minima we consider that they both outperformed the others). Unsurprisingly, as soon as the dimension is larger than 2, all three algorithms perform better than random sampling because they always obtain a score < -1. This is even more the case as the problem size increases, highlighting a growing inefficiency of pure random sampling. Compared to CMA-ES

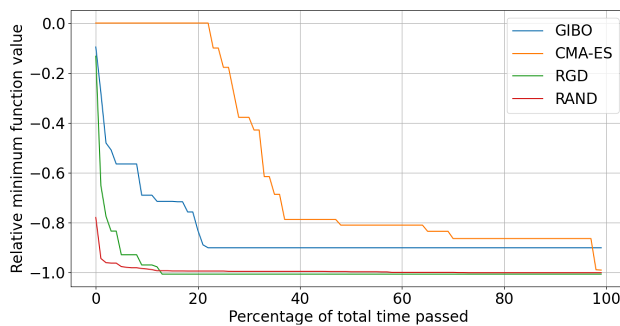


Figure 5: Relative minimum found as a function of computation time, averaged over 10 tests with 2 decision variables.

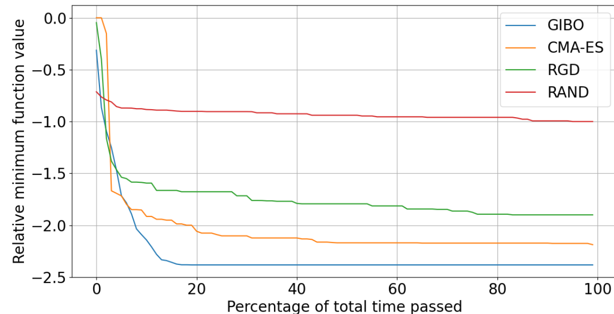


Figure 6: Relative minimum found as a function of computation time, averaged over 10 tests with 23 decision variables.

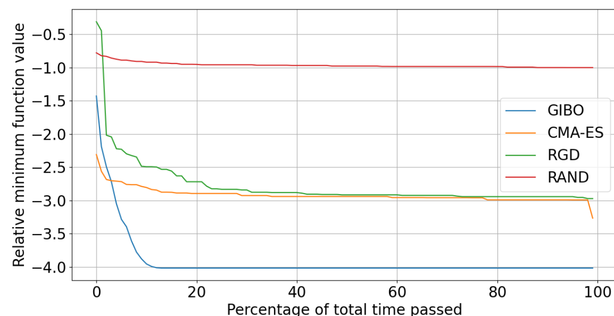


Figure 7: Relative minimum found as a function of computation time, averaged over 10 tests with 72 decision variables.

and RGD, GIBO struggles in small dimensions, probably due to the relatively small number of local minima, which allows RGD and CMA-ESto quickly find the global minimum. However, as the problem dimension grows, GIBO closes the gap and even outperforms CMA-ES by dimension 25. After that GIBO continues to distance itself from the other algorithms as the dimension grows. This shows that GIBO is less affected by the problem dimension, and it works better than the other algorithms when the number of local minima becomes very large. For instance, consider that in dimension 70, our test functions have roughly $3^{70} \approx 10^{33}$ local minima.

Figures 5, 6, 7 show how fast each algorithm converges to its final value. We can see that in dimension 2 GIBO converges slowly and does not always find the best minimum. However, in higher dimensions, GIBO not only converges faster, but also finds better results.

4 Conclusions

We have presented GIBO, a novel unconstrained optimization algorithm for non-convex functions. While state-of-the-art algorithms for global optimization rely on sampling and local optimization, which do not scale well to high dimensions, we rely on the analytical integral of the cost function, which gives us rich non-local information.

We were able to minimize polynomial functions with up to 74 variables, computing their integral over axis-aligned hyper-rectangles. We have compared GIBO with the Covariance Matrix Adaptive Evolution Strategy algorithm and a gradient-based optimization algorithm initialized through random sampling. Our results show that GIBO outperformed the other algorithms for problem sizes above 20, empirically proving the interest of this idea.

However, there are still directions that we would like to investigate to improve GIBO. First, the limitations placed on the type of functions that can be optimized. In this paper we have focused on the optimization of polynomial functions because they are simple to integrate. A possible way to generalize our approach to other functions could be to use *Integral Neural Networks* [Kor23; LIA20], which are networks that provide a direct evaluation of the exact integral of a function. However, this still does not solve the issue that to integrate the function over an N-dimensional hyperbox requires evaluating the network 2^N times.

Moreover, this work has only explored hyper-boxes as regions over which integrals are computed. It may however be possible to compute integrals over more complex shapes while maintaining computational efficiency.

Finally, the update rule for the set size can be potentially improved to skip over many useless iterations where the position of the set does not change significantly, and thus the set size is slowly reduced as well.

References

[XX] X. *X.* X. X. X.

[Lya92] A.M. Lyapunov. “The general problem of motion stability”. In: *Annals of Mathematics Studies* 17 (1892).

[BS93] Thomas Bäck and Hans-Paul Schwefel. “An overview of evolutionary algorithms for parameter optimization”. In: *Evolutionary computation* 1.1 (1993), pp. 1–23.

[JPS93] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. “Lipschitzian optimization without the Lipschitz constant”. In: *Journal of optimization Theory and Applications* 79 (1993), pp. 157–181.

[HN99] Waltraud Huyer and Arnold Neumaier. “Global optimization by multilevel coordinate search”. In: *Journal of Global Optimization* 14 (1999), pp. 331–355.

[Las01] Jean B Lasserre. “Global optimization with polynomials and the problem of moments”. In: *SIAM Journal on optimization* 11.3 (2001), pp. 796–817.

[Har11] Bernd Hartke. "Global optimization". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1.6 (2011), pp. 879–887.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25 (2012).

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747`.

[SBC16] Weijie Su, Stephen Boyd, and Emmanuel J. Candès. "A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights". In: *Journal of Machine Learning Research* 17 (2016), pp. 1–43. ISSN: 15337928.

[PS17] Boris Polyak and Pavel Shcherbakov. "Lyapunov Functions: An Optimization Theory Perspective". In: *IFAC-PapersOnLine*. Vol. 50. 1. 2017, pp. 7456–7461. DOI: `10.1016/j.ifacol.2017.08.1513`.

[KB19] Brian Kuhlman and Philip Bradley. "Advances in protein structure prediction and design". In: *Nature Reviews Molecular Cell Biology* 20.11 (2019), pp. 681–697.

[LIA20] Steffan Lloyd, Rishad A Irani, and Mojtaba Ahmadi. "Using neural networks for fast numerical integration and optimization". In: *IEEE Access* 8 (2020), pp. 84519–84531. ISSN: 21693536. DOI: `10.1109/ACCESS.2020.2991966`.

[GS22] Katelyn Gao and Ozan Sener. *Generalizing Gaussian Smoothing for Random Search*. 2022. arXiv: `2211.14721 [cs.LG]`.

[Iwa+22] Hidenori Iwakiri et al. *Single Loop Gaussian Homotopy Method for Non-convex Optimization*. 2022. arXiv: `2203.05717 [math.OC]`.

[Wol22] Dietmar Wolz. *fcmaes - A Python-3 derivative-free optimization library*. Available at `https://github.com/dietmarwo/fast-cma-es`. Python/C++ source code, with description and examples. 2022.

[Han23] Nikolaus Hansen. *The CMA Evolution Strategy: A Tutorial*. 2023. arXiv: `1604.00772 [cs.LG]`.

[Kor23] Ryan Kortvelesy. *Fixed Integral Neural Networks*. Tech. rep. 3. 2023, pp. 16113–16122. DOI: `10.1109/cvpr52729.2023.01546`. arXiv: `arXiv:2307.14439v3`.