Bacterial Foraging Optimization Algorithm (BFOA)

Technical Report · June 2015

DOI: 10.13140/RG.2.2.10053.24803

1 author:

Miguel Angel Tovar Estrada
Autonomous University of Nuevo León

Bacterial Foraging Optimization Algorithm (BFOA)

Miguel Angel Tovar Estrada
Universidad Autónoma de Nuevo León
Facultad de Ingenierá Mecánica y Eléctrica,
Maestría en Ciencias de la Ingeniería Eléctrica,
email: miguel.tovare@gmail.com

1 Introduction

The goal of this work is the implementation of a BFOA. As is known was proposed by Passino [1], which belong to the family of nature-inspired optimization algorithms. A detailed summary of social foraging and the BFOA is provided in the book by Passino [2]. Passino provides a follow up review of the background models of chemotaxis as optimization and describes the equations of the Bacterial Foraging.

To understanding this Algorithm, we used 3 algorithms that were: a Genetic Algorithm, EvoNorm and a Particle Swarm Optimization. The results were compared in the section 5 of this document, and have a description in each of the Function of evaluation.

2 Description of the Algorithms

As was mentioned were used, to this work, 4 Algorithms including the BFOA, following is a brief description:

2.1 Genetic Algorithm

The Genetic Algorithm is an Adaptive Strategy and a Global Optimization technique. It is an Evolutionary Algorithm and belongs to the broader study of Evolutionary Computation. The GA is inspired by population genetics (including heredity and gene frequencies), and evolution at the population level, as well as the Mendelian understanding of the structure (such as chromosomes, genes, alleles) and mechanisms (such as recombination and mutation). [3]

2.2 Evonorm (Evolutionary Algorithm of Random Variables with Normal Distributions)

Evonorm is an evolutionary algorithm where the population is built by random variables with normal distribution. The parameters of these random variables are determined by the calculation of the mean and the standard deviation of selected population of solutions. The evolutionary algorithm replaces the crossover and the mutation procedure with new procedures to calculate parameters of random variables and to generate new individuals from these random variables with normal distribution. [4]

2.3 Particle Swarm Optimization

Particle Swarm Optimization belongs to the field of Swarm Intelligence and Collective Intelligence and is a sub-field of Computational Intelligence. PSO is inspired by the social foraging behavior of some animals such as flocking behavior of birds and the schooling behavior of fish. Particles in the swarm fly through an environment following the fitter members of the swarm and generally biasing their movement toward historically good areas of their environment. [5]

2.4 Bacterial Foraging Optimization Algorithm

The Bacterial Foraging Optimization Algorithm belongs to the field of Bacteria Optimization Algorithms and Swarm Optimization, and more broadly to the fields of Computational Intelligence and Metaheuristics. It is related to other Bacteria Optimization Algorithms such as the Bacteria Chemotaxis Algorithm [6], and other Swarm Intelligence algorithms such as Ant Colony Optimization and Particle Swarm Optimization. The Bacterial Foraging Optimization Algorithm is inspired by the group foraging behavior of bacteria such as E.coli and M.xanthus. Specifically, the BFOA is inspired by the chemotaxis behavior of bacteria that will perceive chemical gradients in the environment (such as nutrients) and move toward or away from specific signals. [7]

3 BFOA fundamentals knowledge

3.1 How the Bacterial are in the real live E. Coli

[8] During foraging of the real bacteria, locomotion is achieved by a set of tensile flagella. Flagella help an E.coli bacterium to tumble or swim, which are two basic operations performed by a bacterium at the time of foraging. When they rotate the flagella in the clockwise direction, each flagellum pulls on the cell. That results in the moving of flagella independently and finally the bacterium tumbles with lesser number of tumbling whereas in a harmful place it tumbles frequently to find a nutrient gradient. Moving the flagella in the counterclockwise direction helps the bacterium to swim at a very fast rate. In the above-mentioned algorithm the bacteria undergoes chemotaxis, where they like to move towards a nutrient gradient and avoid noxious environment. Generally the bacteria move for a longer distance in a friendly environment. Figure 1 depicts how clockwise and counter clockwise movement of a bacterium take place in a nutrient solution.

When they get food in sufficient, they are increased in length and in presence of suitable temperature they break in the middle to from an exact replica of itself. This phenomenon inspired Passino to introduce an event of reproduction in BFOA. Due to the occurrence of sudden environmental changes or attack, the chemotactic progress may be destroyed and a group of bacteria may move to some other places or some other may be introduced in the swarm of concern. This constitutes the event of elimination-dispersal in the real bacterial population, where all the bacteria in a region are killed or a group is dispersed into a new part of the environment.

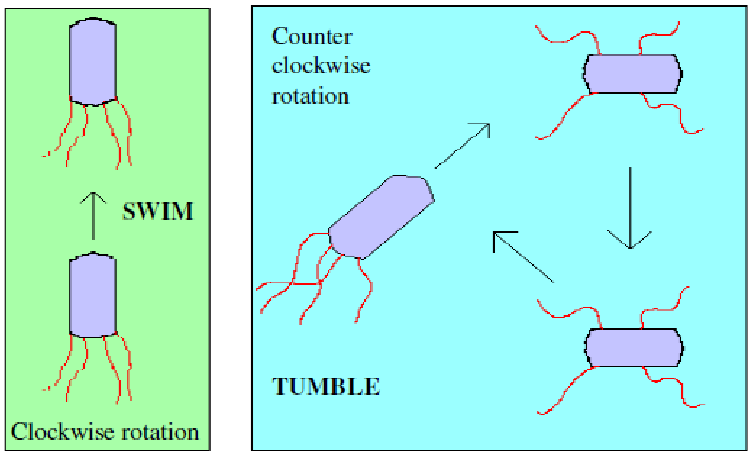

Figure 1: *Swim and tumble of a bacterium* [8]

3.2 Chemotaxis

[9] The random moving patterns that the bacteria generate in the presence of chemical attractants and repellants are called chemotaxis. For E.Coli, this process was simulated by two different moving modes tumble and run or move, and bacterium alternates between these modes until divided into two. In tumble bacterium randomly searches a direction of moving and in run it moves a number of small fixed length steps $(C(i))$ in the selected tumble direction $(\phi(i))$ or better nutrients collection. Mathematically this can be expressed as

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + (C(i))(\phi(i)), \tag{1}$$

$$\forall i = 1, 2, ..., S \tag{2}$$

where S is size of the colony j, k and l are respectively the Chemotaxis, reproduction and elimination-dispersal step indices respectively and

$$\theta^i(j+1,k,l) or \theta^i(j,k,l) \in D_1 \times D_2 \times \cdots \times D_P \tag{3}$$

(in short is represented θ^i)
be position or solution vector for i^{th} bacterium, ($D_i s$ *are domain of* $\mathbb{R}^N$ *Search Space*).

If $J(\theta^i(j,k,l))$ be the cost or fitness function then bacterium uses run if $J(\theta^i(j+1,k,l))$ is better than $J(\theta^i(j,k,l))$ otherwise it enters into the next tumbling step.

For tumble

$$\phi^i(m) = \frac{\Delta(m)}{\sqrt{\Delta^T(m)\Delta(m)}}, \forall m = 1, 2, ..., N \tag{4}$$

$\Delta(m)$ is a random number in $[0, 1]$

$\phi(i) = \phi^i(m)_{m=1}^N$

3.3 Reproduction

[9] As bacteria are not immortal and like to grow population for better social structure they uses rule of evolution and when appropriate conditions appear then individual will reproduce themselves after certain number of chemotaxis steps. For this purpose health of the bacteria, which is sum of fitness in each chemotaxis including initialization step is considered $\sum_{j=1}^{N_c} J(\theta^i(j,k,l))$ and to keep constant population bacterium with better health capture the position of bacterium with poor heath, for this purpose individual reproduce one of its identical clone.

3.4 Elimination-Dispersal

[9] In the evolutionary process, elimination and dispersal events occur such that bacteria in a region are eliminated or a group is dispersed from current location and may reappear in the other regions due to environmental changes or some natural calamities. They have the effect of possibly destroying chemotactic progresses, but they also have the effect of assisting the chemotaxis, since dispersal may place bacteria near good food sources. From evolutionary point of view, elimination and dispersal was used to guarantee diversity of the individuals and to strengthen the ability of global optimization. In BFOA, bacteria are eliminated with a probability P_{ed} , and to keep population size constant, if a bacterium is eliminated, simply disperse one new bacterium to a random location of the search space.

3.5 Swarming

[9]

Prof. Passino had experimented for an interesting group behavior of the E.Coli bacteria and then he was successful to explain this swarming behavior using the following mathematical model. He observed that when a group of E.Coli bacteria is placed in the center of a semisolid agar with a single nutrient chemo-effecter, they move out from the center in a traveling ring of cells by moving up the nutrient gradient created by consumption of the nutrient by the group, and this cell-to-cell signaling attractant and a replant based network group can be modeled:

$$J_{cc}^i(\theta_{ibest}^i(j,k,l), \theta_{best}) = Penalty(i) \tag{5}$$

$$= \sum_{i=1}^{S}\left[-d_{attract}exp^{\left(-w_{attract}\sum_{j=1}^{N}\left(\theta_{best}^{'j}-\theta_{best}^{ij}\right)^2\right)}\right] + \sum_{i=1}^{S}\left[-h_{repellant}exp^{\left(-w_{repellant}\sum_{j=1}^{N}\left(\theta_{best}^{'j}-\theta_{best}^{ij}\right)^2\right)}\right] \tag{6}$$

Where $J_{cc}^i(\theta_{ibest}^i, \theta_{best)}$ is the cell-to-cell cost or penalty ($Penalty(i)$) for i^{th} bacteria in the colony,$\theta_{best}^{'j}$ is the j^{th} component of the current N-dimensional global best solution vector $\theta_{best}^{'}$ and θ_{best}^{ij} is the j^{th} component of N-dimensional best solution vector attended by the i^{th} bacteriain the colony. Other parameters are attractant and repellant dependent constant, $d_{attract}$ is the depth of the attractant released by the bacterial cell and $w_{attract}$ is the measure of the width of the attract signal. Generally $h_{repellant}$ is the height of the repellant effect and $w_{repellant}$ is a measure of the width of the repellant.

4 BFOA

The algorithm presented in this work has certain changes in its structure and added something new based on the original. Passino present the first BFOA as such in 2002 [1], since then there have been several improvements, one of which is in the article submitted by Mahapatra and Banerjee in 2013 [9], in which starting from the presented by Passino requires fewer evaluations in simple detail to eliminate an evaluation in the process, although this improvement was not much help, because many evaluations and much more computation time required.

Adding the described by Passino [2] *For actual bacterial populations, S can be very large (e.g., S = 109), but p=3. In our computer simulations, we will use much smaller population sizes and will keep the population size fixed. We will allow p > 3, so we can apply the method to higher dimensional optimization problems.*

Although is possible used it as is for this work that need dimensions up to 50, but, much computation time is needed, and that is by the value of C(i) (that in some articles propose the value of 0.1 or 0.05) that is somehow exploration, for this reasons, had to do the following:

C(i) initiates with a value of 15 % of the range of the function that there is evaluating who we will call ”gamma”. And there is used a variable that we will call ”mt”, which after there ends the loop of all the bacteria, C(i) will be updated like C(i) =C (i) - mt, up to a minimal value ”lambda”, which is 0. 075% of the range of the function that it is evaluating, so this way the exploration goes away to reducing little by little, so that it is a more suitable form and in less evaluations to come to a better result.

Something that was also modified in the structure, is the use of the best population having individual by individual, i.e. the Pibest.At the end of the loop of chemotactic and of the loop of reproduction the population following is the best bacterium in every position. (see Annex)

Following pseudo code structure is representing the step-bystep details of the simulation of BFOA used in our study.

4.1 Description of the parameter

Let us define a chemotactic step to be a tumble followed by a tumble or a tumble followed by a run.
Let j be the index for the chemotactic step.
Let k be the index for the reproduction step.
Let l be theindex of the elimination-dispersal event. Also let:
N: Dimension of the search space,
S: Total number of bacteria in the population,
Nc : The number of chemotactic steps,
Ns: The swimming length.
Nre : The number of reproduction steps,
Ned : The number of elimination-dispersal events,
Ped : Elimination-dispersal probability,
NG : The number of generations

C (i): The size of the step taken in the random direction specified by the tumble.
MIN: The minimum number of the range of the function to evaluate
MAX: The maximum number of the range of the function to evaluate

4.2 Pseudecode

STEP 1: *Initialization*

1.1 *Initialization of parameters* :

N, S, Nc, Ns, Nre, Ned, Ped, NG, Sr,
$alpha = 2 * Nre * Nc * NG$
$gama = (abs(MAX) + abs(MIN)) * 0.15$
$lambda = (abs(MAX) + abs(MIN)) * 0.00075$
$mt = (gama - lambda)/alpha$
$c = ones(S, Nre) * gama$

```
1.2 for i =1 to S do
begin
    Randomly generate an N-dimensional position vector $\theta^i \in D_1 \times D_2 \times \cdots \times D_N$. Calculate fitness value: $J^i = J(\theta^i)$, $\theta^i_{best} = \theta^i$
end
```

1.3 Find the initial global best solution vector $\theta_{best} = \theta^i$ and fitness $J_{best} = J^i$ from current bacteria colony.

STEP 6:
6.1 *Arrange bacteria in descending order of health*
Sort(θ^i_{best} is the colony)
6.2 *Reproduce*
Replace all S/2 number of relatively unhealthy bacteria appearing at the end half of the sorted list by the copy of bacteria with better health appearing towards the beginning of the list.
6.3
vector θ^i = vector θ^i_{best}

STEP 7:
7.1
For each bacterium $i = 1$ *to* S in the colony with probability Ped , eliminate and disperse. For this purpose generate a random probability and check whether it is greater or equal to the given elimination probability Ped or not, if successful then eliminate the bacterium and place it at a random position in the search space $\theta^i \in D_1 \times D_2 \times \cdots \times D_N$, calculate fitness value $J^i = J(\theta^i)$ and then initialize the i^{th} bacterium with $J^i_{best} = J^i$, θ^i_{best}=θ^i.
7.2
vector θ^i = vector θ^i_{best}

```
for $l = 1$ to $Ned$ do /* STEP 2 */
    for $k = 1$ to $Nre$ do /* STEP 3 */
        for $j = 1$ to $Nc$ do /* STEP 4 */
            for $i = 1$ to $S$ do /* STEP 5 */
                STEP 5.1 Tumble: Generate a random number $\Delta(m)$ in $[0, 1]$
                $\phi^i(m) = \frac{\Delta(m)}{\sqrt{\Delta^T(m)\Delta(m)}}, \forall m = 1, 2, ..., N$
                $\phi(i) = \phi^i(m)_{m=1}^N$
                STEP 5.2 Move:
                $\theta^i(j+1) = \theta^i(j) + (C(i))(\phi(i))$
                STEP 5.3 Compute fitness
                $J^i = J(\theta^i(j+1))$
                if $J^i < J_{best}$ then
                    $J_{best} = J^i$
                    $\theta_{best} = \theta^i(j+1)$
                end
                if $J^i < J^i_{best}$ then
                    $J^i_{best} = J^i$
                    $\theta^i_{best} = \theta^i(j+1)$
                end
                STEP 5.4 Swim:
                $m = 0$
                while $m < Ns$ do
                    $m$=$m$+1
                    if $J^i < J^i_{best}$ then
                        $\theta^i(j+1) = \theta^i(j+1) + (C(i))(\phi(i))$
                        $J^i = J(\theta^i(j+1))$
                        if $J^i < J_{best}$ then
                            $J_{best} = J^i$
                            $\theta_{best} = \theta^i(j+1)$
                        end
                        if $J^i < J^i_{best}$ then
                            $J^i_{best} = J^i$
                            $\theta^i_{best} = \theta^i(j+1)$
                        end
                    else
                        $m = Ns$
                    end
                end
            end
            if $c > lambda$ then
                $c = c - mt$
            end
        end
        STEP 6
    end
    STEP 7
end
```
Algorithm 1: The BFOA used in the simulations, own creation

5 Experimental Setup

The performance of this BFOA has been evaluated on a test bed of 10 benchmark functions. In order to get a better understanding of the algorithm, the results are compared to those from a GA, EvoNorm, and PSO . Details of the mathematical form of the benchmark functions the are shown in table 1.

For each Algorithm was used the next parameters: The size of the population was 100 individuals in the three case of dimension problem, where in case 1: n=10 dimensions, case 2: n=25 dimensions and case 3: n=50 dimensions.

In the case of the BFOA the number of fitness function evaluations (FEs) was different, because this algorithm need it to get a acceptable solution,using different maximum number of FEs depending on the complexity of the problem.. The other 3 algorithms used just 10,000 FEs.

30 independent runs of the four competitor algorithms were carried out on each problem and the average of the best -of- run solutions and standard deviations were noted.

The Software used was Scilab.

The characteristics of the machine used:
Model: Idepad S400u, Lenovo
Processor: Intel(R) Core(TM) i5-3337U CPU@ 1.80GHz
RAM: 4.00 GB
OS: Windows 8.1, 64 bits.

5.1 Functions using in the work

Benchmark Function Tests

Ten benchmark functions are chosen in order to test the performance of the BFOA. The benchmark functions are as follows.

f1: Sphere's function (also known as De Jong's function 1) is the simplest test function, which is continuous, convex and unimodal.

f2: Schwefel's function produces rotated hyper-ellipsoids with respect to the coordinate axes. It is continuous, convex and uni-modal. This function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction.

f3: 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 multi-modal. However, the location of the minima are regularly distributed.

f4: Rosenbrock's function (also known as De Jong's function 2, or Banana function) is a classic optimization problem. The global optimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial, however convergence to the global optimum is more difficult.

f5: Schwefel12

f6: Griewangk's function is similar to Rastrigin's function. It has many widespread local minima. However, the location of the minima are regularly distributed.

f7: Ackley's function is a widely-used multi-modal test function.

f8: Langermann's function is a multimodal test function. The local minima are unevenly distributed.

f9: Michalewicz's function is a multimodal test function (owns n! local optima). The parameter m defines the "steepness" of the valleys or edges. Larger m leads to more difficult search. For very large m the function behaves like a needle in the haystack (the function values for points in the space outside the narrow peaks give very little information on the location of the global optimum).

f10: Deceptive's function is a class of problems in which the total size of the basins for local optimum solutions is much larger than the basin size of the global optimum solution. Clearly, this is a multimodal function.

Table 1: La tabla muestra el modelo:...

f	Function	Mathematical representation	Range	$f(x_i*)$	x_i*
f_1	Sphere	$f(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$	$-100 \leq x_i \leq 100$	0	0
f_2	Schwefel	$f(x) = \sum_{i=1}^{n} \left[-x_i \sin(\sqrt{\lvert x_i \rvert}) \right]$	$-500 \leq x_i \leq 500$	420.9, .., 420.9	$-414.98 * k$
f_3	Rastrigins	$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$	$-5.12 \leq x_i \leq 5.12$	0	0
f_4	Rosenbrock	$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$	$-10 \leq x_i \leq 10$	$1, 1, 1, ..., 1$	0
f_5	Schwefel12	$f(x) = \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{i} x_j \right)^2 \right]$	$-100 \leq x_i \leq 100$	0	0
f_6	Griewangk	$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$-600 \leq x_i \leq 600$	0	0
f_7	Ackley	$f(x) = 20 + e - 20exp\left(-0.2\sqrt{\frac{1}{p}\sum_{i=1}^{p} x_i{}^2}\right) - exp\left(\frac{1}{p}\sum_{i=1}^{p}\cos(2\pi x_i)\right)$	$-32.7 \leq x_i \leq 32.7$	0	0
f_8	Langermann	$f(x) = \sum_{i=1}^{m} c_i \exp[-\frac{1}{\pi}\sum_{j=1}^{n}(x_j - a_{ij})^2] \cos[\pi \sum_{j=1}^{n}(x_j - a_{ij})^2]$	$-1 \leq x_i \leq 10$	$-$	$-$
f_9	Michalewicz	$f(x) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin(\frac{i x_i^2}{\pi}) \right]^{2m}$	$0 \leq x_i \leq \pi$	$n = 10$	-9.66
f_{10}	Deceptive	$f(x) = -\left[\frac{1}{n} \sum_{i=1}^{n} g_i(x_i) \right]^{\beta}$	$-5.12 \leq x_i \leq 5.12$	*To this case* n=10; n=25; n=50	- 0.158489; - 0.076146; - 0.043734

5.2 Parameters used of BFOA

BFOA The parameters used in the algorithm:
S: 100
Ns: 15
Nre: 4
Ped: 0.1
C (i): depend of a formula (see Annex)
To 10 Dimensions:
N: 10
Nc: 3
Ned: 3
NG: 3
To 25 Dimensions:
N: 25
Nc: 4
Ned: 4
NG: 6
To 50 Dimensions:
N: 50
Nc: 4
Ned: 4
NG: 10

5.3 Results

The results of the performance of the algorithms for the different functions. For each function there is a graphic for each problem dimension, and there is a table that has the results of each case.

5.3.1 Sphere's function

The Global minimum $f(x) = 0$ is obtained for $x_i = 0, i = 1, ..., n$.

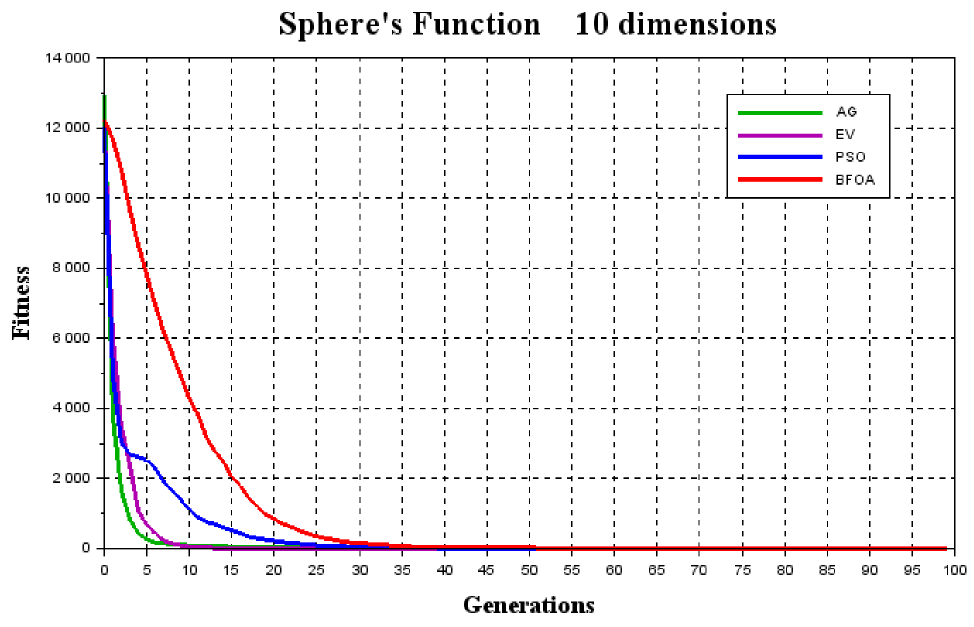

Figure 2: *Sphere's function with search space of 10 dimensions*

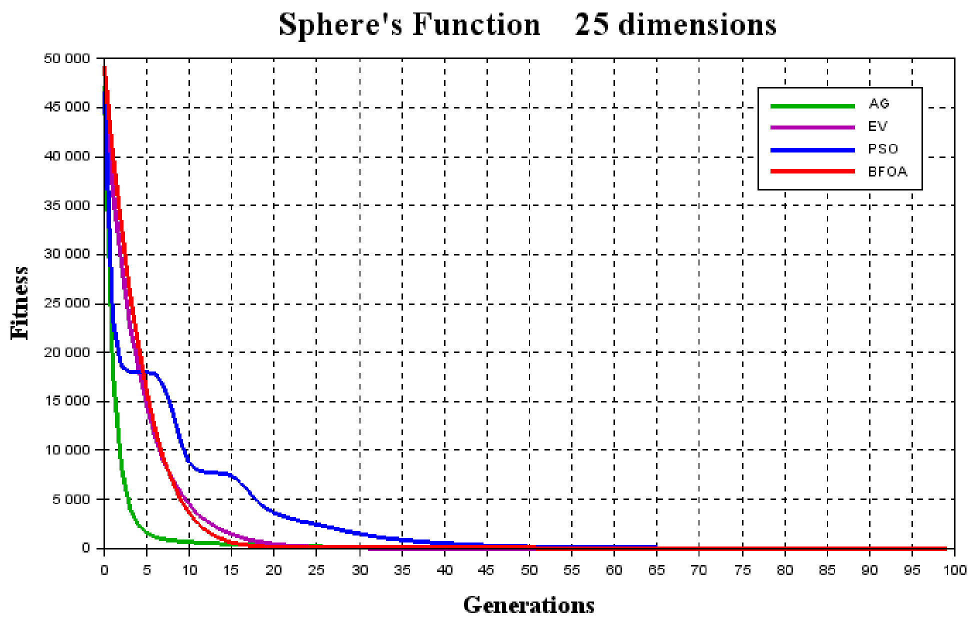

Figure 3: *Sphere's function with search space of 25 dimensions.*

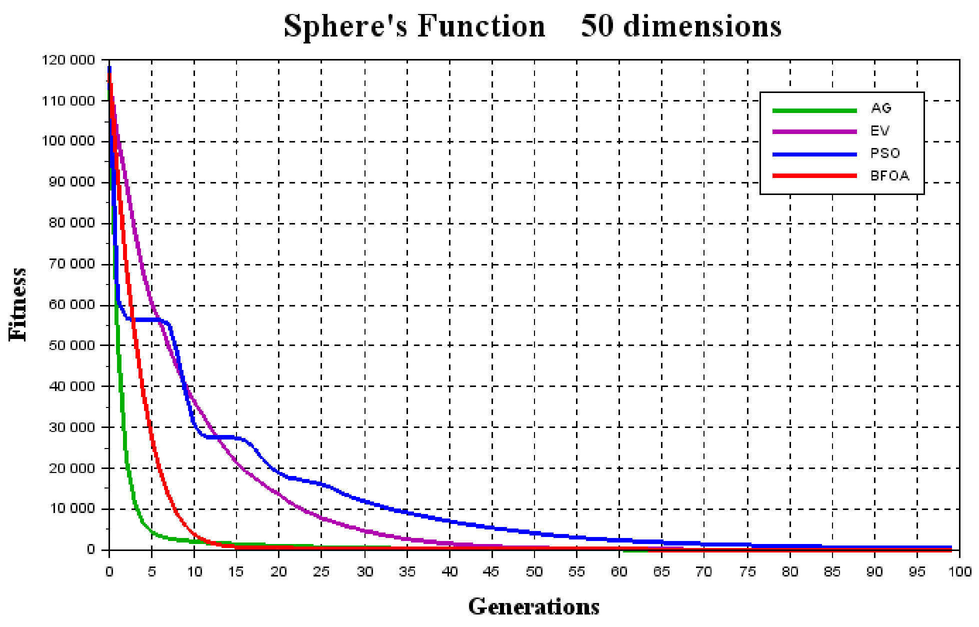

Figure 4: *Sphere's function with search space of 50 dimensions.*

Table 2: Sphere's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	0.350	0.066	1.255	**0.037**	0.049
	25	10000	3.542	1.622	**2.479**	**0.072**	0.931
	50	10000	15.321	1.869	**4.422**	**0.036**	10.439
EvoNorm	10	10000	$\mathbf{4.25}\times 10^{-13}$	$\mathbf{1.32}\times 10^{-15}$	2.199	0.054	$\mathbf{2.05}\times 10^{-13}$
	25	10000	$\mathbf{5.11}\times 10^{-6}$	$\mathbf{3.38}\times 10^{-8}$	5.120	0.152	$\mathbf{1.27}\times 10^{-6}$
	50	10000	3.166	1.090	9.701	0.056	1.243
PSO	10	10000	4.92×10^{-4}	1.03×10^{-4}	**1.220**	0.047	8.78×10^{-5}
	25	10000	1.210	0.089	2.611	0.085	0.342
	50	10000	433.540	234.260	4.772	0.037	127.940
BFOA	10	14775	0.121	0.042	1.397	0.038	0.045
	25	61341	0.073	0.007	7.056	0.270	0.043
	50	126870	**0.718**	**0.296**	19.761	0.517	**0.505**

In this table we can found that the BFOA has in the problem of 50 dimensions the best fitness, the disadvantage is the compute time. In the other case of dimensions has acceptable results, but again the disadvantage is the compute time.

5.3.2 Schwefel's function

The Global minimum $f(420.9687, ...420.9687) = -414.9829 * n$
to n=10: - 4149.829
to n=25: - 10374.572
to n=50: - 20749.145

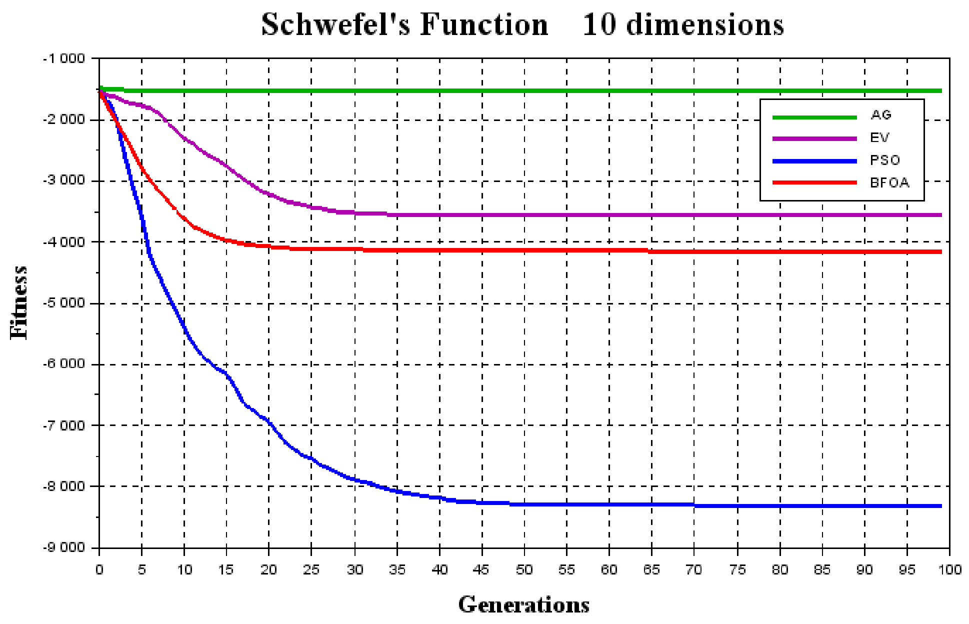

Figure 5: *Schwefel's function with search space of 10 dimensions.*

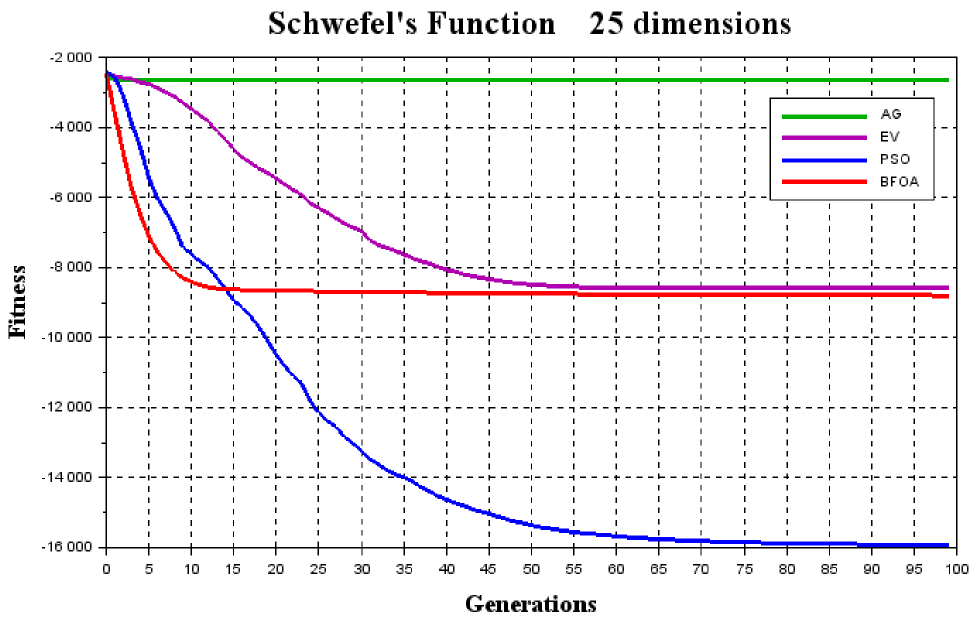

Figure 6: *Schwefel's function with search space of 25 dimensions.*

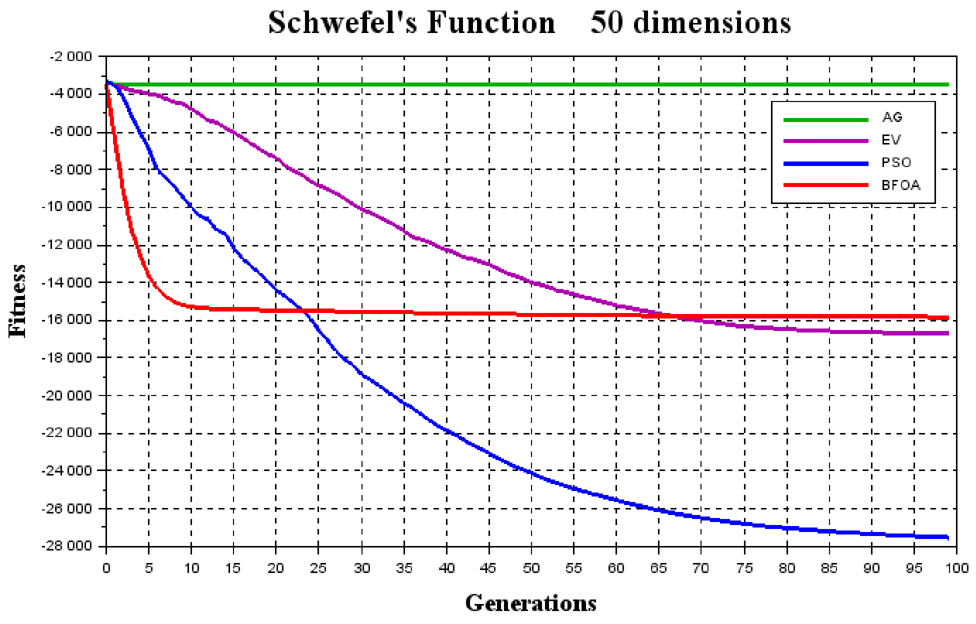

Figure 7: *Schwefel's function with search space of 50 dimensions.*

Table 3: Schwefel's function

Algorithm	**Dim**	**Mean Evaluations**	**Mean Best Fitness**	**Standard Deviation of Fitness**	**Time(s)**	**Standard Deviation of Time**	**Best Fitness**
GA	10	10000	-1514.600	**186.750**	1.363	0.008	-2109.5
	25	10000	-2639.900	136.060	**2.664**	0.046	-4049.1
	50	10000	-3463.700	1443.400	**4.850**	**0.045**	-4907.1
EvoNorm	10	10000	-3550.100	284.450	2.291	**0.007**	-4071.4
	25	10000	-8587.000	288.370	5.275	0.067	-10001.0
	50	10000	**-16690.000**	471.090	10.192	0.094	-17798.0
PSO	10	10000	-8302.100	190.720	**1.323**	0.045	**-12401.0**
	25	10000	-15921.000	9826.100	2.797	**0.039**	**-25747.0**
	50	10000	-27534.000	7833.200	5.261	0.132	**-40755.0**
BFOA	10	15798	**-4154.000**	339.370	1.638	0.091	-5512.1
	25	53132	**-8798.300**	**57.387**	7.589	0.210	-11007.0
	50	95922	-15827.000	**86.438**	21.264	0.698	-18460.0

In this case the results are close to the global minimum, the BFOA has in the problems 10 and 25 dimensions the best results, just in the problem 50 dimensions the EvoNorm has the close result

5.3.3 Rastrigins's function

The Global minimum $f(0, 0, 0, 0, ...0) = 0$

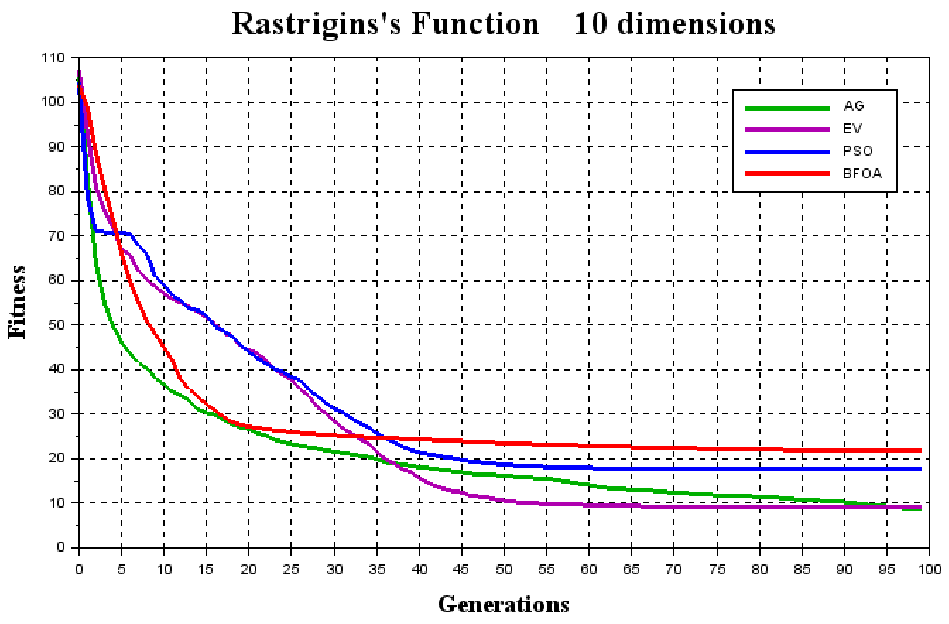

Figure 8: *Rastrigins's function with search space of 10 dimensions.*

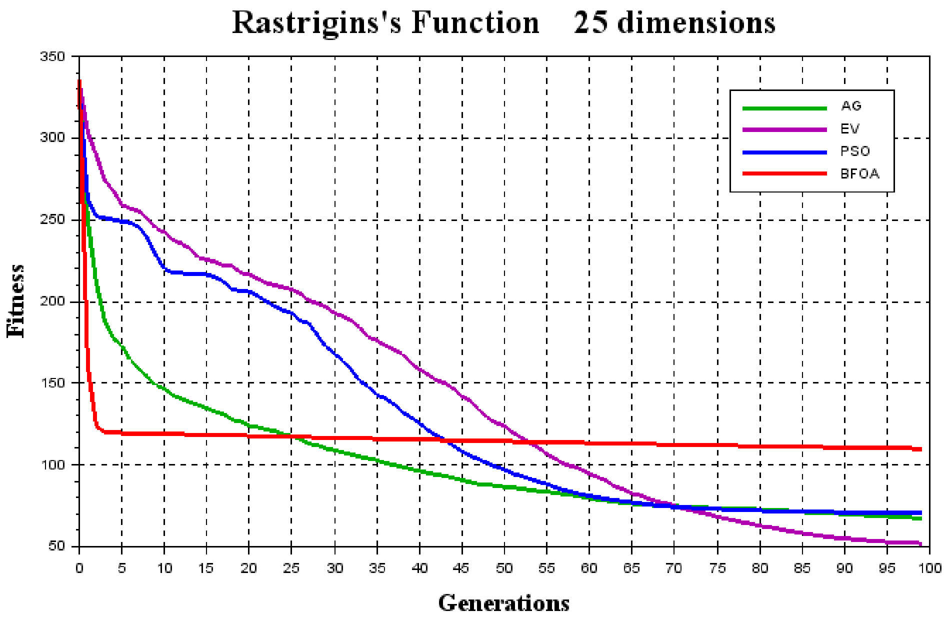

Figure 9: *Rastrigins's function with search space of 25 dimensions.*

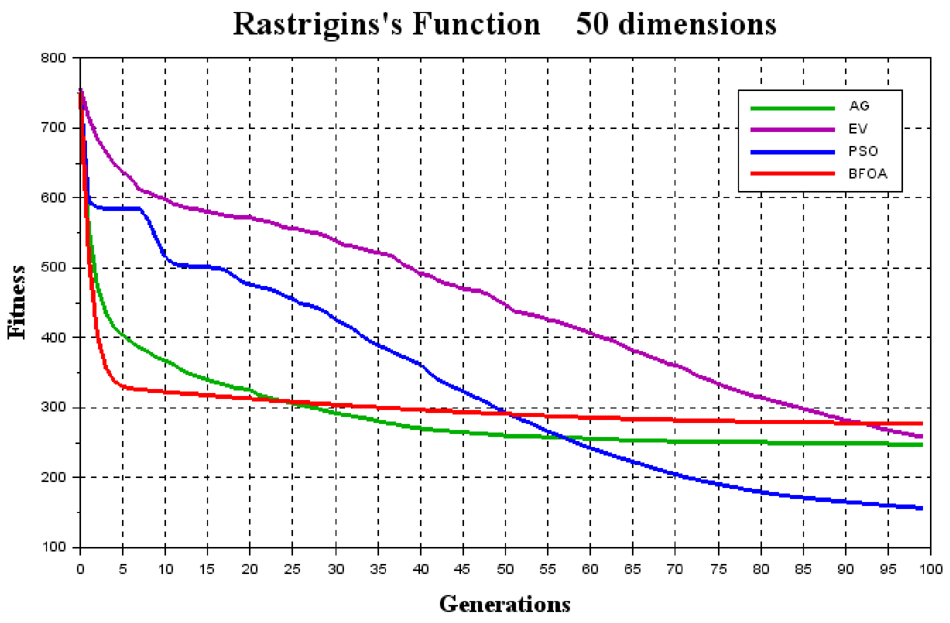

Figure 10: *Rastrigins's function with search space of 50 dimensions.*

Table 4: Rastrigins's function

Algorithm	**Dim**	**Mean Evaluations**	**Mean Best Fitness**	**Standard Deviation of Fitness**	**Time(s)**	**Standard Deviation of Time**	**Best Fitness**
GA	10	10000	**8.861**	2.908	1.594	0.019	**2.490**
	25	10000	67.405	**0.521**	**3.235**	**0.044**	25.391
	50	10000	247.780	21.811	**6.004**	0.051	181.230
EvoNorm	10	10000	9.253	3.283	2.524	**0.019**	2.985
	25	10000	**51.922**	8.864	5.836	0.063	**17.808**
	50	10000	258.530	7.424	11.332	0.077	173.950
PSO	10	10000	17.677	**0.233**	**1.570**	0.066	3.980
	25	10000	70.574	18.192	3.371	0.053	39.350
	50	10000	**156.440**	21.879	6.385	**0.035**	**107.210**
BFOA	10	15106	21.699	7.169	1.978	0.066	7.064
	25	136910	109.780	1.016	24.525	0.331	66.655
	50	131640	277.410	**5.698**	42.693	1.082	193.100

Difinitely the results of the 4 algorithms aren't good.

5.3.4 Schwefel12's function

The Global minimum $f(0, 0, 0, 0, ...0) = 0$

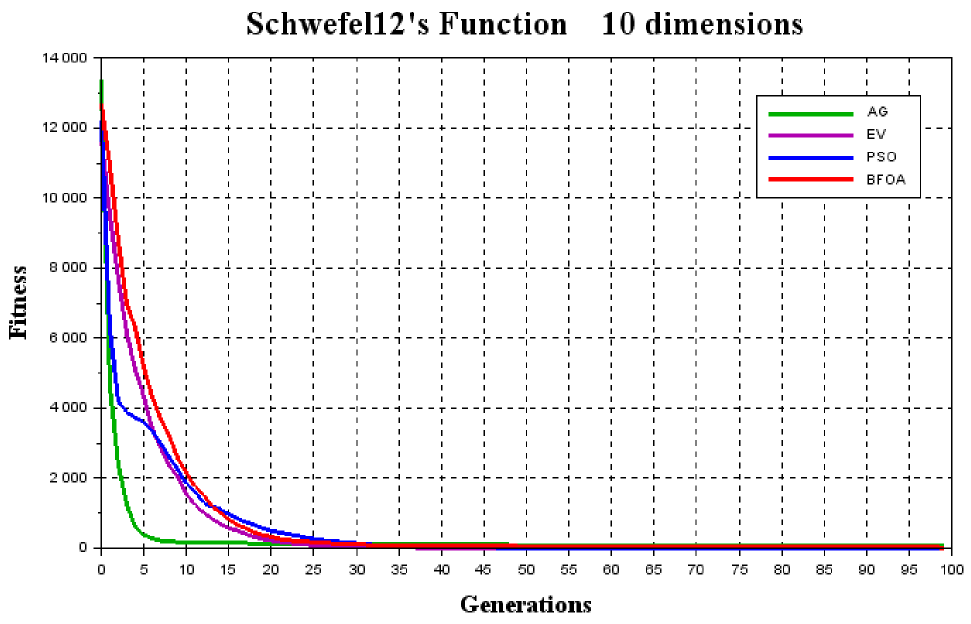

Figure 11: *Schwefel12's function with search space of 10 dimensions.*

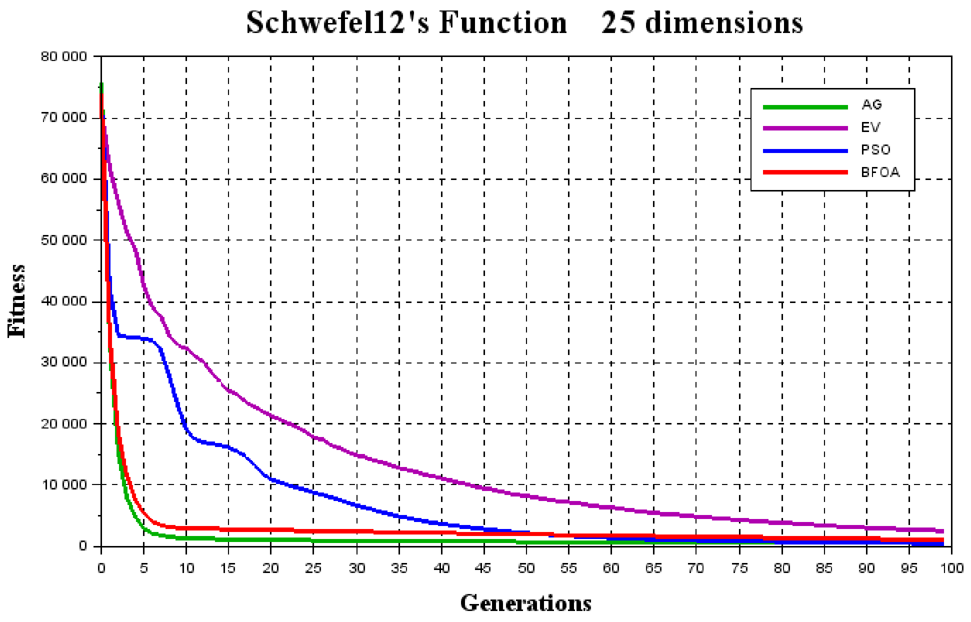

Figure 12: *Schwefel12's function with search space of 25 dimensions.*

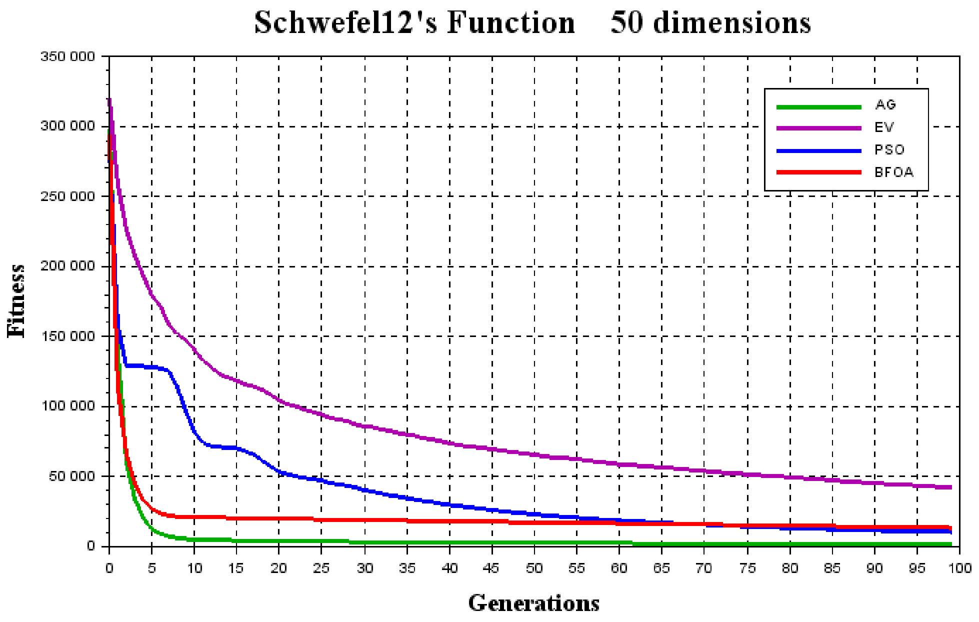

Figure 13: *Schwefel12's function with search space of 50 dimensions.*

Table 5: Schwefel12's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	62.405	51.698	1.769	0.023	7.072
	25	10000	531.200	**60.285**	**5.093**	0.142	158.230
	50	10000	**1683.700**	**1315.500**	**14.373**	0.096	**748.190**
EvoNorm	10	10000	1.770	1.760	2.705	0.027	$\mathbf{3.34\times10^{-7}}$
	25	10000	2531.000	1043.700	7.662	0.093	541.750
	50	10000	41976.000	11876.000	19.731	0.104	28013.000
PSO	10	10000	**0.004**	$\mathbf{1.50\times10^{-4}}$	**1.719**	**0.023**	2.18×10^{-4}
	25	10000	**478.370**	321.750	5.177	**0.024**	**131.710**
	50	10000	10338.000	2703.000	14.724	**0.054**	4704.800
BFOA	10	15282	9.796	7.485	2.234	0.114	0.715
	25	80448	1084.900	433.940	29.706	1.486	334.360
	50	140810	13357.000	2758.000	162.260	4.962	8456.300

In this function just the EvoNorm and PSO in the problem of 10 dimensions the results are acceptable.

5.3.5 Rosenbrock's function

The Global minimum $f(1, 1, 1, 1, ...1) = 0$

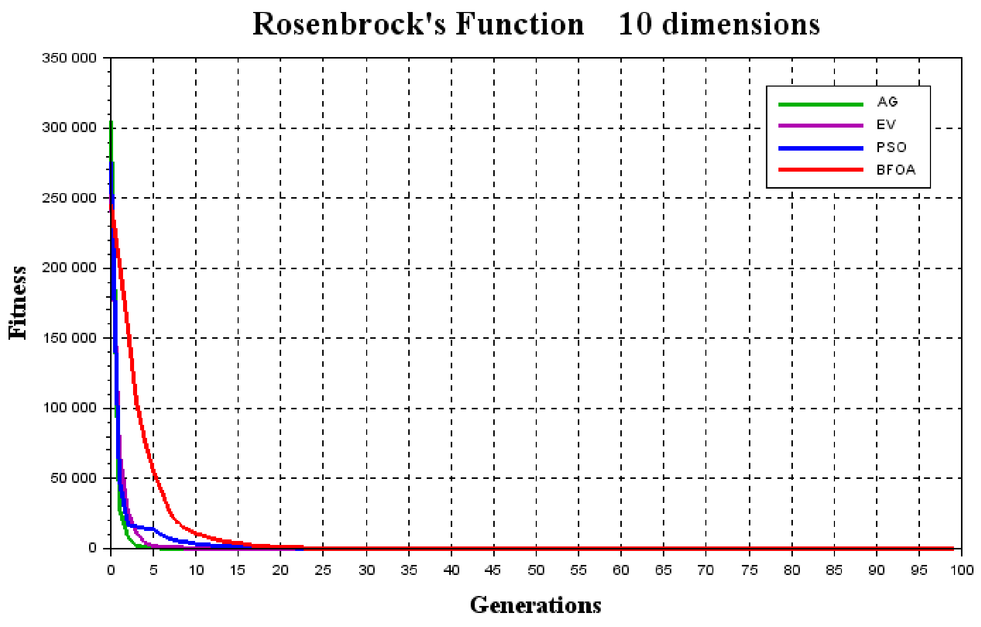

Figure 14: *Rosenbrock's function with search space of 10 dimensions.*

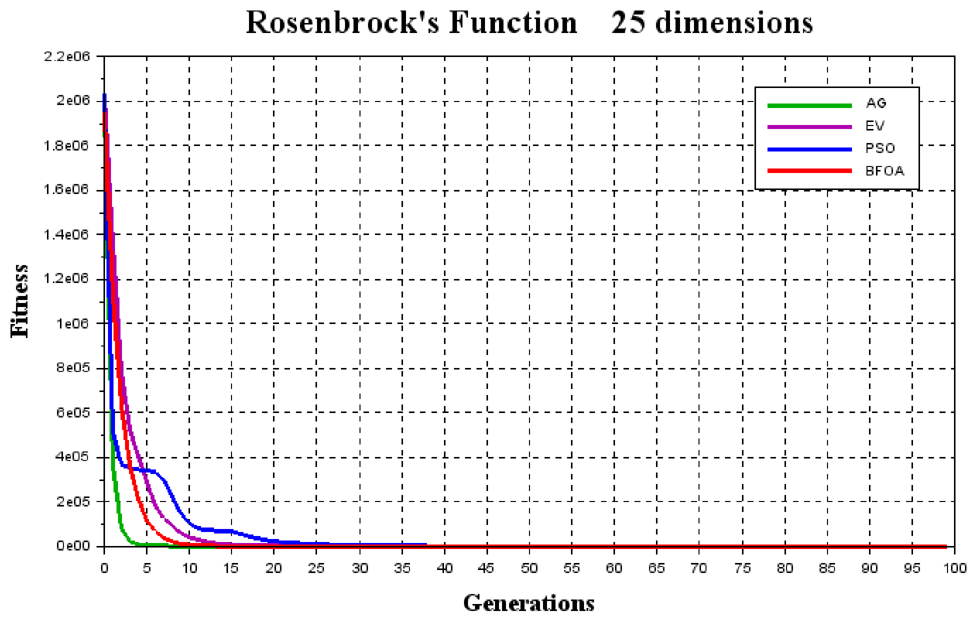

Figure 15: *Rosenbrock's function with search space of 25 dimensions.*

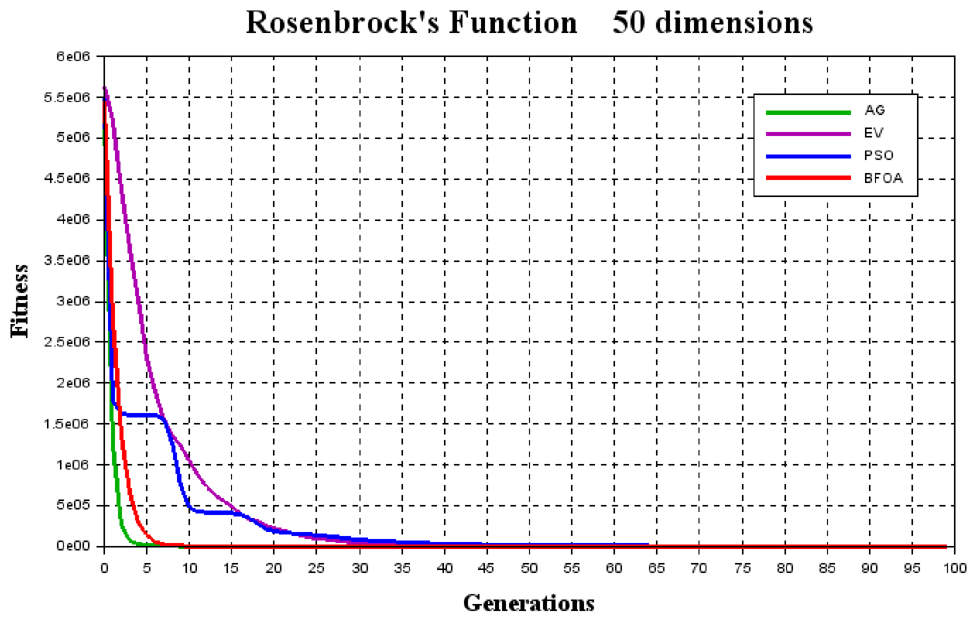

Figure 16: *Rosenbrock's function with search space of 50 dimensions.*

Table 6: Rosenbrock's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	16.901	**3.003**	1.435	0.020	7.895
	25	10000	98.035	23.843	**2.878**	0.058	69.110
	50	10000	253.100	**60.954**	**5.304**	0.041	183.960
EvoNorm	10	10000	21.429	14.172	2.366	0.046	4.570
	25	10000	**80.351**	58.083	5.477	0.126	**17.741**
	50	10000	250.810	61.820	10.582	0.042	91.565
PSO	10	10000	19.126	14.786	**1.383**	**0.018**	**0.010**
	25	10000	93.047	55.334	3.001	**0.048**	23.074
	50	10000	954.110	336.290	5.682	**0.029**	376.950
BFOA	10	16810	**7.290**	5.892	1.817	0.125	1.398
	25	66460	95.632	**1.013**	10.277	0.940	20.594
	50	139990	**145.090**	205.940	33.933	3.011	**48.079**

Despite that the results of the algorithms aren't good, the BFOA has similar or better results that the other algorithms.

5.3.6 Griewangk's function

The Global minimum $f(0, 0, 0, 0, ...0) = 0$

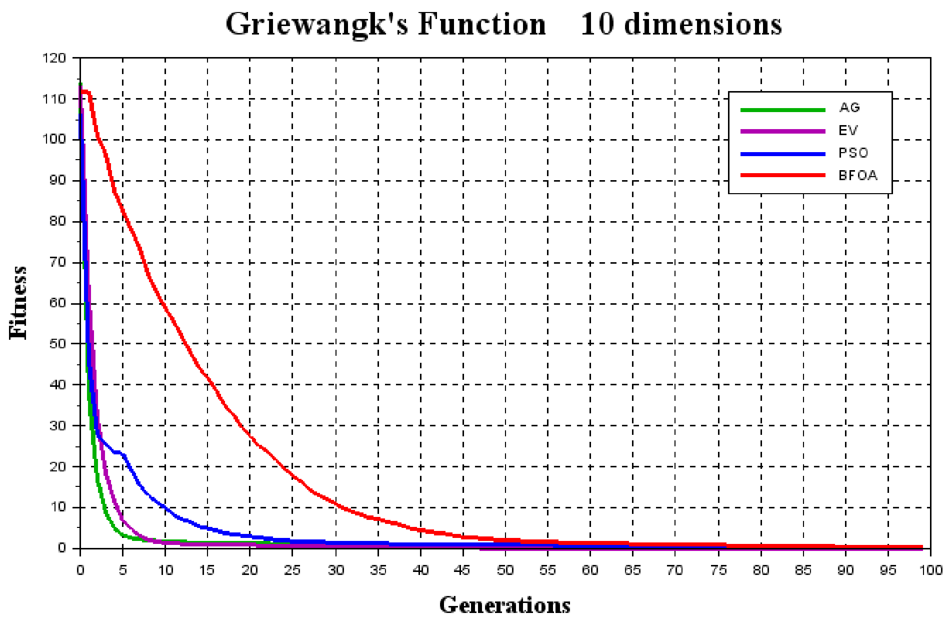

Figure 17: *Griewangk's function with search space of 10 dimensions.*

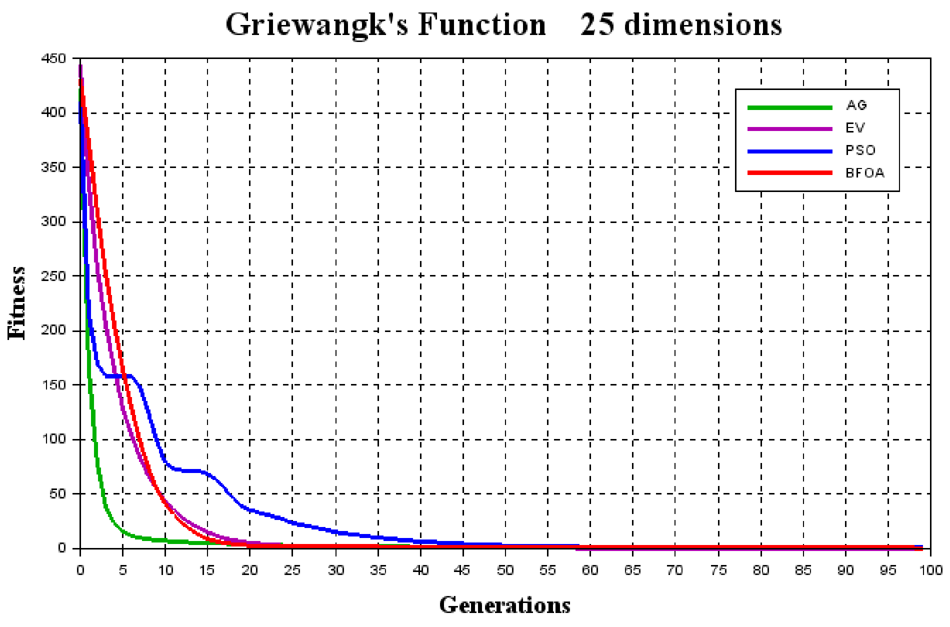

Figure 18: *Griewangk's function with search space of 25 dimensions.*

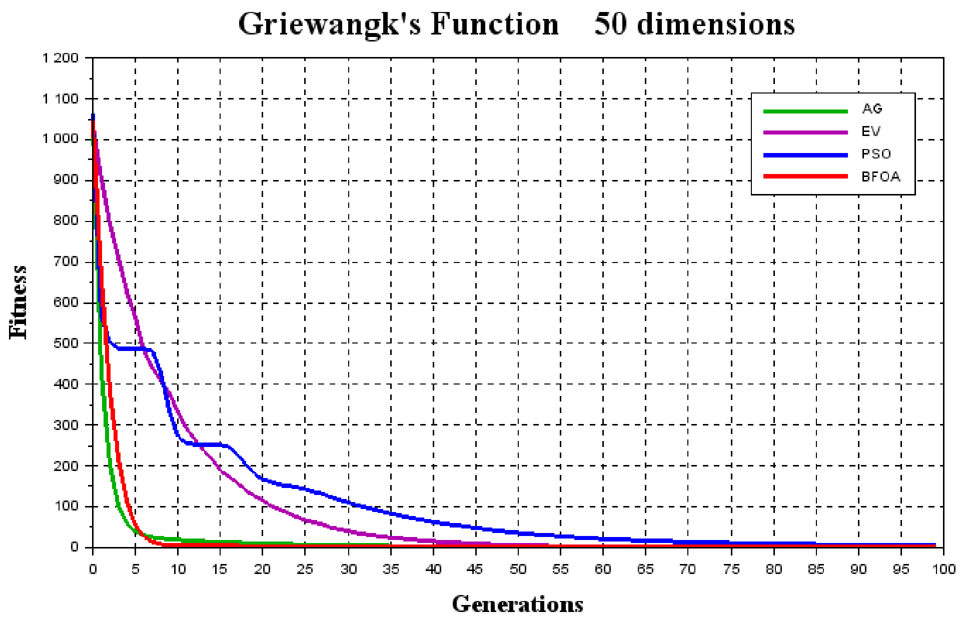

Figure 19: *Griewangk's function with search space of 50 dimensions.*

Table 7: Greiwangk's function

Algorithm	**Dim**	**Mean Evaluations**	**Mean Best Fitness**	**Standard Deviation of Fitness**	**Time(s)**	**Standard Deviation of Time**	**Best Fitness**
GA	10	10000	0.343	0.182	1.484	0.040	0.054
	25	10000	0.909	0.046	**2.879**	**0.035**	0.728
	50	10000	1.070	**0.009**	**5.286**	0.049	1.037
EvoNorm	10	10000	**0.049**	$\mathbf{5.73}\times 10^{-5}$	2.426	0.082	$\mathbf{2.00}\times 10^{-14}$
	25	10000	**0.009**	**0.009**	5.470	0.097	$\mathbf{7.30}\times 10^{-6}$
	50	10000	0.989	0.065	10.597	0.204	0.828
PSO	10	10000	0.232	0.140	**1.424**	**0.040**	0.090
	25	10000	0.755	0.255	3.012	0.068	0.445
	50	10000	4.875	4.787	5.633	**0.042**	2.214
BFOA	10	12592	0.228	0.051	1.596	0.041	0.129
	25	60424	0.173	0.035	9.655	0.228	0.116
	50	194790	**0.968**	0.065	43.857	1.328	**0.733**

In the table the BFOA has the best result in the problem of 50 dimensions.

5.3.7 Ackley's function

The Global minimum $f(0, 0, 0, 0, ...0) = 0$

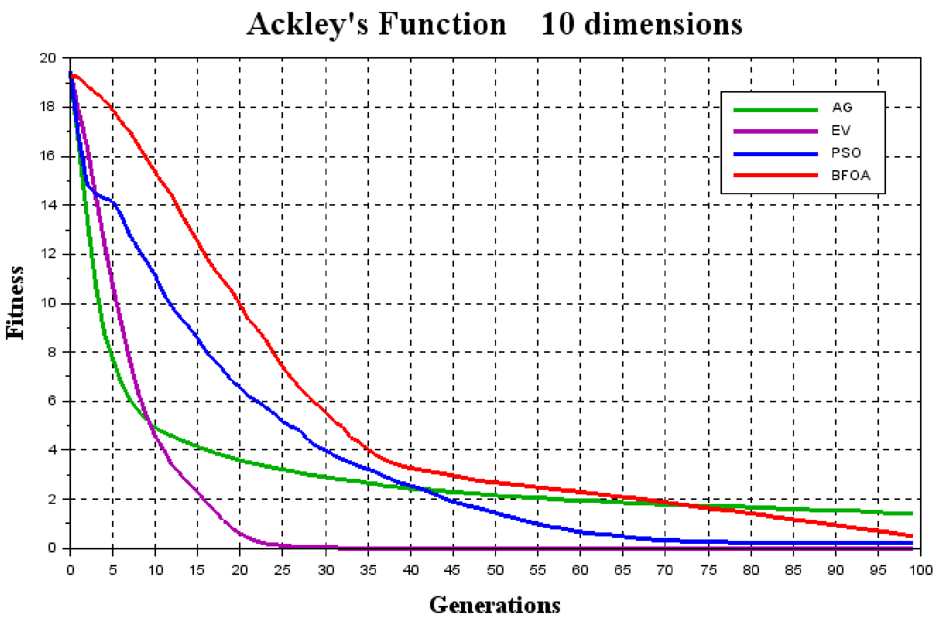

Figure 20: *Ackley's function with search space of 10 dimensions.*

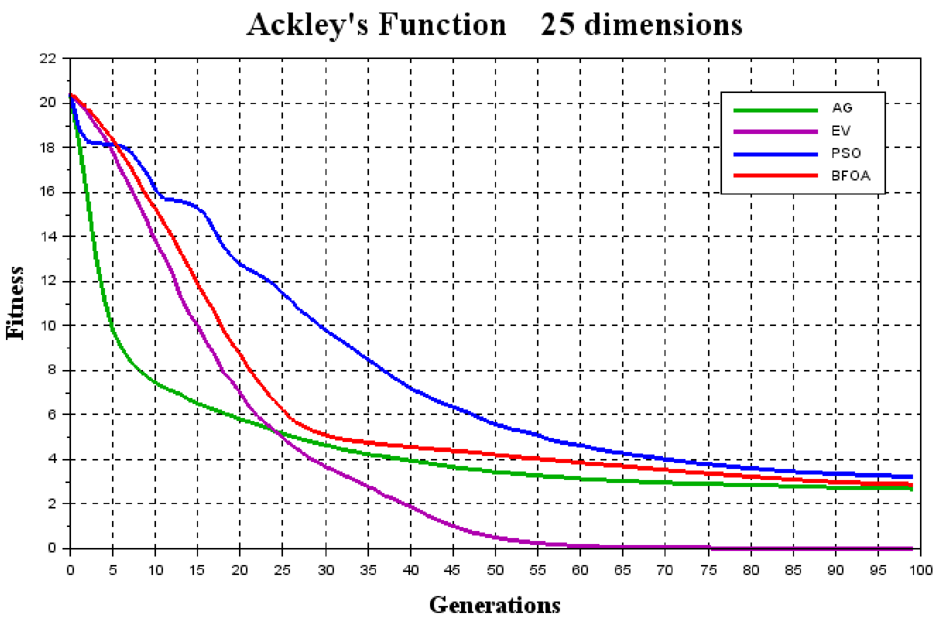

Figure 21: *Ackley's function with search space of 25 dimensions.*

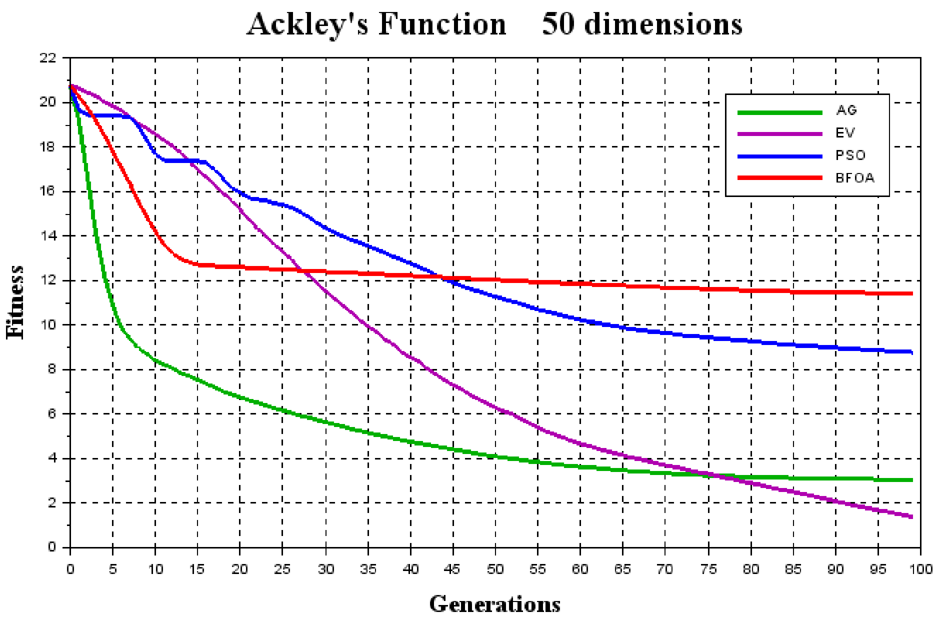

Figure 22: *Ackley's function with search space of 50 dimensions.*

Table 8: Ackley's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	1.424	0.249	1.563	0.048	0.168
	25	10000	2.682	0.411	**3.029**	**0.077**	1.753
	50	10000	3.039	**0.155**	**5.256**	0.043	2.547
EvoNorm	10	10000	**4.94**$\times 10^{-7}$	**9.26**$\times 10^{-8}$	2.485	0.046	**3.48**$\times 10^{-7}$
	25	10000	**1.09**$\times 10^{-3}$	**1.02**$\times 10^{-4}$	5.687	0.132	**0.001**
	50	10000	**1.378**	0.812	10.532	0.038	**0.566**
PSO	10	10000	0.191	0.187	**1.499**	**0.034**	0.003
	25	10000	3.219	0.497	3.148	0.094	0.611
	50	10000	8.777	1.540	5.578	**0.032**	6.396
BFOA	10	13259	0.481	0.053	1.747	0.052	0.156
	25	49172	2.868	0.044	8.844	0.299	1.827
	50	98433	11.421	6.888	25.275	1.641	4.534

In the table the BFOA has a good results, Not the best result.

5.3.8 Langermann's function

Local minimum are unevenly distributed

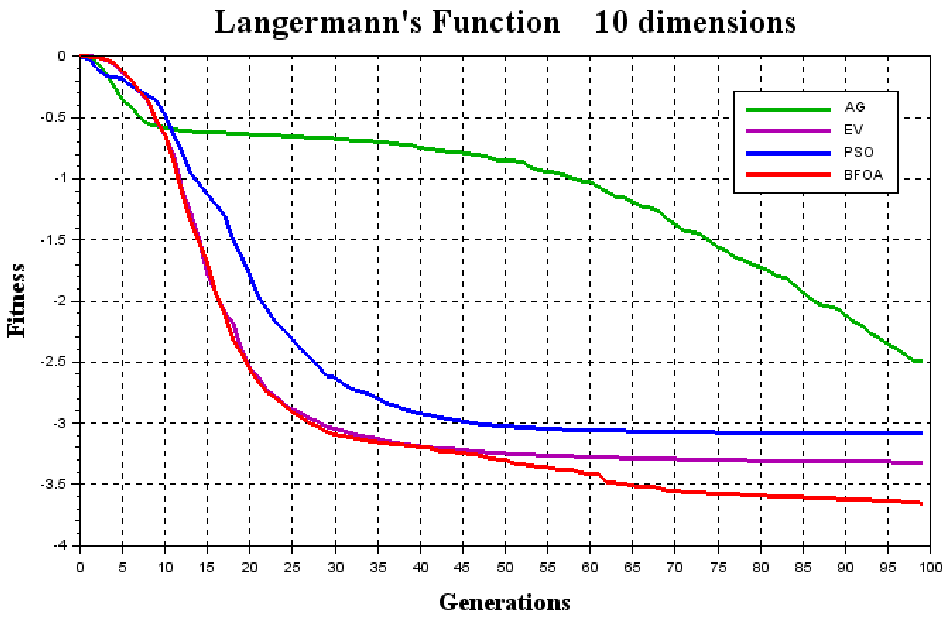

Figure 23: *Langermann's function with search space of 10 dimensions.*

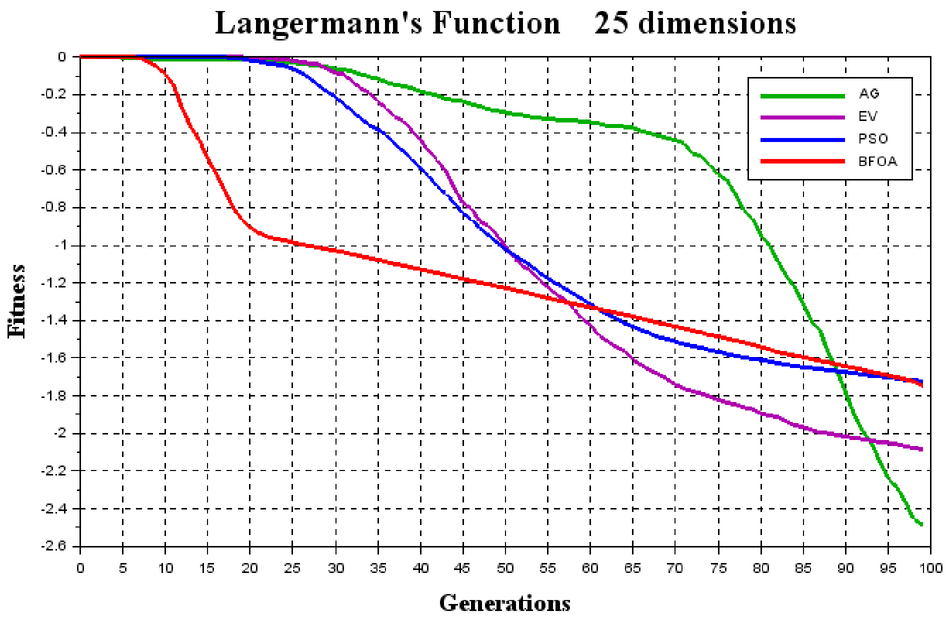

Figure 24: *Langermann's function with search space of 25 dimensions.*

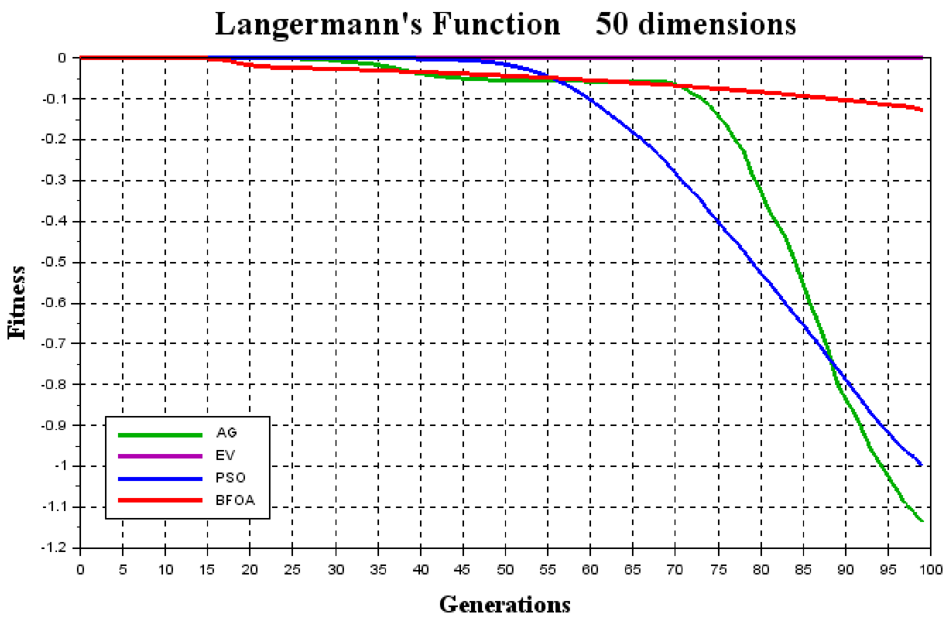

Figure 25: *Langermann's function with search space of 50 dimensions.*

Table 9: Langermann's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	-2.494	1.827	3.252	**0.032**	-4.591
	25	10000	**-2.485**	0.472	**6.629**	0.183	-3.757
	50	10000	**-1.138**	0.343	**12.101**	**0.076**	**-1.777**
EvoNorm	10	10000	-3.317	**0.345**	4.199	0.164	-4.957
	25	10000	-2.084	0.297	9.211	0.186	**-4.416**
	50	10000	-0.001	**0.001**	17.380	0.080	-0.022
PSO	10	10000	-3.081	1.084	**3.200**	0.076	**-4.957**
	25	10000	-1.723	**0.182**	6.664	**0.130**	-1.951
	50	10000	-0.999	0.056	12.358	0.187	-1.244
BFOA	10	13046	**-3.653**	0.995	4.091	0.223	-4.915
	25	47372	-1.746	0.436	26.096	1.876	-3.888
	50	79689	-0.127	0.038	77.088	2.020	-0.267

5.3.9 Michalewicz's function

Global minimum
n=10 is -9.66
other number is unknown

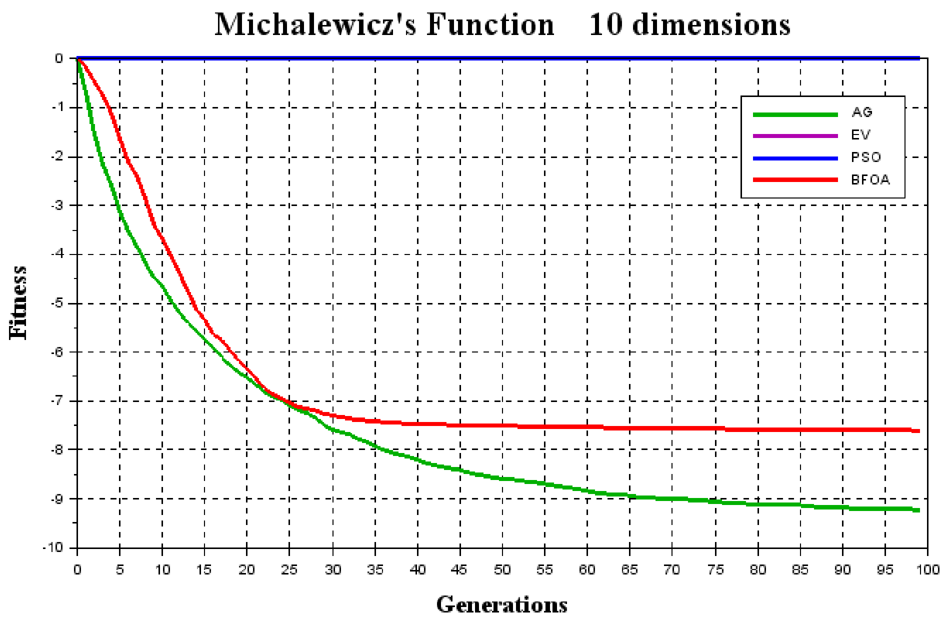

Figure 26: *Michalewicz's function with search space of 10 dimensions.*

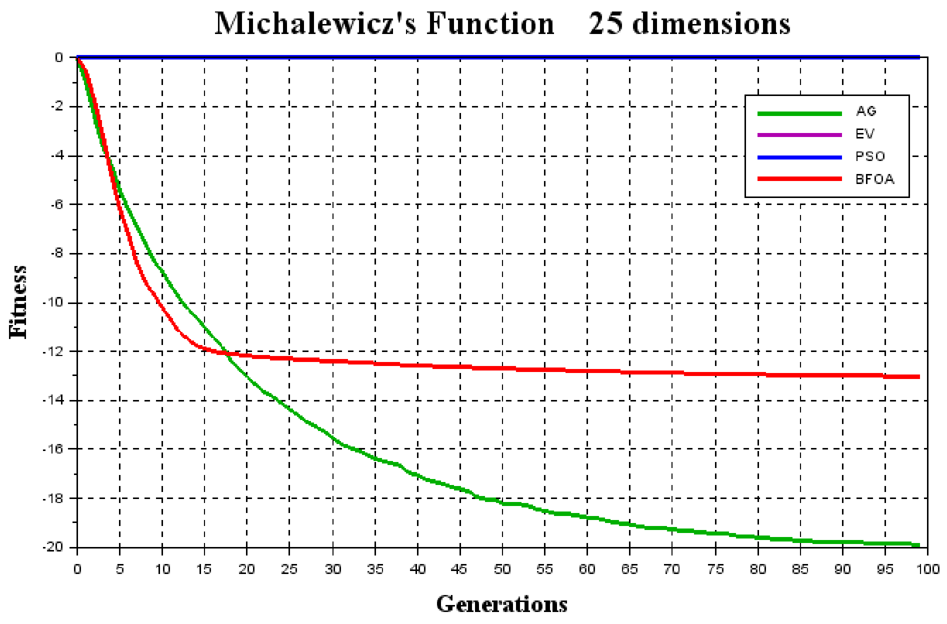

Figure 27: *Michalewicz's function with search space of 25 dimensions.*

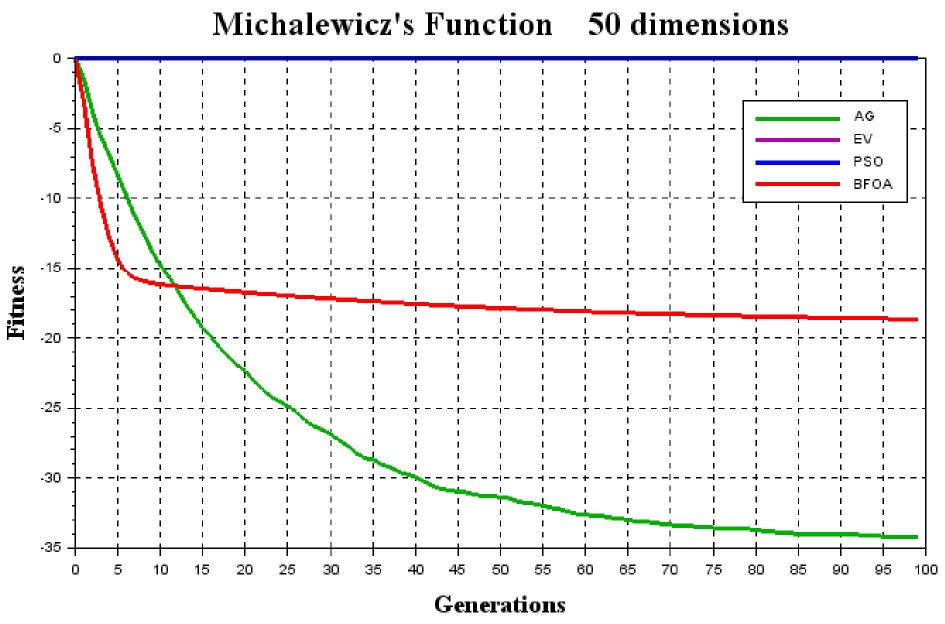

Figure 28: *Michalewicz's function with search space of 50 dimensions.*

Table 10: Michalewicz's function

Algorithm	**Dim**	**Mean Evaluations**	**Mean Best Fitness**	**Standard Deviation of Fitness**	**Time(s)**	**Standard Deviation of Time**	**Best Fitness**
GA	10	10000	**-9.218**	0.143	1.687	0.038	**-9.533**
	25	10000	**-19.896**	1.404	**3.393**	**0.023**	**-21.599**
	50	10000	**-34.237**	1.195	**6.306**	**0.019**	**-37.086**
EvoNorm	10	10000	-1.31×10^{-45}	$\mathbf{1.56}\times10^{-61}$	2.763	**0.024**	-1.31×10^{-45}
	25	10000	-3.38×10^{-44}	$\mathbf{9.96}\times10^{-60}$	6.404	0.148	-3.38×10^{-44}
	50	10000	-2.50×10^{-43}	$\mathbf{1.19}\times10^{-58}$	12.412	0.047	-2.50×10^{-43}
PSO	10	10000	-1.31×10^{-45}	$\mathbf{1.56}\times10^{-61}$	**1.607**	0.035	-1.31×10^{-45}
	25	10000	-3.38×10^{-44}	$\mathbf{9.96}\times10^{-60}$	3.483	0.114	-3.38×10^{-44}
	50	10000	-2.50×10^{-43}	$\mathbf{1.19}\times10^{-58}$	6.595	0.061	-2.50×10^{-43}
BFOA	10	13871	-7.599	0.397	1.976	0.065	-8.726
	25	46022	-13.018	0.793	10.563	0.215	-15.591
	50	88104	-18.653	2.714	33.868	0.643	-24.672

In this function the GA has the better results, the other Algorithm with good results is the BFOA.

5.3.10 Deceptive's function

The Global minimum
to n=10: - 0.158489
to n=25: - 0.076146
to n=50: - 0.043734

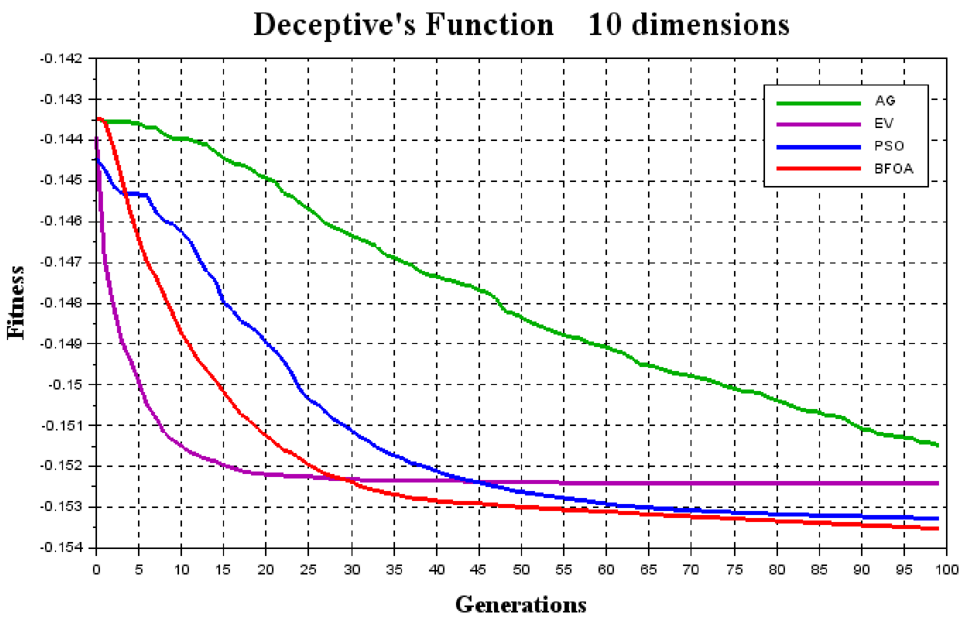

Figure 29: *Deceptive's function with search space of 10 dimensions.*

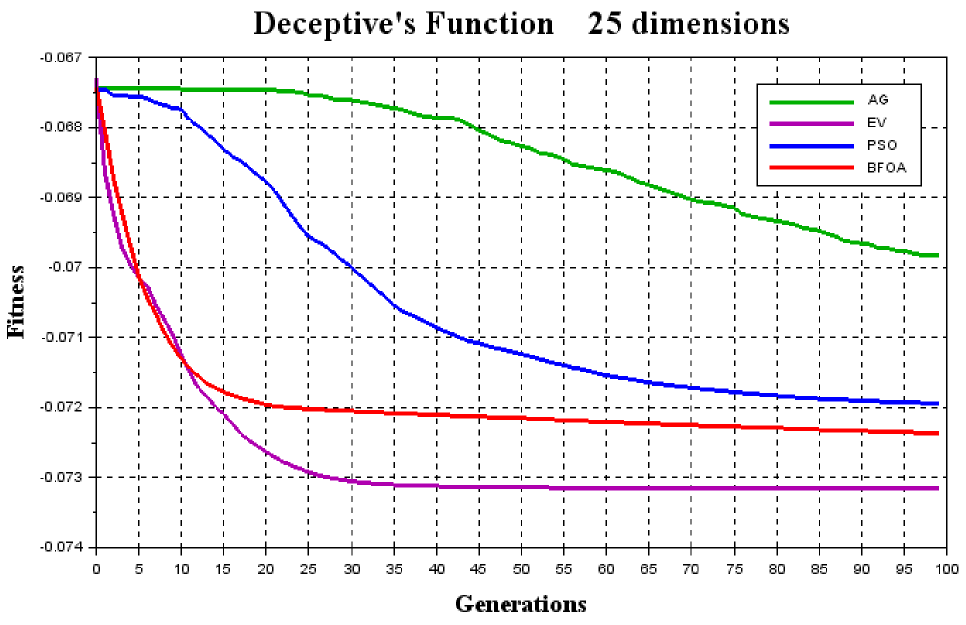

Figure 30: *Deceptive's function with search space of 25 dimensions.*

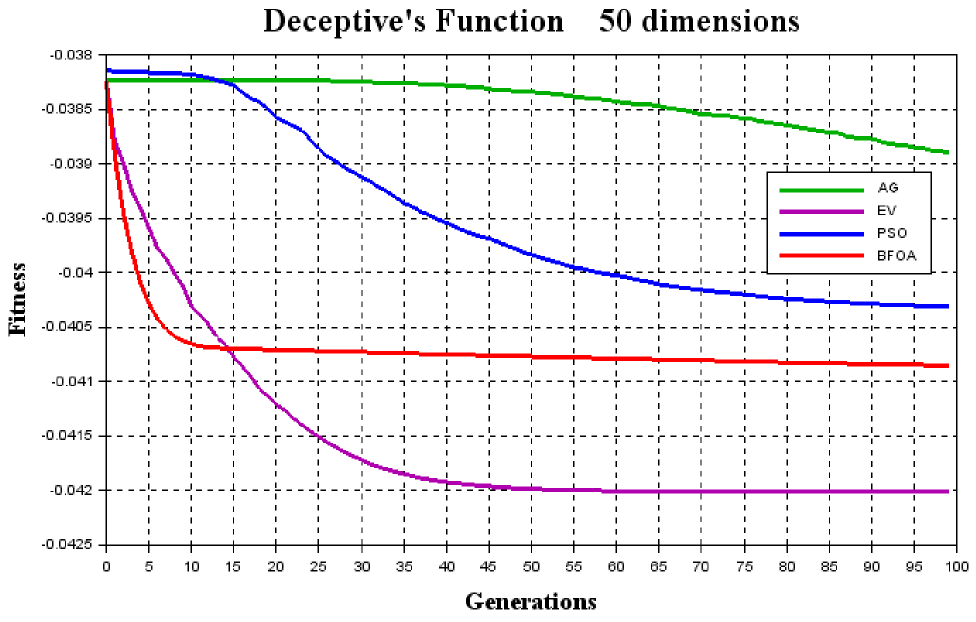

Figure 31: *Deceptive's function with search space of 50 dimensions.*

Table 11: Deceptive's function

Algorithm	Dim	Mean Evaluations	Mean Best Fitness	Standard Deviation of Fitness	Time(s)	Standard Deviation of Time	Best Fitness
GA	10	10000	-0.151	**4.86**$\times 10^{-4}$	2.754	0.066	-0.153
	25	10000	-0.070	0.001	**5.966**	0.055	-0.071
	50	10000	-0.039	8.26×10^{-5}	**11.386**	0.121	-0.040
EvoNorm	10	10000	-0.152	0.001	3.673	0.061	-0.154
	25	10000	**-0.073**	1.88×10^{-4}	8.493	0.070	**-0.074**
	50	10000	**-0.042**	**6.17**$\times 10^{-5}$	16.588	0.112	**-0.042**
PSO	10	10000	-0.153	0.002	**2.659**	**0.035**	-0.155
	25	10000	-0.072	0.001	6.035	**0.054**	-0.073
	50	10000	-0.040	2.43×10^{-4}	11.639	**0.099**	-0.041
BFOA	10	12473	**-0.154**	0.001	3.215	0.092	**-0.156**
	25	44429	-0.072	**6.11**$\times 10^{-5}$	21.905	0.689	-0.073
	50	86950	-0.041	0.001	77.971	5.848	-0.042

To this function the algorithms has good result, very close to the global minimum, including the BFOA, and has a good DS of the fitness

6 Conclusion

In this work presents a BFOA with some changes for improvement, as was seen in the comparisons of each function with other algorithms, the results are similar, this means that the performance of the algorithm is good. The problems encountered are the number of evaluations needed to reach an acceptable result and the computation time employed, although it should be noted that these changes have improved performance to previous versions.[10, 11, 12, 13, 14]

References

[1] K. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *Control Systems, IEEE*, vol. 22, pp. 52–67, Jun 2002.

[2] "Cooperative foraging and search," in *Biomimicry for Optimization, Control, and Automation*, pp. 765–828, Springer London, 2005.

[3] J. Brownlee, "Genetic algorithm," in *Clever Algorithms: Nature-Inspired Programming Recipes*, pp. 257–264, LuLu, 2011.

[4] L. Torres, "Evonorm: easy and effective implementation of estimation of distribution algorithms," *Research in Computing Science*, vol. 23, pp. 75–83, 2006.

[5] J. Brownlee, "Particle swarm optimization," in *Clever Algorithms: Nature-Inspired Programming Recipes*, pp. 232–237, LuLu, 2011.

[6] S. Muller, J. Marchetto, S. Airaghi, and P. Kournoutsakos, "Optimization based on bacterial chemotaxis," *Evolutionary Computation, IEEE Transactions on*, vol. 6, pp. 16–29, Feb 2002.

[7] J. Brownlee, "Bacterial foraging optimization algorithm," in *Clever Algorithms: Nature-Inspired Programming Recipes*, pp. 257–264, LuLu, 2011.

[8] S. Das, A. Biswas, S. Dasgupta, and A. Abraham, "Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications," in *Foundations of Computational Intelligence Volume 3* (A. Abraham, A.-E. Hassanien, P. Siarry, and A. Engelbrecht, eds.), vol. 203 of *Studies in Computational Intelligence*, pp. 23–55, Springer Berlin Heidelberg, 2009.

[9] G. Mahapatra and S. Banerjee, "Article: A study of bacterial foraging optimization algorithm and its applications to solve simultaneous equations," *International Journal of Computer Applications*, vol. 72, pp. 1–6, June 2013. Full text available.

[10] A. Biswas, S. Dasgupta, S. Das, and A. Abraham, "Synergy of pso and bacterial foraging optimization a comparative study on numerical benchmarks," in *Innovations in Hybrid Intelligent Systems* (E. Corchado, J. Corchado, and A. Abraham, eds.), vol. 44 of *Advances in Soft Computing*, pp. 255–263, Springer Berlin Heidelberg, 2007.

[11] J. Dang, A. Brabazon, M. ONeill, and D. Edelman, "Estimation of an egarch volatility option pricing model using a bacteria foraging optimisation algorithm," in *Natural Computing in Computational Finance* (A. Brabazon and M. ONeill, eds.), vol. 100 of *Studies in Computational Intelligence*, pp. 109–127, Springer Berlin Heidelberg, 2008.

[12] N. Kushwaha, V. S. Bisht, and G. Shah, "Article: Genetic algorithm based bacterial foraging approach for optimization," *IJCA Proceedings on National Conference on Future Aspects of Artificial intelligence in Industrial Automation 2012*, vol. NCFAAIIA, pp. 11–14, May 2012. Full text available.

[13] V. Sharma, S. S. Pattnaik, and T. Garg, "Article: A review of bacterial foraging optimization and its applications," *IJCA Proceedings on National Conference on Future Aspects of Artificial intelligence in Industrial Automation 2012*, vol. NCFAAIIA, pp. 9–12, May 2012. Full text available.

[14] E. S. A. M. Salamh, "Article: Synergy of particle swarm optimization and bacterial foraging for sssc damping controller design," *Advances in Energy Engineering Advances in Energy Engineering 2013*, no. 1, 2013.

7 ANNEX

```
function [Ibest, P, ce, time, SbestG]=Algoritmo(TI, TR, NF)

tic ()
   S=TI;
   p=TR;

rt=0

      if TR==10 then
         Nc=3
         Ned=3
         NG=3
      end

      if TR==25 then
         Nc=4
         Ned=4
         NG=6
      end

      if TR==50 then
         Nc=4
         Ned=4
         NG=10
      end

//NÚMERO DE GENERACIONES: PARA 10=3 ; PARA 25=6 ; PARA 50=10


   Ns=15
   Nre=4
   Sr=S/2;
   Ped=.1

   conjbe=zeros(10000,1)
   ce=1

   [MAX,MIN]=parametros(NF)
   P=iniBFOA(p,S,Nc,MAX,MIN)

   for i=1:S
      J(i,1)=feva (P(:,i,1),NF,TR)
      Jibest(i,1)=J(i,1)
      Pibest(:,i)=P(:,i,1)
   end

   [Y,I]=min(Jibest)
   Jbest=Y
   Pbest=P(:,I)

conjbe(ce)=Jbest

   alpha=2*Nre*Nc*NG;//parametro para controlar el descenso del c(i)
   gama=(abs(MAX)+abs(MIN))*0.15//3/2/15  ideal .15
   lambda=(abs(MAX)+abs(MIN))*0.00075
   qw=(gama-lambda)/alpha //.25/.15/.1/.05 ideal .15
   //qw=(gama-0.15)/alpha //.25/.15/.1/.05 ideal .15
   c=ones(S,Nre)*gama

      for cl=1: NG
[P,J,Jibest,Pibest,Jbest,Pbest,conjbe,c,ce]=Loops(P,J,Jibest,Pibest,Jbest,Pbest,Ned,Nre,Nc,S,conjbe,c,ce)
   end
time=toc()

[das2,auxind]=gsort(conjbe,'g','d');
das=das2(das2~=0);
```

```
[m,n]=size(das)

SbestG=zeros(100,1)
SbestG(1)=das(1)
SbestG(100)=das(m)

r=m/100

for t=1:98
    SbestG(t+1)=das(ceil(r*t))
end

[ce,w]=size(conjbe)
Ibest=Jbest;

//j=1:1:100;
  //  plot(j-1,SbestG(j))

endfunction

//*****************************************************************************************************//
//****************************FUNCION LOOP**********************************************//
//*****************************************************************************************************//

function [P, J, Jibest, Pibest, Jbest, Pbest, conjbe, c, ce]=Loops(P, J, Jibest, Pibest, Jbest, Pbest, Ned, Nre, Nc, S,
conjbe, c, ce)

for el=1:Ned //Eliminación y dispersión loop
    for ka=1:Nre //Reproducción loop
        for j=1:Nc //Chemotaxis loop
            for ii=1:S // bacteria loop

            Delta(:,ii)=(2*round(rand(p,1))-1).*rand(p,1);
            fi(:,ii)=((c(ii,ka)*Delta(:,ii))/sqrt(Delta(:,ii)'*Delta(:,ii)))
            P(:,ii,j+1)=P(:,ii,j)+fi(:,ii)
            J(ii,1)=feva (P(:,ii,j+1),NF,TR)
                    ce=ce+1

            if J(ii,1)<Jbest then
                    Jbest=J(ii,1)
                    conjbe(ce)=Jbest
                    Pbest=P(:,ii,j+1)
            end

             if J(ii,1)<Jibest(ii,1)
                    Jibest(ii,1)=J(ii,1)
                    Pibest(:,ii)=P(:,ii,j+1)
             end

                m=0;
                    while m<Ns
                        m=m+1;
                            if J(ii,1)<=Jibest(ii,1) then
                                 P(:,ii,j+1)=P(:,ii,j+1)+fi(:,ii)
                                 J(ii,1)= feva((P(:,ii,j+1)),NF,TR);
                                 ce=ce+1

                                        if J(ii,1)<Jbest then
                                            Jbest=J(ii,1)
                                            conjbe(ce)=Jbest
                                            Pbest=P(:,ii,j+1)
                                        end
```

```
                                      if J(ii,1)<=Jibest(ii,1)
                                          Jibest(ii,1)=J(ii,1)
                                          Pibest(:,ii)=P(:,ii,j+1)
                                      end
                                else
                                    m=Ns;
                                end
                            end//while
                        end//i
                  if c>lambda
                  c=c-qw;
                  end
               end//j

            //step 6
            [Jibest,auxind]=gsort(Jibest,'g','i');
            Pibest(:,:)=Pibest(:,auxind)
                    for u=1:Sr
                           Pibest(:,u+Sr,1)=Pibest(:,u,1);
                           Jibest(u+Sr,1)=Jibest(u,1)
                    end

            P(:,:,1)=Pibest(:,:,1)

                    end//ka

            conteo=0

            //step 7
            for x=1:Sr
                if Ped>rand()
                    conteo=conteo+1
                    for w=1:p
                        Pibest(w,x+Sr,1)=desnormalization(rand(),MIN,MAX)
                    end
                        Jibest(x,1)=feva((Pibest(:,x,1)),NF,TR)
                end

            end

            P(:,:,1)=Pibest(:,:,1)

            end//el

            endfunction

            //*****************************************************************************************//
            //***************************FUNCIÓN DE INICIALIZACIÓN**************************************//
            //*****************************************************************************************//

            function P=iniBFOA(p, S, Nc, MAX, MIN)
               P = zeros(p,S,Nc+1);
                for k=1: p
                    for r=1: S
                        P(k,r,1) = desnormalization(rand(), MIN, MAX);
                    end
                end
            endfunction

            //*****************************************************************************************//
            //**********************************FUNCIONES  AUXILIARES*****************************************//
            //*****************************************************************************************//

            function a=desnormalization(Vn, LMIN, LMAX) //OKKKKKKKKKK
                a = (Vn*LMAX) + (LMIN*(1 - Vn));
```

```
if a > LMAX then
        a = LMAX;
    end
    if a < LMIN then

        a = LMIN;
    end
    endfunction

function a=normalization(Van, LMIN, LMAX) //OKKKKKKKKKK
     a=(Van-LMIN)/((LMAX-LMIN)+0.000001)
    if a>1 then
        a=1
    end
    if a<0 then
        a=0
    end
endfunction

//*************************************************************************************************//
//***************************FUNCIÓN  SELECCION DE FUNCION DE EVALUACION*******************//
//*************************************************************************************************//

function y=feva(X, NF, TR)

select NF

        case 1 then
        y = GenSphere(X);

        case 2 then
        y = GenSchwefel(X);

        case 3 then
        y = GenRastrigins(X);

        case 4 then
        y = GenRosenbrock(X);


        case 5 then
        y = GenSchwefel12(X);

        case 6 then
        y = GenGriewangk(X);

        case 7 then
        y = GenAckley(X);

        case 8 then
        y = GenLangermann(X);

        case 9 then
        y = GenMichalewicz(X);

        case 10 then
       //Deceptive function
        if TR==10 then
              alpha=[0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33, 0.59];
       end

       if TR==25 then
       alpha=[0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33, 0.59,0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33,
0.59,0.3, 0.93, 0.21, 0.31, 0.36];
        end
        if TR==50
       alpha=[0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33, 0.59,0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33,
0.59,0.3, 0.93, 0.21, 0.31, 0.36,0.3, 0.93, 0.21, 0.31, 0.36, 0.29, 0.55, 0.48, 0.33, 0.59,0.3, 0.93, 0.21, 0.31, 0.36, 0.29,
0.55, 0.48, 0.33, 0.59,0.3, 0.93, 0.21, 0.31, 0.36];
        end

       y = GenDeceptive(X,alpha);

       end

endfunction
//***************************************************************************************************//
//**************************FUNCIÓNES  AUXILIAR  DE EVALUACIÓN*******************************//
//***************************************************************************************************//

//Generalized Sphere model (minimization)
//  -100 <= X(i) <= 100
//Minimum f(0,0,0,0,...0) = 0
function y=GenSphere(X)

k = max(size(X));

s=0;

for i=1:k
    s = s + X(i)^2;
end
y = s;

endfunction

//Generalized Schwefel's Problem (minimization)
//  -500 <= X(i) <= 500
//Minimum f(420.9687,...420.9687) = -414.9829*k
function y=GenSchwefel(X)

k = max(size(X));
s = 0;
for i=1:k
    s=s+X(i)*sin(sqrt(abs(X(i))));
end

y=-s;

endfunction

//Generalized Rastrigins function (minimization)
//  -5.12 <= X(i) <= 5.12
//Minimum f(0,0,0,0,...0) = 0
function y=GenRastrigins(X)

k = max(size(X));

s=0;

for i=1:k
    s = s + X(i)^2-10*cos(2*%pi*X(i))+10;
end
y = s;
endfunction
```

```
//Generalized Rosenbrock function (minimization)
//  -10 <= X(i) <= 10
//Minimum f(1,1,1, 1,...1) = 0

function y=GenRosenbrock(X)

k = max(size(X));
s=0;

for i=1:k-1
    s = s + ((100*(X(i+1) - X(i)^2)^2) + (1 -  X(i))^2);
end
y = s;

endfunction

//Generalized Schwefel's Problem 1.2 (minimization)
//  -100 <= X(i) <= 100
//Minimum f(0,0,0,0,0,...0) = 0
function y=GenSchwefel12(X)
k = max(size(X));
s=0;
for i=1:k
    s2=0;
    for j=1:i
        s2 = s2 + X(j);
    end
    s = s + s2^2;
end

y = s;

endfunction

//Generalized Grienwangk's function
//  -600 <= X(i) <= 600
//Minimum f(0,0,0,0,0,...0) = 0
function y=GenGriewangk(X)
n = max(size(X));

    s1=0;
    for j=1:n
        s1 = s1 + (X(j)^2);
    end
    s2=1;
    for j=1:n
        s2 = s2 * (cos(X(j)/sqrt(j)));
    end

    y = (1/4000)*s1 - s2 + 1;

endfunction

//Generalized Ackley's function
//  -32.768 <= X(i) <= 32.768
//Minimum f(0,0,0,0,0,...0) = 0
function y=GenAckley(X)
    //Recpommended:
    a=20; b=0.2; c=2*%pi;
n = max(size(X));

    s1=0;
    for j=1:n
        s1 = s1 + (X(j)^2);
    end
    s2=0;
    for j=1:n
        s2 = s2 + cos(c*X(j))
    end

    y = -a*exp(-b*sqrt((1/n)*s1)) - exp((1/n)*s2) + a + exp(1);

endfunction

//Generalized Langermann's function
//  -1 <= X(i) <= 10
//Local minimum are unevenly distributed
function y=GenLangermann(X)
    //Recpommended:

    n = max(size(X));
    a = [3 5 2 1 7]; b = [5 2 1 4 9]; c = [1 2 5 2 3];

s=0;
for ii=1:5

    s1=0;
    for j=1:n
        s1 = s1 + (X(j)-a(ii))^2;
    end

    s2=0;
    for j=1:n
        s2 = s2 + (X(j)-b(ii))^2
    end

    s = s + c(ii)*exp(-(1/%pi)*s1)*cos(%pi*s2);



end //for ii

    y=s;

endfunction

//Generalized Michalewics's function
//  0 <= X(i) <= pi
//global minimum n=5 is -4.687
//n=10 is -9.66
//other number is unknown
function y=GenMichalewicz(X)
    //Recommended: m=1;
    //Larger m becomes like search  a needle in the haystack...
   m=1;
    n = max(size(X));

    s=0;
    for j=1:n
        s = s + sin(X(j))*(sin(j*X(j)^2/%pi))^(2*m);
    end

    y=-s;

endfunction

//Deceptive function

function rt=g(x, i, alpha)

    if ((x>=0) & (x<((4/5)*alpha(i)))) then
```

```
        rt=(4/5)-x/alpha(i);
    end

    if (x>((4/5)*alpha(i)) & (x<=alpha(i))) then
        rt=5*x/alpha(i)-4;
    end

    if ((x>alpha(i)) & (x<=((1+4*alpha(i))/5))) then
        rt=(5*(x-alpha(i)))/(alpha(i)-1)+1;
    end

    if ((x>((1+4*alpha(i))/5)) & (x<=1)) then
        rt=((x-1)/(1 - alpha(i)))+4/5;
    end


endfunction

//FIrst case
function y=GenDeceptive(X, alpha)

    vbeta=0.2;
    n = max(size(X));
    s=0;
    for j=1:n
        s = s + g(X(j),j,alpha)
    end
    y=-(s)^vbeta/n;

endfunction

//**********************************************************************************************//
//*********************************PARAMETROS PARA DESNORMALIZACION************************//
//**********************************************************************************************//

function [MAX, MIN]=parametros(NF)

select NF

case 1 then
//GenSphere
MIN=-100;
MAX=100;

case 2 then
//GenSchwefel
MIN=-500;
MAX=500;

case 3 then
//GenRastrigins
MIN=-5.12;
MAX=5.12;

case 4 then
//GenRosenbrock
MIN=-10;
MAX=10;

case 5 then
//GenSchwefel12
MIN=-100;
MAX=100;

case 6 then
//GenGriewangk
```

```
MIN=-600;
MAX=600;

case 7 then
//GenAckley
MIN=-32.768;
MAX=32.768;

case 8 then
//GenLangermann
MIN=-1;
MAX=10;

case 9 then
//GenMichalewicz
MAX=0;
MIN=%pi;

case 10 then
//Deceptive functions
MIN=0;
MAX=1;

end

endfunction
```