Analytical Integral Global Optimization

Sebastien Labbe SEBASTIEN.LABBE@ENS.PSL.EU
Dept. d'Informatique, Ecole normale supérieure, Paris

Andrea Del Prete ANDREA.DELPRETE@UNITN.IT
Dept. of Industrial Engineering, University of Trento, Italy

Editors: N. Ozay, L. Balzano, D. Panagou, A. Abate

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 several state-of-the-art algorithms when tested on sparse polynomial functions in dimensions up to 170 decision variables.

Keywords: Global-optimization, Integral, Graduated-optimization, Randomized-smoothing

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 Ruder (2016) 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 Kuhlman and Bradley (2019), global cluster structure optimization Hartke (2011), and the training of deep neural networks Larochelle et al. (2009).

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 Bäck and Schwefel (1993), Bayesian optimization Snoek et al. (2012)) 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 Hansen (2023). Deterministic global optimization algorithms exist, and can sometimes provide strong optimality guarantees Jones et al. (1993); Huyer and Neumaier (1999); Lasserre (2001). However, they are typically limited to a specific class of functions (e.g., polynomials, or Lipschitz functions) and they scale badly with the number of decision variables.

Our goal is thus to develop an algorithm that scale better to high-dimensional spaces. 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

© 2025 S. Labbe & A. Del Prete.

methods such as Gaussian Smoothing Gao and Sener (2022), Randomized Smoothing, Gaussian Homotopy Continuation Iwakiri et al. (2022) and Graduated Optimization, some of which have been recently applied to robotics problems Lidec et al. (2021); Suh et al. (2022). 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 focus on 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 Analytical Integral Global Optimization (AIGO).

We implemented AIGO in Python, and evaluated it on several non-convex optimization problems, comparing it with other global optimization methods. Our results show that AIGO has great capabilities of avoiding local minima, outperforming other algorithms in high dimensions (up to 170 variables), either in terms of quality of the solution found, or in terms of speed of convergence.

2. Method

2.1. Problem Statement

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

$$x^\star = \underset{x}{\operatorname{argmin}}\, f(x) \tag{1}$$

where $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is a continuous function, $n \in \mathbb{N}_+$ is the dimension of the pre-image of f, and $x^\star \in \mathbb{R}^n$ is a global minimizer of $f(\cdot)$. We introduce now the notation used throughout the paper.

- 0_n and 1_n are n-dimensional vectors, whose elements are all 0 and 1, respectively.
- $A \subset \mathbb{R}^n$ is an axis-aligned hyper-rectangle, parametrized 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

  $$A(c, w) = \{y \in \mathbb{R}^n | c - w \leq y \leq c + w\},$$

  where inequalities between vectors must be interpreted elementwise.
- $S(w) = \prod_{i=1}^n (2w_i)$ is a function that computes the hyper-volume (or, less formally, size) of an axis-aligned hyper-rectangle $A(c, w)$.

2.2. One-dimensional case

Before introducing our algorithm for the general multi-variate case, we develop its core idea in the 1D case ($x \in \mathbb{R}$), which gives us insight into its working principles. AIGO's key idea is to minimize the integral of f over an interval of fixed width w, with respect to its center c:

$$\underset{c}{\text{minimize}}\ I(c, w) \overset{\text{def}}{=} \int_{c-w}^{c+w} f(x)\, \mathrm{d}\, x, \tag{2}$$

We first solve the problem for a large value of w, and then we slowly decrease it, each time warm-starting the solver with the result of the previous optimization, until we reach $w \approx 0$. Therefore, AIGO alternates between: i) minimizing $I(c, w)$ for a fixed value w, and ii) decreasing w. The efficiency of AIGO depends greatly on the strategies used to minimize $I(c, w)$ and to decrease w. To minimize $I(c, w)$ one can rely on classic gradient-based approaches, but choosing how to decrease w is more complex, and deserves further discussion.

2.2.1. Decreasing the interval width w

To investigate the strategy to reduce w, we follow the classic approach Su et al. (2016) of analysing an Ordinary Differential Equation (ODE) that is the limit of the 1D AIGO algorithm:

$$\begin{cases} \dot{c}(t) = -\partial_c I(c(t), w(t)) = f(c(t) - w(t)) - f(c(t) + w(t)) \\ \dot{w}(t) = -g(c(t), w(t)), \end{cases} \tag{3}$$

where $\partial_c I(\cdot)$ is the partial derivative of $I(\cdot)$ with respect to c. The first equation describes a gradient descent on the cost function of (2), whereas the second equation describes the decrease of the interval width w, which is defined by the positive function $g(\cdot) : \mathbb{R}^2 \to \mathbb{R}_+$. In the following we omit the time dependency to ease the notation.

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 $x^\star$ should never leave $A(c, w)$. However, we will see that this can only be guaranteed if w does not decrease too fast.

Lemma 1 *Assume the initial values of c, w are such that $c(0) - w(0) \leq x^\star \leq c(0) + w(0)$, and that $c(t)$ and $w(t)$ evolve according to the ODE (3). Then, assuming that:*

$$g(c, w) \leq |\dot{c}| \qquad \forall (c, w) : x^\star = c \pm w, \tag{4}$$

we have that:

$$c(t) - w(t) \leq x^\star \leq c(t) + w(t) \qquad \forall t \geq 0 \tag{5}$$

Proof 1 *Since c and w evolve continuously, as long as $c - w < x^\star < c + w$, then $x^\star$ cannot leave $A(c, w)$. We only need to make sure that when $x^\star = c \pm w$, the dynamics (3) ensures that (5) keeps holding. Let us separately analyze the two cases.*

1. *If $x^\star = c + w$, then (5) can hold if and only if $\dot{c} + \dot{w} \geq 0$, or equivalently $\dot{w} \geq -\dot{c}$. Moreover, given (3), we know that $\dot{c} = f(c - w) - f(x^\star) \geq 0$ because $f(x^\star)$ is the minimum of f.*

2. *If $x^\star = c - w$, then (5) can hold if and only if $\dot{c} - \dot{w} \leq 0$, or equivalently $\dot{w} \geq \dot{c}$. Moreover, given (3), we know that $\dot{c} = f(x^\star) - f(c + w) \leq 0$ because $f(x^\star)$ is the minimum of f.*

In conclusion, if when $x^\star = c \pm w$ we have $\dot{w} \geq -|\dot{c}|$, or equivalently $g(c, w) \leq |\dot{c}|$, then (5) holds.

Lemma 1 gives us a sufficient bound on $g(\cdot)$ to ensure that $x^\star$ never leaves $A(c, w)$. A simple way to ensure that such a bound is satisfied is to set:

$$\dot{w} = -\beta |\dot{c}|, \tag{6}$$

with $\beta \in [0,1]$. However, this leads to $\dot{w} = 0$ whenever $\dot{c} = 0$, making the algorithm converge to values that are not a global minimum. To overcome this issue, we can ensure that w decreases at least as a linear function of w:

$$\dot{w} = -\max(\beta|\dot{c}|, \varepsilon w), \tag{7}$$

with $\varepsilon > 0$ being a hyperparameter of the algorithm. In the following theorem we define the conditions under which (7) ensures convergence to a global minimum.

Theorem 2 *Let $x^\star$ be the unique global minimizer of the function f. Assume the initial values of c, w are such that $c(0) - w(0) \le x^\star \le c(0) + w(0)$, and assume that ε satisfies the following bound:*

$$\varepsilon < 2\frac{f(y) - f(x^\star)}{|y - x^\star|} \qquad \forall y \in A(c(t), w(t)), y \neq x^\star, \forall t \ge 0 \tag{8}$$

Then, letting $c(t)$ and $w(t)$ evolve according to the ODE (3) and (7) we have that:

$$\lim_{t\to\infty} c(t) = x^\star, \qquad \lim_{t\to\infty} w(t) \quad = 0 \tag{9}$$

Proof 2 *It is straightforward to see that $(x^\star, 0)$ is an equilibrium for the system (3) and (7). To prove convergence to this equilibrium point, we prove its asymptotic stability using Lyapunov's direct method Lyapunov (1892), which has many analogies with optimization theory Polyak and Shcherbakov (2017). We use the following candidate Lyapunov function:*

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

This function satisfies the two conditions of being always positive, except at $(x^\star, 0)$, where it is null:

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

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

$$\dot{V} < 0 \qquad \forall\, (c, w) \neq (x^\star, 0)$$

From Lemma 1 we know that if $\dot{w} \ge -|\dot{c}|$ when $x^\star = c \pm w$, then $x^\star$ never leaves $A(c,w)$. As long as $x^\star \in A(c,w)$ then we have that $V = 2w$, and so, based on (7), $\dot{V} = 2\dot{w} < 0$, $\forall w \neq 0$. Therefore, proving that $\dot{w} \ge -|\dot{c}|$ when $x^\star = c \pm w$ is sufficient to prove $\dot{V} < 0$.

Given (7), we know that $\dot{w}$ can take one of two values. When $\dot{w} = -\beta|\dot{c}|$, then the condition $\dot{w} \ge -|\dot{c}|$ is trivially satisfied. When instead $\dot{w} = -\varepsilon w$, we need to prove that:

$$\begin{aligned} -\varepsilon w \ge -|\dot{c}| &= -|f(c-w) - f(c+w)| \\ \varepsilon w &\le |f(c-w) - f(c+w)| \\ \varepsilon &\le 2\frac{|f(c-w) - f(c+w)|}{2w} \end{aligned} \tag{10}$$

However, whenever $x^\star = c \pm w$, (10) is trivially implied by (8).

Therefore, in all cases we have $\dot{V} < 0$, as long as ε is sufficiently small to satisfy (8). This guarantees that c and w converge to the minimizer of V, i.e. $c \to x^\star$ and $w \to 0$.

Algorithm 1: AIGO-1D

```
Input: $f, c, w, w_{thr}, \beta, \varepsilon$
1 while $w > w_{thr}$ do
2     $c_{dot} \leftarrow \alpha_c \cdot \partial_c I(c, w)$
3     $c \leftarrow c - c_{dot}$
4     $w \leftarrow w -$ max$(\beta \cdot |c_{dot}|, \varepsilon w)$
5 return $c$
```

This proof shows that using the update rule (7) to decrease w, the ODE (3) makes c converge to $x^\star$. Moreover, the bound (8) says that ε must be upper bounded by the (normalized) difference between $f(x^\star)$ and any other values of f in $A(c, w)$. From this we infer that:

- if there are local minima that are very distant from $x^\star$, but with value very close to $f(x^\star)$, then we need to set ε very small to ensure $\dot{V} < 0$;
- while ε must satisfy (8) to ensure convergence, it can be set arbitrarily small without affecting the convergence speed of the algorithm, which is mainly dictated by the term $\beta|\dot{c}|$;
- if there exist multiple global minima, then the upper bound on ε is zero, therefore the algorithm may work poorly.

A discretized version of this algorithm is summarized in Alg. 1, where α_c can be chosen using a standard line search.

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

$$I(c, w) = \int_{A(c,w)} f(x)\, \mathrm{d}^n x, \qquad \text{and} \qquad S(w) = \prod_{i=1}^{n} (2w_i). \tag{11}$$

While in the 1D case the minimization of the integral was only affecting the variable c, in the multi-dimensional case it can change both c and w. This is because we can now change w without changing the size of $A(c, w)$. Therefore, the minimization problem becomes:

$$\underset{c,w}{\text{minimize}} \quad I(c, w) \qquad \text{subject to} \qquad S(w) = s, \tag{12}$$

Algorithm 2: AIGO N-D

```
Input: $f, c, w, s_{thr}, \beta, \gamma, \zeta$
1 $s \leftarrow S(w)$
2 while $s > s_{thr}$ do
3     $c_{dot} \leftarrow \alpha_c \cdot \partial_c I(c, w)$
4     $c \leftarrow c - c_{dot}$
5     $w \leftarrow w - \beta \cdot |c_{dot}|$
6     $s \leftarrow \min(S(w), s/\gamma)$
7     $w \leftarrow \text{argmin}_y I_p(c, y, s)$
8 return $c$
```

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:

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

Alternatively, we could use the Method of Multipliers (an Augmented Lagrangian scheme), which has better convergence properties (no need for $\zeta \to \infty$ to ensure constraint satisfaction). Starting with an initial value $\mu^{(0)} \in \mathbb{R}$ and a penalty weight $\zeta > 0$:

$$\begin{aligned} c^{(k+1)}, w^{(k+1)} &= \underset{c,w}{\text{argmin}}\ I(c, w) + \frac{\zeta}{2}\left(s - S(w) + \mu^{(k)}\right)^2 \\ \mu^{(k+1)} &= \mu^{(k)} + (s - S(w^{(k+1)})) \end{aligned} \tag{13}$$

2.3.1. Overview of the algorithm

Putting it all together, we summarize the multi-dimensional version of AIGO in Alg. 2. Lines 3, 4, 5 are almost the same as in the 1d algorithm. The only difference is that the term ϵw, which ensured a non-zero decrease of w, is here replaced by line 6, which ensures that s decreases by at least a γ factor at each iteration. Lines 5 and 6 play the role of decreasing s and changing w proportionally to c_{dot}. Then, in line 7 we try to minimize the cost by changing the set's shape without changing its size. To achieve this, we perform gradient-base optimization of 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 7 can be limited to a certain number of iterations. In our tests, we used 2 iterations. We also set a lower bound for the width w in order to better spread out the optimization on all the variables and to prevent AIGO from over-optimizing some variables 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 $[10^5, 10^9]$.
- γ represents the minimum rate of decrease for s, and we used values in $[1.001, 1.05]$.
- s_{thr} is the convergence threshold, which we set to 0.5^n in our tests.

2.3.2. Efficient Integral Evaluation

Finally, we need to discuss the computation of the integral. Even for analytically integrable functions, in general the evaluation of the integral over an axis-aligned hyper-rectangle requires 2^n evaluations of the primitive function. This would make the algorithm inefficient for large values of n. For this reason, we restricted our focus to 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 (multi-variate) polynomials can be represented under this form, where the functions $f_{i,j}(x_j)$ 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:

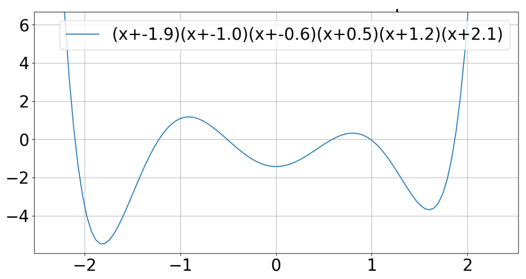
$$I(c, w) = \sum_i a_i \prod_j \int_{c_j - w_j}^{c_j + w_j} f_{i,j}(y)\, \mathrm{d}\, y$$

3. Results

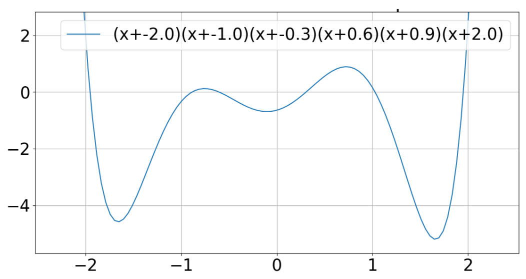
We designed our tests to see how well AIGO performed when trying to minimize non-convex polynomial functions, compared to other four approaches: repetitive gradient descent (RGD) from random initial points, the Covariance Matrix Adaptation Evolution strategy (CMA-ES) Hansen et al. (1995), the Dual Annealing method (DA) Xiang et al. (1997), and the Differential Evolution method (DE) Storn and Price (1997). 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 Labbe (2023) and used 2 outside function calls per loop. The gradient descent step that updates the hyper-rectangle's center c is computed using the `minimize` function from the *scipy* library, with the Sequential Least SQuares Programming (SLSQP) method. The $\text{argmin}_y\, I_p$ step is computed using the same `scipy.minimize` function with the quasi-Newton BFGS method. RGD 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 `advretry.minimize` function of the Fast-CMA-ES library Wolz (2022). Both DA and DE were implemented by the `scipy.optimize` library.



(*a*) Example 1.



(*b*) Example 2.

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

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). This term is simply the sum of n 1D 6-th order polynomials, defined 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 $A(c = 0_n, w = 2.2 \cdot 1_n)$.

3.3. Test Description

Each function f was optimized using the 5 different algorithms mentioned above, using $A(0_n, 2.2 \cdot 1_n)$ as bounds. We first ran AIGO to completion. All the other algorithms were then given the same amount of time as AIGO, running on a single CPU core. For each function we also ran an additional test, simply used to normalize the results. We randomly sampled points in $A(0_n, 2.2 \cdot 1_n)$ for the same amount of time as AIGO and used the average f_{avg} and minimum value f_{min} of f at the sampled points to normalize the best result found by the other algorithms $f^\star$ in this way:

$$\text{normalized score} = \frac{f^\star - f_{avg}}{f_{avg} - f_{min}} \tag{14}$$

A score of 0 means that the algorithm has found a solution equal to the average value found by random sampling f_{avg}. A score of -1 instead means that the algorithm has found the same minimum found by random sampling f_{min}. Lower scores mean that the algorithm outperformed random sampling.

3.4. Discussion of the Results

Fig. 2 shows the average normalized score and its standard deviation after running each algorithm on 10 randomly generated functions for each size of the decision variables. Unsurprisingly, as soon as the dimension is larger than 2, all algorithms outperform 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. For small dimensions all algorithms perform similarly,

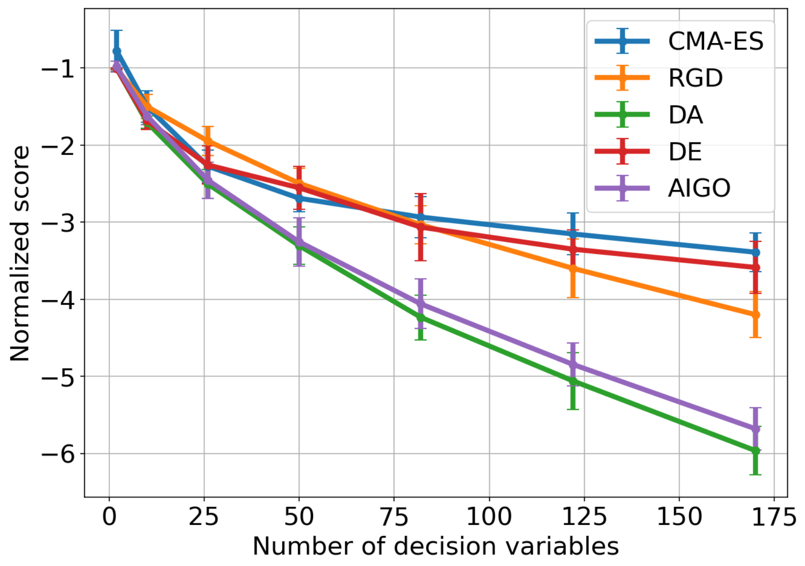


Figure 2: Normalized score as a function of the number of decision variables.

probably due to the relatively small number of local minima. However, as the problem dimension grows above 50, AIGO and DE vastly outperformed the others, with DE being slightly better than AIGO on average.

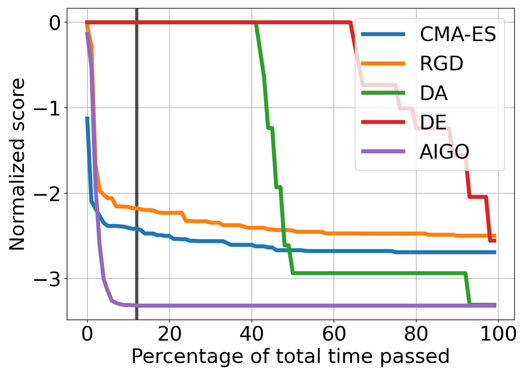
Fig. 3 shows how fast the different algorithms have been able to converge to the solution for different problem sizes. Interestingly, AIGO was consistently able to converge faster than all the other algorithms. The black vertical line shows the moment when AIGO's solution stopped changing, even though the algorithm kept running much longer to meet the convergence threshold on s. This suggests that by setting the convergence threshold s_{thr} to a higher value, AIGO could convergence almost 10 times faster, so outperforming all the other algorithms.

This shows that this simple implementation of AIGO seems to perform as well as some state-of-the-art algorithms and is less effected by the growing dimensions, which is translated in a faster comparative convergence as the number of local minima increases.

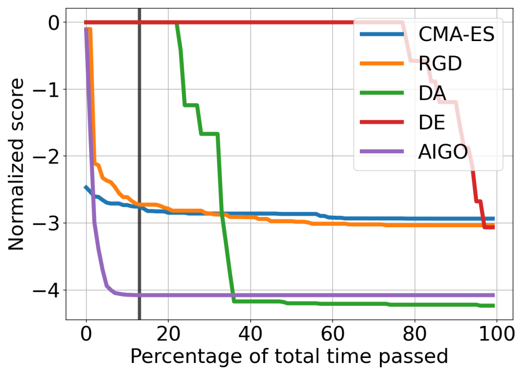
4. Conclusions

We have presented AIGO, a novel unconstrained optimization algorithm for non-convex functions. While many 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.

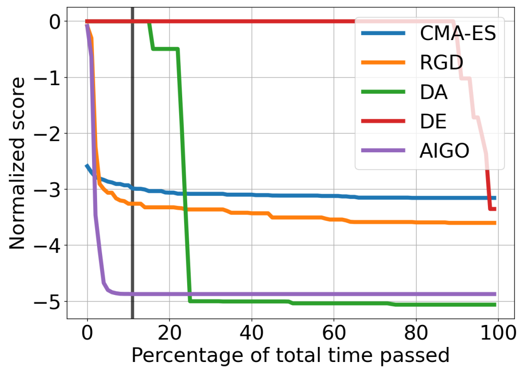
We were able to minimize polynomial functions with up to 170 variables, computing their integral over axis-aligned hyper-rectangles. We have compared AIGO with other global optimization algorithms. AIGO outperformed most of the other algorithms for problem sizes above 10, empirically proving the interest of this idea. For larger problem size, AIBO almost matched the performance of the best algorithms in terms of solution quality, while converging to that solution in $\approx$ 15% of the time.



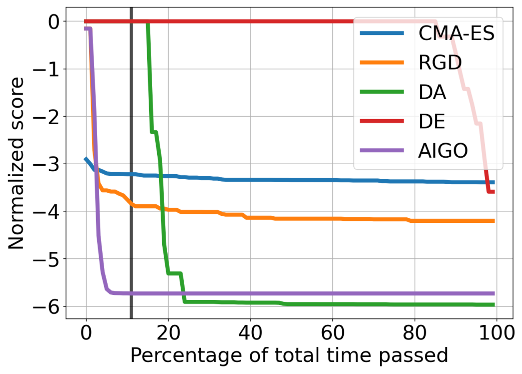
(*a*) 50 variables



(*b*) 82 variables



(*c*) 122 variables



(*d*) 170 variables

Figure 3: Average normalized score as a function of computation time.

In the future, we would like to improve AIGO in several aspects. Our results have shown that AIGO typically converges to the final solution much before meeting our convergence criterium. This suggests that a better convergence criterium could be used to terminate AIGO earlier, saving precious computation time.

In this paper we have focused on 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* Kortvelesy (2023); Lloyd et al. (2020), which are networks that provide a direct evaluation of the exact integral of a function. However, integrating over an N-dimensional hyperbox would require evaluating the network 2^N times.

This work has only explored hyper-boxes as regions over which integrals are computed. It is however 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

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

Katelyn Gao and Ozan Sener. Generalizing gaussian smoothing for random search, 2022.

Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023.

Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *ICGA*, pages 57–64, 1995.

Bernd Hartke. Global optimization. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(6):879–887, 2011.

Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14:331–355, 1999.

Hidenori Iwakiri, Yuhang Wang, Shinji Ito, and Akiko Takeda. Single loop gaussian homotopy method for non-convex optimization, 2022.

Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of optimization Theory and Applications*, 79:157–181, 1993.

Ryan Kortvelesy. Fixed Integral Neural Networks. Technical Report 3, 2023.

Brian Kuhlman and Philip Bradley. Advances in protein structure prediction and design. *Nature Reviews Molecular Cell Biology*, 20(11):681–697, 2019.

Sebastien Labbe. Gibo - a python-3 global integral-based optimization. Available at https://gitlab.com/SebastienLabbe1/gibo, 2023. Python source code.

Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *Journal of machine learning research*, 10(1), 2009.

Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

Quentin Le Lidec, Louis Montaut, Cordelia Schmid, Ivan Laptev, and Justin Carpentier. Leveraging Randomized Smoothing for Optimal Control of Nonsmooth Dynamical Systems. Technical report, 2021. URL http://arxiv.org/abs/2203.03986.

Steffan Lloyd, Rishad A Irani, and Mojtaba Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2991966.

A.M. Lyapunov. The general problem of motion stability. *Annals of Mathematics Studies*, 17, 1892.

Boris Polyak and Pavel Shcherbakov. Lyapunov Functions: An Optimization Theory Perspective. In *IFAC-PapersOnLine*, volume 50, pages 7456–7461, 2017. doi: 10.1016/j.ifacol.2017.08.1513.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

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

Hyung Ju Terry Suh, Tao Pang, and Russ Tedrake. Bundled Gradients Through Contact Via Randomized Smoothing. *IEEE Robotics and Automation Letters*, 7(2):4000–4007, 2022. ISSN 23773766. doi: 10.1109/LRA.2022.3146931.

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

Yang Xiang, DY Sun, W Fan, and XG Gong. Generalized simulated annealing algorithm and its application to the thomson model. *Physics Letters A*, 233(3):216–220, 1997.