# A comparative analysis of three methods used for numerical optimization: Genetic Algorithms, Hill Climbing and Particle Swarm Optimization

**Mădălina-Alina Racoviță and Cristian Vîntur[1]**

[1]***Masters of Computational Optimization**, 1st Year*
*Faculty of Computer Science, **'Alexandru Ioan Cuza'** University of Iași, Romania*

## Abstract

This paper explores some methods used in the purpose of numerical optimization. It includes the description of the functions considered for the optimization task along with a detailed overview on the methods used to achieve that. On this line, Hill Climbing, a general version of a genetic algorithm, two hybridizations of the previous two algorithms and Particle Swarm Optimization are presented individually, along with implementation details and with the manner in which they performed in the numerical context emphasized. A comparative analysis of all is included, by punctuating similarities, differences and an overview on the obtained results.

*Keywords:* **Numerical optimization, Hill Climbing, Genetic algorithms, Particle Swarm Optimization.**

## 1 Introduction

Determining the minimum of a function (or, more generally, the optimum of a function) is a very common task which occurs in real-world problems. Many **Machine Learning algorithms**, such as **SVM** or **linear regression**, require minimizing a loss function as one of the steps in determining a model. *Sometimes, the optimum value of a function does not have a closed form formula or computing it is very time consuming*. In these scenarios, **an iterative method for approximating the optimum value is used**. All the methods further described belong to this category.

The second chapter presents the functions considered for the task of minimization. Except from De Jong function, **all of them are hard to optimize** because there are **numerous local minima and only a single a point or surface in that n-dimensional space identified as a global minimum**.

The first methods described were **Hill Climbing** and **a general schema of a Genetic Algorithm**. Along with them, **two hybrid versions** which are basically a combination of those two previously mentioned algorithms are presented. The configuration of the layers for the hybrid version is included, along with the pseudocode and implementation details regarding parameters and number's representation. The article continues with the presentation of the **Particle Swarm Optimization** method which tries to simulate the behaviour of organisms in biological systems, such as bird and fish banks or ant colonies.

For each method individually, the experiments pursued in order to emphasize their performance are presented. There are punctuated the configurations which performed best. Visualization graphics are also presented in order to underline the evolution of the particle's population during a specific number of generations. In the case of the PSO there are included a couple of frames

***A comparative analysis of three methods used for numerical optimization: Genetic Algorithms, Hill Climbing and Particle Swarm Optimization (2020)***

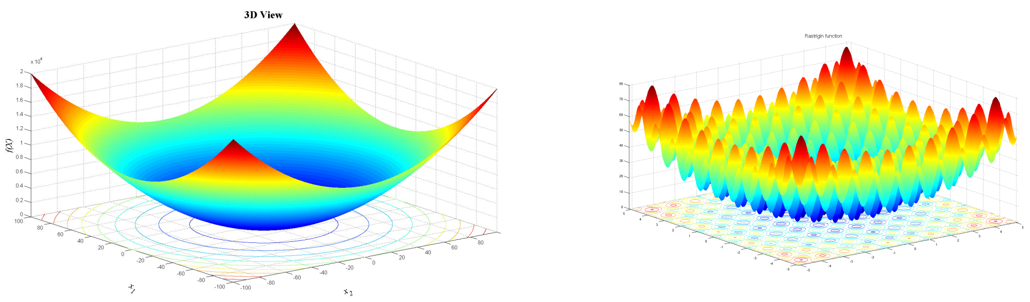
Mădălina-Alina Racoviță, Cristian Vîntur

from a 3D animation that covers the migration of the particles to global minima while a number of iterations has passed.

The paper covers as well **the hyper-parameter influence in the convergence of these methods**. In the case of the PSO a grid search through different configurations of the three specific weights is emphasized along with screenshots from 3d interactive plots.

The next section presents **a comparative analysis between the considered algorithms by punctuating similarities, differences and performance achievements**.

## 2 Functions to be minimized



(a) The sphere model [2].

(b) Rastrigin's function illustration [4].

**Figure 1.** Illustrations for two of the functions to be minimized

### 2.1 De Jong function [1]

This was the simplest test function considered. It is also known as the sphere model. It is continuous, convex and unimodal. The function is defined as:

$$f_1(x) = \sum_{i=1}^{n} x[i]^2$$

where $-5.12 \leq x[i] \leq 5.12, i = \overline{1,n}$. **The global minimum** is $f(x) = 0$, reached for $x[i] = 0, i = \overline{1,n}$.

### 2.2 Rastrigin's function [1]

Rastrigin's function is based on function 1 with the addition of cosine modulation to produce many local minima. Thus, the test function is highly multimodal. However, the location of the minima are regularly distributed. The function is defined as:

$$f_2(x) = 10n + \sum_{i=1}^{n} (x[i]^2 - 10\cos(2\pi x[i]))$$

where $-5.12 \leq x[i] \leq 5.12, i = \overline{1,n}$. **The global minimum** is $f(x) = 0$, reached for $x[i] = 0, i = \overline{1,n}$.

### 2.3 Griewangk's function [1]

The function is defined as:

$$f_3(x) = \sum_{i=1}^{n} \frac{x[i]^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x[i]}{\sqrt{i}}\right) + 1$$

where $-600 \leq x[i] \leq 600, i = \overline{1, n}$. **The global minimum** is $f(x) = 0$, reached for $x[i] = 0, i = \overline{1, n}$.

### 2.4 Rosenbrock's valley [1]

The global optimum of this function is inside a long, narrow, parabolic shaped flat valley [1]. The functions is defined as:

$$f_4(x) = \sum_{i-1}^{n-1} 100(x[i+1] - x[i]^2)^2 + (1 - x[i])^2$$

where $-2.048 \leq x[i] \leq 2.048, i = \overline{1, n}$. **The global minimum** is $f(x) = 0$, reached for $x[i] = 0, i = \overline{1, n}$.

### 2.5 Six-hump camel back function [1]

Within the bounded region of this function are six local minima, two of them are global minima.

$$f_5(x) = (4 - 2.1x[1]^2 + x[i]^{\frac{4}{3}})x[1]^2 + x[1]x[2] + (-4 + 4x[2]^2)x[2]^2$$

where $-3 \leq x[1] \leq -3$ and $-2 \leq x[1] \leq -2$. **The global minimum** is $f(x[1], x[2]) = -1.0316$, reached for: $(x[1], x[2]) = (-0.0898, 0.7126), (0.0898, -0.7126)$.

## 3 Methods for numerical optimization

### 3.1 Hill Climbing and genetic algorithms

The **Hill Climbing algorithm** [5] is an iterative method which implements a local search in the given definition domain. It is also named **Iterative Improvement**, since from an iteration to another the new Hill Climber's state is a better one than the past one. It was used the iterative version in which the Hill Climber is restarted having the purpose of increasing the grade of exploration in the searching space. This algorithm is based on a function that searches for the first (in the case of **First Improvement**) or for the best (in the case of **Best Improvement**) candidate from all the neighbours of the current state. This best candidate will become the current reference individual for the algorithm.

A **genetic algorithm** [6] is a probabilistic algorithm that maintains a population of representations of the candidates for the solution. The estate of the algorithm evolves during a generation or a iteration, under the velocity of a fitness function which measures the individual reward. The population is built of multiple candidates, each candidate being an $x$ array of chromosomes. By generations' evolutions the candidates become better and better in minimizing the given function. The genetic algorithm that was build uses **elitism or roulette wheel as selections strategies**, or both, this aspect being configurable from a constants file. The recombination of candidates is made by **single point crossover and simple mutations**.

### Genetic algorithm's parameters

The genetic algorithm had the following parameters:

$$\texttt{probability of mutation} = 0.1$$
$$\texttt{probability of crossover} = 0.7$$
$$\texttt{precision} = 10^{-4}$$
$$\texttt{pop\_size} = 15$$
$$\texttt{number of generations} = 6000$$
$$\texttt{number representation} = \texttt{Gray code}\ [3]$$

**Table 1. Results for the Hill Climbing**

| Max | Mean | Min | n | function |
|---|---|---|---|---|
| 75.62046 | 63.87924 | 38.811379 | 30 | Rastrigin |
| 36.81376 | 20.03239 | 9.95037 | 10 | Rastrigin |
| 19.89918 | 11.50854 | 4.97488 | 5 | Rastrigin |
| 0.00768 | 0.00196 | 0.00068 | 30 | De Jong |
| 73.05537 | 8.61985 | 0.04115 | 30 | Rosenbrock |
| 0.13856 | 0.0404 | 0.00427 | 30 | Griewangk |

**Table 2. Results for the genetic algorithm**

| Elitism | Roulette wheel | Min | Max | Mean | n | function |
|---|---|---|---|---|---|---|
| 60% | 40% | 47.5536 | 60.6164 | 54.30930 | 30 | Rastrigin |
| 100% | 0% | 43.4951 | 58.1014 | 48.5661 | 30 | Rastrigin |
| 100% | 0% | 0.4707 | 1.1523 | 0.8604 | 30 | De Jong |
| 100% | 0% | 3.2665 | 4.5279 | 3.937 | 30 | Griewangk |
| 100% | 0% | 37.533 | 95.117 | 56.612 | 30 | Rosenbrock |
| 100% | 0% | −1.031608 | −1.031628 | −1.03162 | 2 | SHC |

## 3.2 Two hybrid versions that use a Hill Climber and a genetic algorithm

There were implemented **two versions of hybridizations**. **The first one** *starts with a randomly generated population with which a genetic algorithm is trained*. The genetic algorithm will return an array $x$ which best minimizes the considered function and a minimum $f(x)$ value. That specific $x$ will be *input for a Hill Climber*, which will return the best descendant $x$ and the set of $x$ arrays whose values are constructing the pool of attraction. The $f$ value corresponding to the best Hill Climber descendant will be the minimum value for the function. The pseudocode for this version of algorithm is written bellow.

**Algorithm 1** Hybrid GA + Hill Climbing

```
1: procedure GA_HC_HYBRIDIZATION
2:     $pop \leftarrow$ generate_population()
3:     $x\_ga, min\_ga \leftarrow$ genetic_algorithm($pop$)
4:     $best\_descendant, xes\_set \leftarrow$ hill_climbing($population = x\_ga$)
5:     $min\_after\_hc \leftarrow$ f($best\_descendant$)
6:     return $min\_after\_hc$
7: end procedure
```

**The second version of hybridization** starts similarly with 10 iterations of a Hill Climber, each iteration having as correspondent a randomly generated population. Among all iterations, a set of cardinal $pop\_size$ of arrays $x$ which will minimize best the given $f$ function will be chosen. This will represent the input population for the genetic algorithm. The minimum value returned by this last procedure call will be the minimum value for the $f$ function. The pseudocode is described down below.

**Algorithm 2** Hybrid Hill Climbing + GA

```
1: procedure HC_GA_HYBRIDIZATION
2:     $population\_ga \leftarrow$ hill_climbing($iterations = 10$)
3:     $x\_ga, min\_ga \leftarrow$ genetic_algorithm($population\_ga$)
4:     return $min\_ga$
5: end procedure
```

## Experimental results

Some experiments were made in order to see how well the Genetic Algorithm or the Hill Climber perform individually and after that as a hybrid, in both of the hybridization versions. The data from the bellow tables illustrates the performance of those algorithms previously enumerated. Each algorithm was run **30 times having the same input** in order **to emphasize the non deterministic behaviour** in the case of the genetic algorithm and **30 times on different starting points** to emphasize the trajectory nature of the Hill Climber.

**Table 3. Results for the GA + Hill Climber hybridization**

| Elitism | Roulette wheel | Min | Max | Mean | n | function |
|---|---|---|---|---|---|---|
| 100% | 0% | 15.9226 | 32.8346 | 25.606 | 30 | Rastrigin |
| 100% | 0% | 1.99 | 6.9662 | 4.7762 | 10 | Rastrigin |
| 100% | 0% | 0.000045 | 1.9899 | 1.0615 | 5 | Rastrigin |

**Table 4. Results for the Hill Climber + GA hybridization**

| Elitism | Roulette wheel | Min | Max | Mean | n | function |
|---|---|---|---|---|---|---|
| 100% | 0% | 31.65191 | 47.49870 | 39.28243 | 30 | Rastrigin |
| 100% | 0% | 4.12957 | 9.77823 | 6.88900 | 10 | Rastrigin |
| 100% | 0% | 0.00042 | 2.26594 | 1.19979 | 5 | Rastrigin |

#### Results interpretation

It is noticed that a genetic algorithm that uses as a selection method 100% elitism performs better than an algorithm which select a part of the candidates from the new population with roulette wheel, aspect which is intuitive and normal. By choosing the best $pop\_size$ candidates to represent the new population in a new generation the convergence is increased. **The roulette wheel by having that randomness** aspect has a **slower convergence** and this is why for reaching the same results as a GA with 100% elitism obtains, **a bigger number of generations is required**.

It was observed that **the minimum for Six Hump Camel function is reached** and for the rest of the functions the results are leading to the real global minimum. The Hill Climber obtains a good result for the Rastrigin function with 30 dimensions, taking into account the fact that *the genetic algorithm evolves during 6000 generations in comparison with the Hill Climber that runs only on that specific candidate given as input*. Additionally, Hill Climbing obtains better results in minimizing De Jong, Rosenbrock and Griewangk with $n = 30$ than the genetic algorithm.

For the hybrid versions the minimization experiments were made only on Rastrigin function with fluctuating number of dimensions, since Rastrigin was one of the most complex test functions. The results show that **the first hybrid version performs better and reaches the minimum of approximately 16** which is the closest one to global minima of 0.

## 3.3 Particle Swarm Optimization

Along with Ant Colony Optimization, **Particle Swarm Optimization** [7], these algorithms form the class of he class of **Swarm Intelligence** [8] algorithms.

**Swarm Intelligence** is *a new computational paradigm based on the study of the collective behavior in decentralized, self-organized systems*. Problem solving takes place at a superior level by having as a foundation a collection of idealized agents, without having the individual intention from their behalf. In other words, **individual agents are not aware that they are solving a problem, but the collective interaction leads to solving it**. The exchange of information between members of the same species offers an evolutionary advantage: group members can benefit from the previous discoveries and experience of all other members.

**PSO** was discovered (**Kennedy and Eberhart, 1995**) by simulating the movement of individuals in a flock of birds or in fish banks. It was initially conceived as a method of optimizing continuous nonlinear functions but was later adapted to solve permutation problems, integer programming, constraint problems, optimization in dynamic or multi-objective environments.

#### Algorithm's description

**Particle swarm optimization** (PSO) [8] is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an $n$-dimensional space. **Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles**. Particles then move through the solution space, and are evaluated according to some fitness criterion after each timestep. Over time, **particles are accelerated towards those particles within their communication grouping which have better fitness values**. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that *the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.*

## Algorithm's parameters

- `precision` $\rightarrow 10^{-9}$
- `dimensions for the function` $\rightarrow 30$
- `w1 - weight of inertia` $\rightarrow$ *ensures the balance between exploration / exploitation; it is also used as a decreasing weight over time (as in Simulted Annealing)*
- `w2 - cognitive parameter` $\rightarrow$ *the tendency to duplicate one's own past actions that was proved to be successful*
- `w3 - social parameter` $\rightarrow$ *the tendency to follow the success of other individuals*
- `maximum velocity allowed` $\rightarrow$ *indirectly affects the ability to explore, reducing the value of the speed below a predefined allowed limit*

## Pseudocode [9]

**Algorithm 3** Particle Swarm Optimisation Algorithm

```
procedure PSO($b_{lo}, b_{up}$)
    $x \leftarrow U(b_{lo}, b_{up})$    ▷ Initialize the particle's position with a uniformly distributed random vector
    $v \leftarrow U(-|b_{up} - b_{lo}|, |b_{up} - b_{lo}|)$    ▷ Initialize the particle's velocity
    for each particle $i \leftarrow 1, \ldots, S$ do
        $p_i \leftarrow x_i$    ▷ Initialize the particle's best known position to its initial position
        if $f(x_i) < f(p_i)$ then
            $p_i \leftarrow x_i$    ▷ Update the particle's best known position
        end if
    end for
    while a termination criterion is not met do
        for each particle $i \leftarrow 1, \ldots, S$ do
            for each dimension $d \leftarrow 1, \ldots, n$ do
                $r_p, r_g \leftarrow U(0, 1)$    ▷ Pick random numbers
                $v_{i,d} \leftarrow \omega v_{i,d} + \phi_p r_p (p_{i,d} - x_{i,d}) + \phi_g r_g (g_d - x_{i,d})$    ▷ Update the particle's velocity
            end for
            $x_i \leftarrow x_i + v_i$    ▷ Update the particle's position
            if $f(x_i) < f(p_i)$ then
                $p_i \leftarrow x_i$    ▷ Update the particle's best known position
                if $f(p_i) < f(g)$ then
                    $g \leftarrow p_i$    ▷ Update the swarm's best known position
                end if
            end if
        end for
    end while
    return $g$    ▷ Return the swarm's best known position
end procedure
```

## Visualization

The illustrations enumerated below are exported as frames from a 3d animation that emphasize the migration of the particles during generations for the Rastrigin function. On the $Ox$ and $Oy$ axis were ploted the first 2 dimensions of the considered candidates (i.e. $x[0]$ and $x[1]$ elements from the position arrays). On the $Oz$ axis is the minimum found in the current iteration. It was shown how while some generations pass, the each particle is influenced by the collective reward and they keep migrating to a minimum value (in the case of Rastrigin function to a closer value to the zero global minima). The parameters considered for the animation were:

`pop_size` $\longrightarrow$ 250

`generations_number` $\longrightarrow$ 700

`n` $\longrightarrow$ 30

`w1 - weight of inertia` $\longrightarrow$ 0.5

`w2 - cognitive parameter` $\longrightarrow$ 0.5

`w3 - social parameter` $\longrightarrow$ 4

`maximum_velocity_allowed` $\longrightarrow$ 1

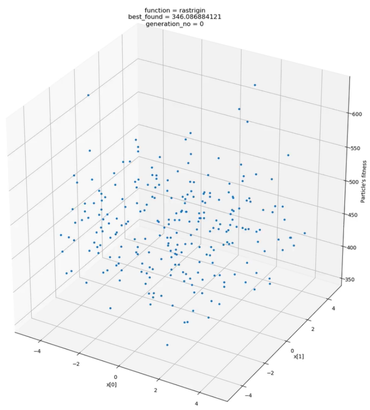
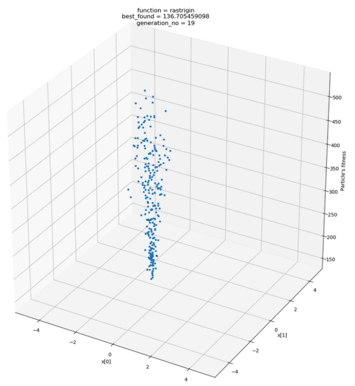
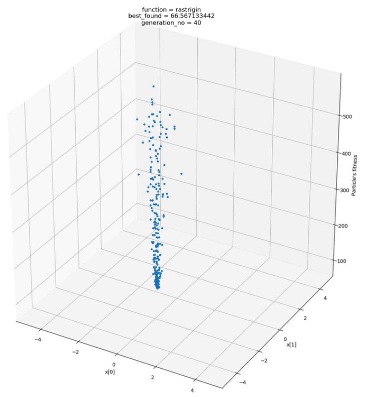


**Figure 2.** In the first frame it can be noticed the randomness in the distribution of the particles. In the first iteration the best found for the Rastrigin fitness function was 346.086884121.

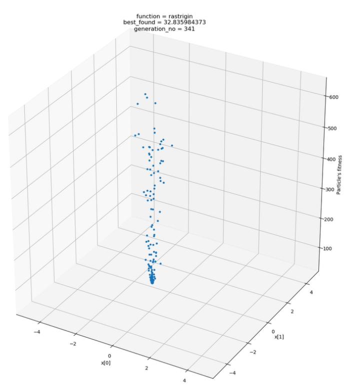


(a) The particle position for the 19th generation. The best minimum value found is 136.70549098.

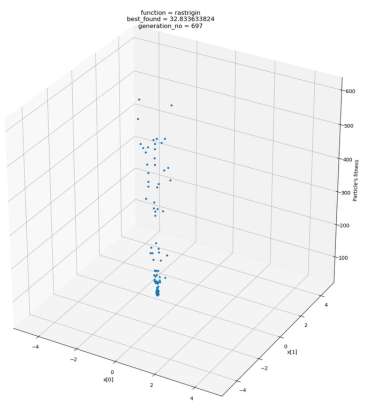


(b) The particle position for the 40th generation. The best minimum value found is 66.567133442.

**Figure 3.** In the next 40 iterations the particles start to migrate to a minimum value.



(a) The particle position for the 341th generation. The best minimum value found is 32.835984373.



(b) The particle position for the 697th generation. The best minimum value found is 32.833633824.

**Figure 4.** The improvements of the PSO algorithm are now more than notable. There are only a few particles that have to migrate to a better minimum value.

## Experimental results

**The experiments were started by performing a grid search on three hyperparameters** of the algorithm, namely the three different weights: **inertia, cognitive and social**. The possible values for each of these weights were $\{0.5, 1, 2.05, 4\}$, resulting in 64 possible combinations. Out of all of them, the first two with best results were chosen for the following experiments.

With the exception of Griewangk function, the maximum value allowed for velocity was around 10% of the domain length of one axis. The following table summarizes the hyperparameters used by the PSO algorithm for each function.

**Table 5. Hyperpameters used by the PSO algorithm.**

| Function | n | Population | Generations | Inertia | Cognitive | Social | Max velocity |
|---|---|---|---|---|---|---|---|
| Rastrigin | 30 | 250 | 700 | 0.5 | 0.5 / 1 | 0.5 | 1 |
| Griewangk | 30 | 250 | 700 | 0.5 | 0.5 / 1 | 0.5 | 12 |
| Rosenbrock | 30 | 250 | 700 | 0.5 | 0.5 / 1 | 0.5 | 400 |
| SixHump | 2 | 15 / 50 / 100 | 50 | 0.5 | 0.5 / 1 | 0.5 | 1 |

For each combination of parameters from the table 5, **the algorithm was ran** 30 **times independently**. The following table presents the minimum, mean and maximum value found over these runs.

There was observed that **PSO obtained very good results in the case of Griewangk and Six-Hump functions**. Although the minimum value for the Rosenbrock function, in the case of a cognitive weight equal to 1, is very close to the optimal value of 0, the mean value of the experiments suggests that this happened by chance and the algorithm tends to block in a local minimum around 23. For the **Rastrigin** function, which has the minimal value equal to 0, the best value obtained was around 12 and the mean of the best values was 36.

**Table 6.** **Results for PSO algorithm.**

| Function | Population | Cognitive | Min | Mean | Max |
|---|---|---|---|---|---|
| Rastrigin | 250 | 0.5 | 13.92942 | 36.71395 | 69.64702 |
| Rastrigin | 250 | 1 | 11.93956 | 30.08105 | 52.73290 |
| Griewangk | 250 | 0.5 | 0.00246 | 0.01411 | 0.03200 |
| Griewangk | 250 | 1 | 0.00248 | 0.01498 | 0.06119 |
| Rosenbrock | 250 | 0.5 | 22.58334 | 34.46527 | 208.33797 |
| Rosenbrock | 250 | 1 | 0.00058 | 23.41544 | 27.06576 |
| SixHump | 15 | 0.5 | -1.03162 | -1.03162 | -1.03156 |
| SixHump | 15 | 1 | -1.03162 | -1.03159 | -1.03126 |
| SixHump | 50 | 0.5 | -1.03162 | -1.03162 | -1.03162 |
| SixHump | 50 | 1 | -1.03162 | -1.03162 | -1.03162 |
| SixHump | 100 | 0.5 | -1.03162 | -1.03162 | -1.03162 |
| SixHump | 100 | 1 | -1.03162 | -1.03162 | -1.03162 |

Using the value of 1 for the cognitive weight improved all results in the case of Rastrigin and Rosenbrock functions and it did not worsen the rest of them.

Another important aspect to notice is that there were **obtained values very close to the global minimum for SixHump function, even though the population of particles was smaller**. The experiments pursued included trying the usage of a smaller population in the other cases, but the results were much worse. Two possible reasons are the fact that this function has only two dimensions, compared to 30 for the others, and that it has only six local minima, two of which are global.

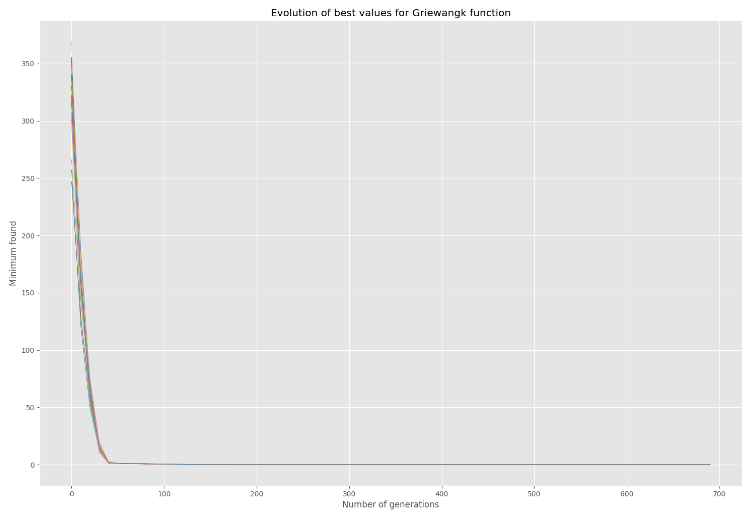


**Figure 5.** The evolution of the minimum values (of each of the 30 populations considered in an experiment) found while generations pass for **Griewangk** function by using the cognitive weight 0.5 and a pop_size of 250.

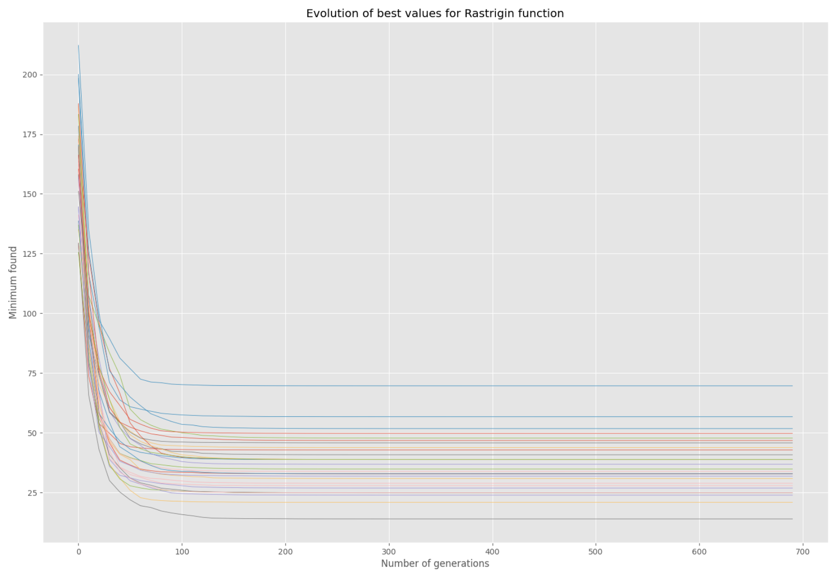


**Figure 6.** The evolution of the minimum values (of each of the 30 populations considered in an experiment) found while generations pass for **Rastrigin** function by using the cognitive weight 0.5 and a pop_size of 250.

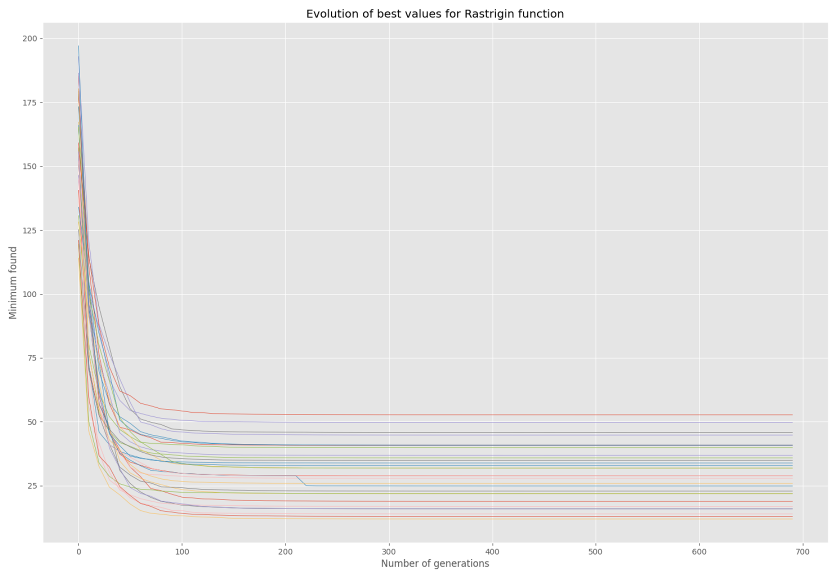


**Figure 7.** The evolution of the minimum values (of each of the 30 populations considered in an experiment) found while generations pass for **Rastrigin** function by using the cognitive weight 1 and a pop_size of 250.

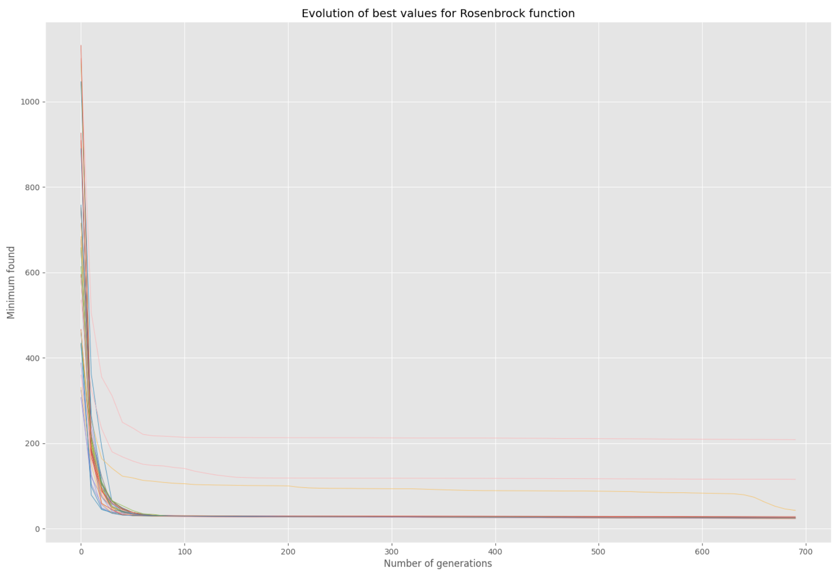


**Figure 8.** The evolution of the minimum values (of each of the 30 populations considered in an experiment) found while generations pass for **Rosenbrock** function by using the cognitive weight 0.5 and a pop_size of 250.

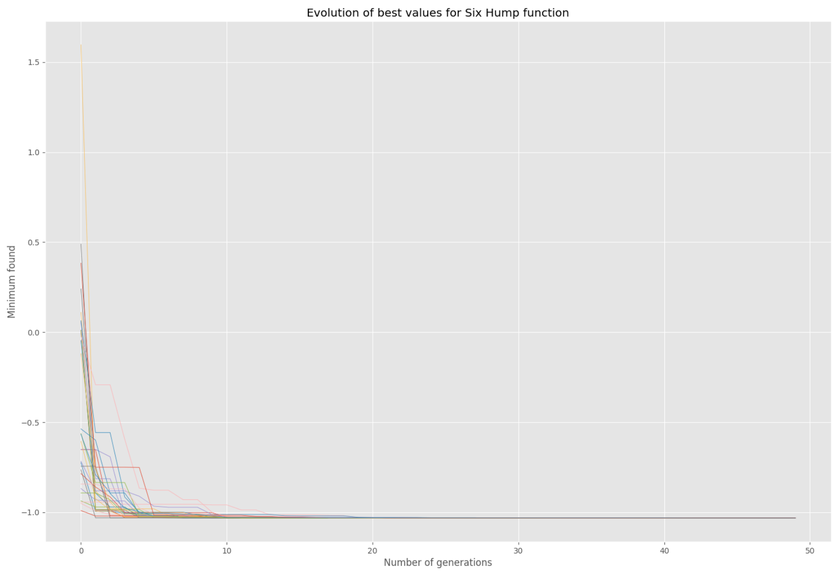


**Figure 9.** The evolution of the minimum values (of each of the 30 populations considered in an experiment) found while generations pass for **Six Hump Camel Back** function by using the cognitive weight 1 and a pop_size of 15.

**The influence of different parameters over the final result**

It was observed that, **for a population size of** 250**, the algorithm takes about** 50 – 150 **generations to converge to a value (although not always optimal)**. *The only exception is the SixHump function* which converges much faster for the reasons presented above.

Although the results are not presented in this paper, *for the first three function we run the algorithm with the same parameters, but with a smaller population (100 individuals), and the results were much worse*.

From the results presented in table 6, we can conclude that **using a cognitive weight of** 1 **instead of** 0.5 **improves the results significantly** for two functions if the rest of the parameters were unchanged. This type of influence is presented in more details in the following tabel.

Table 7 presents the Min / Mean / Max values obtained by running the PSO algorithm 30 times with the corresponding parameters on the Rastrigin function. For the other parameters, not shown here, we used a population size of 100 and run 200 generations. Maximum velocity was 0.5. Although we performed the grid search for only one function, we also performed it for the Griewangk function as well and the top combinations were almost the same (best two combinations for Rastrigin were also top 3 for Griewangk).

**Table 7. Results for the hyperparameter grid search.**

| Function | Inertia | Cognitive | Social | Min | Mean | Max |
|---|---|---|---|---|---|---|
| rastrigin | 0.5 | 0.5 | 4.0 | 10.28066 | 20.80699 | 38.06427 |
| rastrigin | 0.5 | 1.0 | 4.0 | 11.24556 | 21.29217 | 33.39683 |
| rastrigin | 0.5 | 2.05 | 4.0 | 13.21287 | 22.83945 | 36.09134 |
| rastrigin | 0.5 | 4.0 | 1.0 | 15.10297 | 24.28610 | 39.79170 |
| rastrigin | 0.5 | 4.0 | 2.05 | 13.59703 | 24.74947 | 40.70649 |
| rastrigin | 0.5 | 2.05 | 2.05 | 11.32813 | 25.44473 | 41.38837 |
| rastrigin | 1.0 | 2.05 | 2.05 | 16.64983 | 29.89028 | 40.75223 |
| rastrigin | 1.0 | 1.0 | 2.05 | 15.99256 | 31.04287 | 52.08850 |
| rastrigin | 0.5 | 2.05 | 1.0 | 19.44248 | 32.10813 | 56.37964 |
| rastrigin | 1.0 | 2.05 | 1.0 | 17.45419 | 32.42274 | 53.92702 |
| rastrigin | 0.5 | 1.0 | 2.05 | 18.87294 | 32.49733 | 71.73266 |
| rastrigin | 0.5 | 1.0 | 1.0 | 21.03334 | 33.43216 | 48.56972 |
| rastrigin | 1.0 | 1.0 | 4.0 | 23.36179 | 35.36028 | 60.93375 |
| rastrigin | 1.0 | 4.0 | 2.05 | 23.63737 | 36.38836 | 46.42380 |
| rastrigin | 1.0 | 0.5 | 2.05 | 16.30778 | 36.47437 | 59.13847 |
| rastrigin | 1.0 | 1.0 | 1.0 | 17.69475 | 36.51629 | 53.81774 |
| rastrigin | 0.5 | 4.0 | 4.0 | 20.94198 | 36.80604 | 83.32819 |
| rastrigin | 0.5 | 0.5 | 2.05 | 25.89729 | 39.06804 | 54.53574 |
| rastrigin | 1.0 | 4.0 | 1.0 | 18.62856 | 39.12462 | 73.88529 |
| rastrigin | 1.0 | 0.5 | 0.5 | 23.43372 | 39.83342 | 56.16616 |

The rows from the table were conveniently sorted by the Mean value. We can easily extract some useful information, such as what are the best combinations of weights to further use. Because all the values for inertia weight are 0.5 and 1 we conclude that bigger values of 2.05 and 4 were not suitable for this experiment. Also, the first three combinations have a social weight of 4, which means that going to the global current minimum is more important than going to the particle best minimum found.

Even with this grid search optimization, it is likely that there are other combinations of hyperparameters, not taken into account by us, which will perform even better.

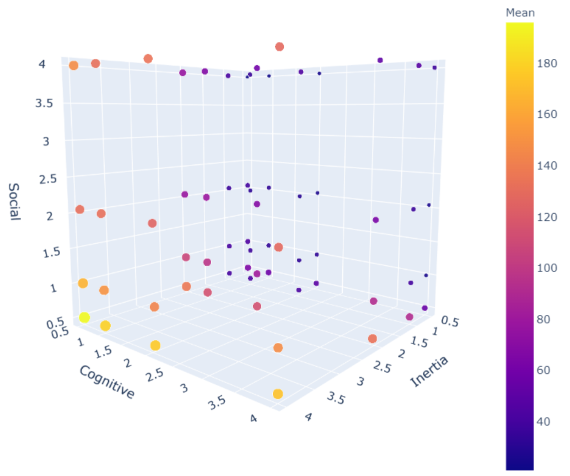


**Figure 10.** Section within a 3d interactive graphic that illustrates the effect of different configurations for inertia / cognitive/ social parameters over the PSO results for the Rastrigin function.

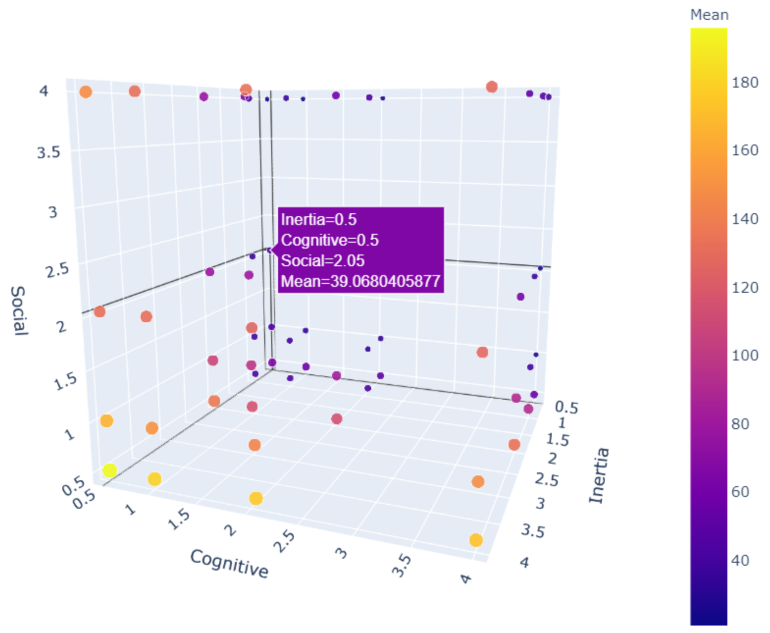


**Figure 11.** Section within a 3d interactive graphic that illustrates the effect of different configurations for inertia / cognitive/ social parameters over the PSO results for the Rastrigin function: **one of the best configurations found**

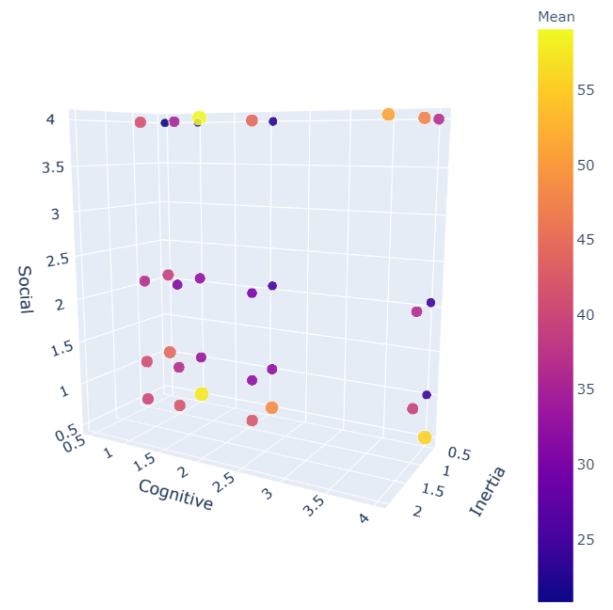


**Figure 12.** Section within a 3d interactive graphic that illustrates the effect of different configurations for inertia / cognitive/ social parameters over the PSO results for the Rastrigin function. **There were selected only the configurations that lead to an average minimum value smaller than 60.**

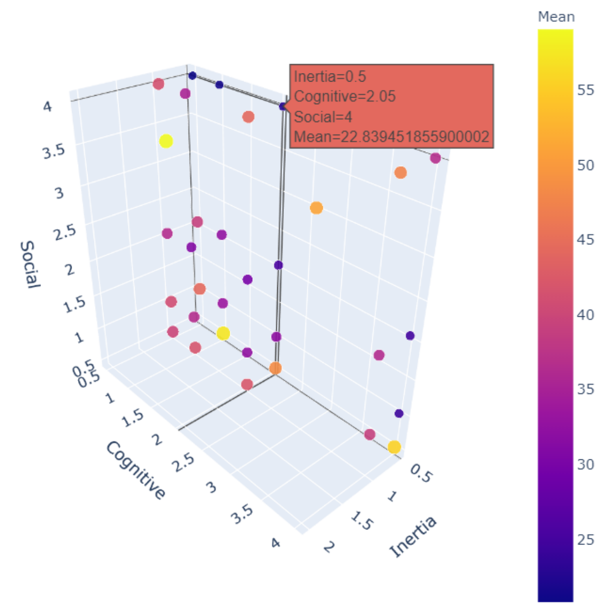


**Figure 13.** Section within a 3d interactive graphic that illustrates the effect of different configurations for inertia / cognitive/ social parameters over the PSO results for the Rastrigin function: **one of the best configurations found**. **There were selected only the configurations that lead to an average minimum value smaller than 60.**

# 4 A comparison between PSO, Hill Climbing and Genetic algorithms

## 4.1 Similarities

The similarities between those three methods described in the previous sections are mainly the characteristics of evolutionary calculus: **all three maintain a population of representations of specific candidate solutions which evolves along iterations applying a series of operators depending on the information given by a fitness function that measures individual reward**.

Another aspect which is worth to be taken into account when talking about the performance of all described algorithms, is **the necessity of a hyperparameter optimization**, in the case of the algorithms which do have a genetic algorithm in their implementation and as well, in the case of the PSO. **This hyperparameters search can consist an expensive step, computationally talking, if the function belongs to a bigger n-dimensional space**, because for those algorithms to return statistically significant results (i.e. results resulted during an experiment with a minimum number of runs equal to 30) there are required a couple of hours for them to finish their execution.

## 4.2 Differences

Even though, **the article do not include an execution time analysis, we noticed that the smallest execution time is provided by the PSO, since it manages to obtain best results when time constraints are added**. PSO is followed by the general schema of a genetic algorithm regarding performance and execution time analysis, and lastly by the Hill Climbing which due to his trajectory nature involves a longer execution time. Even though using a Hill Climber in the implementation of a numerical optimization software implies an increased computational execution time, the Hill Climber is the one which can improve significantly the results when used in combinations of more than one method, as the hybridizations presented.

Another important difference is that **PSO is suitable in problems in which the input space is a region of an N-dimensional space**. This can be sometimes **very restrictive**, as the **genetic algorithms can work with other representations as well, such as permutations or neural network topology** (both mutation and crossover can be adapted for this case).

## 4.3 Results comparative analysis

The results presented above do not allow for a conclusive argument that one method is definitely better than another. For example, **the PSO algorithm performed well on Griewangk, SixHump and Rosenbrock functions, but it was overcomed by the GA + HC hybridization regarding the results on Rastrigin function**. Even though the PSO managed to obtain a minimum value in the experimental results of 11.93956 with a cognitive weight of 1 for Rastrigin, comparing to the hybrid version with a best minimum of 15.9226, if we take into consideration as well the maximum and the average results, it is obvious that the hybridization performed better.

Although it failed on reaching the global minima, the **Hill Climbing algorithm performed significantly better on the Rosenbrock function** than the rest of the algorithms presented, by having the smallest average result of 8.61985. However, **it returned the worst average result on the Rastrigin function in a 30-dimensional space of** 63.87924, compared to the rest of the results whose maximum average in the experiments implied was close to 60, in the case of using the general schema of a genetic algorithm.

It can be clearly noticed that both hybrid versions of the Hill Climbing and Genetic Algorithm brought improved results, compared to the performance obtained by using those two algorithms individually. On this line, **using combinations of the described methods can consist one of the alternatives for improving the results of a stand-alone algorithm**. For instance, even though we did not pursued experiments on a combination that has on a first layer a genetic algorithm and then a PSO, it can be an alternative worth taking into account. *An another idea since the GA + HC hybridization performed best, would be to add in this hybrid version a middle layer that will consist of a PSO implementation*. It is required to keep the Hill Climber as a last layer since it was noticed that only then, Hill Climbing improved significantly the results.

## 5 Conclusions

In this paper we studied three different methods for function optimization and we presented the values obtained for some problems often used for performance testing. **Most of the functions are complex to minimize** because they have **many local minima** but **very few** (most of them only one) **global minima**.

From the results emphasized, it is clear that **SixHump was the easiest to optimize, as both GA and PSO found the global minima with very high accuracy**. Additionally, this is the function with the smallest cardinal of the local minima set and has even two global minima instead of one.

On the other hand, **the hardest function to optimize was definitely Rastrigin, as the best average value for the minimization found was** $25$ **given by the GA + HC hybridization**. This is relatively far from the global minimum which is 0 for $x = 0$.

One important aspect to consider when choosing a model is **the execution time**. **Even though a genetic algorithm might perform better on some problem, if it takes too long to fund the results it may not be feasible**. In this case, **using PSO, which may give a slightly worse results, but converge much faster on our experiments, might be a necessary trade-off**. Other alternatives for improving the processing time is to use hybrid algorithm, such as performing a few iterations of a GA and then start PSO from the resulting population.

The comparative analysis is based solely on empirical results and can not give a conclusive argument on the performance. From the previous section we can tell that **no method outperforms another on all functions, so choosing an algorithm to solve a problem must be carefully chosen, possibly by trial and error**. Also, *a more thorough parameter search, using better methods than an exhaustive grid search, must be used*.

## 6 Future work

We think that **a more thorough hyperparameter optimization would improve the results for the Rastrigin and Rosenbrock function**. Although the grid search method we used does not scale well, other methods, such as using a different genetic algorithm, for this task are available.

## References

[1] **The definitions for the given functions to be minimized**: http://www.geatbx.com/docu/

fcnindex-01.html

[2] **De Jong function’s graphic**:
https://www.al-roomi.org/component/tags/tag/1st-de-jong-s-function

[3] **Gray Code representation**: https://en.wikipedia.org/wiki/Gray_code

[4] **Rastrigin’s function graphic**: https://en.wikipedia.org/wiki/Rastrigin_function

[5] **Hill Climbing pseudocode**: https://profs.info.uaic.ro/~pmihaela/GA/laborator2.html

[6] **Genetic algorithms description**: https://profs.info.uaic.ro/~pmihaela/GA/laborator3.html

[7] **Particle Swarm Optimization - Introduction**: https://profs.info.uaic.ro/~pmihaela/MOC/PSO.html

[8] **Swarm intelligence**: https://en.wikipedia.org/wiki/Swarm_intelligence

[9] **Particle Swarm Optimization - Implementation details**: https://en.wikipedia.org/wiki/Particle_swarm_optimization