



Escuela Superior de Cómputo  
Instituto Politécnico Nacional  
Ingeniería en Sistemas Computacionales



## Evolutionary Computing

### Practice 3: Introduction to Genetic Algorithms

**Professor:** Jorge Luis Rosas Trigueros  
**Student:** Ayala Segoviano Donaldo Horacio  
**Creation date:** October 2th 2021  
**Delivery date:** October 8th 2021

## 1 Introduction

### 1.1 Genetic Algorithms

Genetic algorithms (GAS) are **numerical optimisation algorithms inspired by both natural selection and natural genetics**. Genetic algorithms encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality, these strings which are candidate solutions to the search problem are referred to as **chromosomes**, the alphabets are referred to as **genes** and the values of genes are called **alleles**.

To evolve good solutions and implement natural selection, we need a measure for distinguishing good solutions from bad solutions, it could be an objective function that is a mathematical model or even a computer simulation that allows to choose better solutions over worse ones. This functions is referred to as **fitness**.

Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using the following steps:

1. **Initialization:** The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated.
2. **Evaluation:** Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.
3. **Selection:** it keeps more copies of the solutions with higher fitness values and imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions to worse ones, and many selection procedures have been proposed to accomplish this idea, including **roulette-wheel** selection, stochastic universal selection, ranking selection and tournament selection.
4. **Recombination:** Recombination combines parts of two or more parental solutions to create new, possibly better solutions. There are many ways of accomplishing this.
5. **Mutation:** While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. There are many variations of mutation, but it usually involves one or more changes being made to an individual's trait or traits.
6. **Replacement:** The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as **elitist replacement**, generation-wise replacement and steady-state replacement methods are used in genetic algorithms.

## 1.2 Optimization Functions

In applied mathematics, test functions, known as artificial landscapes, are useful to evaluate characteristics of optimization algorithms, such as:

- Convergence rate.
- Precision.
- Robustness.
- General performance.

In this practice optimization test functions will be useful to observe the efficiency of a genetic algorithm to find critical points in a search area. The ones used to test our genetic algorithms are *Ackley's* function using 2 dimensions and *Rastrigin's* function using 3 dimensions.

## 2 Material and equipment

Following are the hardware and software used during the realization of this practice. *Google Colaboratory* was the tool used for the development of this practice and the next list shows the specifications of the hardware and software provided by Google Colab.

- **Hardware:**

- **CPU:** Intel(R) Xeon(R) CPU @ 2.30GHz
- **Memory:** 12GB
- **Disk:** 108GB

- **Software:**

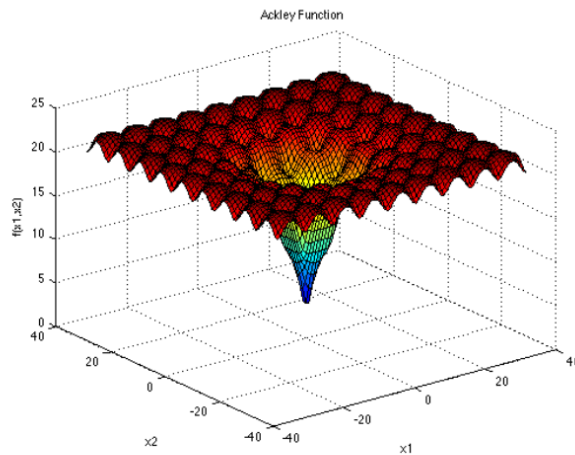
- **Platform used:** *Google colaboratory* was used for this practice
- **Programming language:** Python 3.7.11

### 3 Development

#### 3.1 Optimization of Ackley's function

The first step is to determine a data type to represent chromosomes. In this case, since both functions have more than one dimension, it was decided to use lists of lists, where each sublist will be treated separated, that is, crossover can only happen between corresponding lists. Each list will represent the value of one dimension, so for the Ackley's function which will be optimized in 2 dimensions, a chromosome will be a list of two lists, each list will have binary values (1 or 0) to represent the genes.

The Ackley's function will be the objective function to determine the fitness of each chromosome. Figure 3.1 shows the general form of the function along with its graph for two dimensions. In the formula  $d$  is the number of dimensions, and recommended variable values are:  $a = 20$ ,  $b = 0.2$  and  $c = 2\pi$ .



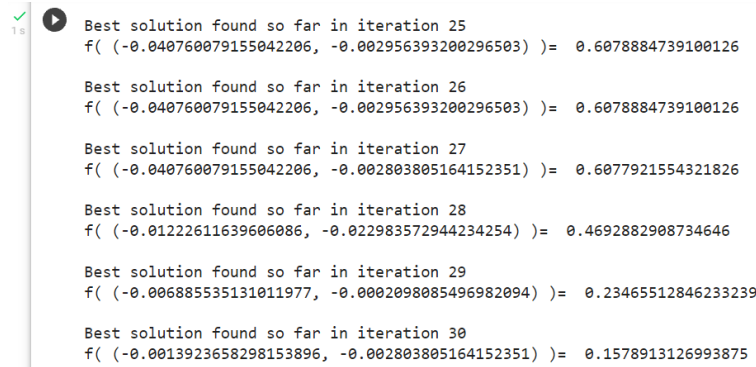
$$f(\mathbf{x}) = -a \exp \left( -b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2} \right) - \exp \left( \frac{1}{d} \sum_{i=1}^d \cos(cx_i) \right) + a + \exp(1)$$

Figure 3.1: Generalization of Ackley's function and 2 dimensions plot.

The approach used for selection is the roulette-wheel, where the fittest elements will have more chances to survive. Crossover will be used for recombination. Mutation will change one bit of have a probability of mutating new chromosomes every time new chromosomes are created. And for replacement the elitist approach will be used, where the fittest element of the population will pass by default to the next generation.

### 3.1.1 Implementation and testing

The genetic algorithm along with the different strategies mentioned above will be coded in Python programming language. Once implemented, the program will be tested to verify a right solution. Code implementation can be found by clicking [here](#).



```

Best solution found so far in iteration 25
f( (-0.040760079155042206, -0.002956393200296503) )= 0.6078884739100126

Best solution found so far in iteration 26
f( (-0.040760079155042206, -0.002956393200296503) )= 0.6078884739100126

Best solution found so far in iteration 27
f( (-0.040760079155042206, -0.002803805164152351) )= 0.6077921554321826

Best solution found so far in iteration 28
f( (-0.01222611639606086, -0.022983572944234254) )= 0.4692882908734646

Best solution found so far in iteration 29
f( (-0.006885535131011977, -0.0002098085496982094) )= 0.23465512846233239

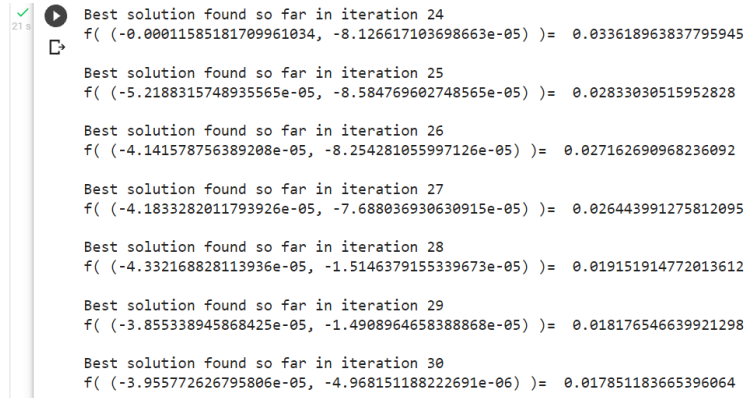
Best solution found so far in iteration 30
f( (-0.0013923658298153896, -0.002803805164152351) )= 0.1578913126993875

```

Figure 3.2: GA's output for Ackley's function.

Figure 3.2 shows the execution of the genetic algorithm to optimize Ackley's function. It shows the last best solutions found after **30 iterations**, each chromosome has **two values of 20 bits each** and a **population of 100** chromosomes. It can be seen that after 30 iterations the algorithms gets very close to the global solution, which is exactly at  $f(0, 0) = 0$  but the answer obtained is  $f((0.000629, 0.00429)) = 0.1859$  which still is very near to the global optimal point.

It can be seen that the result obtained is very close to the global solution, however, the parameters will be increased to verify how much does the parameters affect the outcome.



```

Best solution found so far in iteration 24
f( (-0.00011585181709961034, -8.126617103698663e-05) )= 0.033618963837795945

Best solution found so far in iteration 25
f( (-5.2188315748935565e-05, -8.584769602748565e-05) )= 0.02833030515952828

Best solution found so far in iteration 26
f( (-4.141578756389208e-05, -8.254281055997126e-05) )= 0.027162690968236092

Best solution found so far in iteration 27
f( (-4.1833282011793926e-05, -7.688036930630915e-05) )= 0.026443991275812095

Best solution found so far in iteration 28
f( (-4.332168828113936e-05, -1.5146379155339673e-05) )= 0.019151914772013612

Best solution found so far in iteration 29
f( (-3.855338945868425e-05, -1.4908964658388868e-05) )= 0.018176546639921298

Best solution found so far in iteration 30
f( (-3.955772626795806e-05, -4.968151188222691e-06) )= 0.017851183665396064

```

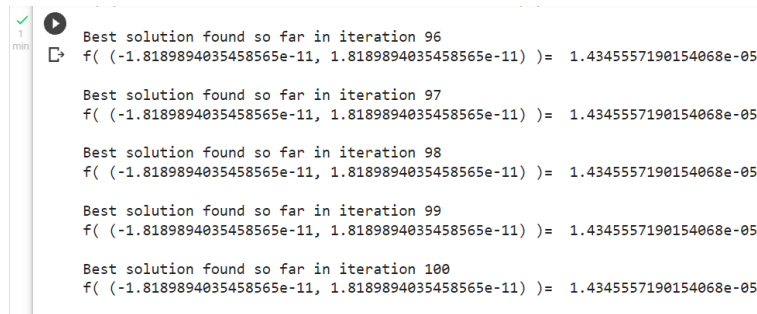
Figure 3.3: GA's output for Ackley's function with increased parameters.

Figure 3.3 shows the execution of the genetic algorithm, it shows the last best so-

lutions found after **30 iterations**, but in this case each chromosome has **two values of 40 bits each** and a **population of 1000** chromosomes. Comparing with last solution, it can be seen that the result improved considerably obtaining a solution of  $f((-1.0559e-05, 1.9667e-06)) = 0.009267$  which is even closer to the global solution. However, it can be easily observed that it takes more work to process more precise parameters.

After running the algorithm several times, one can observe that the solution obtained by the program is similar, even when random parameters are used, the program still gets similar solutions every time it is run.

As a last test, the program will be run again with the same parameters as before, but the only modification will be increasing the number of iterations.



```

Best solution found so far in iteration 96
f( (-1.8189894035458565e-11, 1.8189894035458565e-11) )=  1.4345557190154068e-05

Best solution found so far in iteration 97
f( (-1.8189894035458565e-11, 1.8189894035458565e-11) )=  1.4345557190154068e-05

Best solution found so far in iteration 98
f( (-1.8189894035458565e-11, 1.8189894035458565e-11) )=  1.4345557190154068e-05

Best solution found so far in iteration 99
f( (-1.8189894035458565e-11, 1.8189894035458565e-11) )=  1.4345557190154068e-05

Best solution found so far in iteration 100
f( (-1.8189894035458565e-11, 1.8189894035458565e-11) )=  1.4345557190154068e-05

```

Figure 3.4: GA's output for Ackley's function with increased parameters and more iterations.

Figure 3.4 shows the results for more iterations, in this case 100 iterations were done and it can be seen that the value is way closer to the optimal global than earlier results.

## 3.2 Optimization of Rastrigin's function

The chromosome for this function will be similar to the used in Ackley's function, however, in this case the Rastrigin's will use 3 dimensions, so for the chromosomes a list of three lists will be used, and each containing binary values.

Rastrigin's function in its 3 dimensional form will determine the fitness of each chromosome. Figure 3.5 shows the general form of the Rastrigin's function, as well as a plot of its two dimensional form. Where  $d$  is the number of dimensions.

The approaches for this function will not differ from Ackley's function, the roulette-wheel for selection, crossover for recombination, random bit mutation for new chromosomes and elitism will be applied.

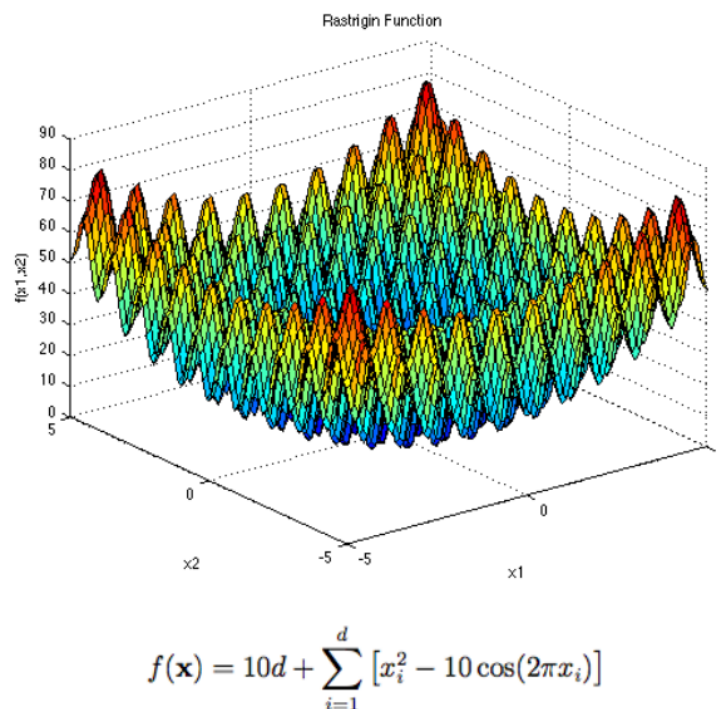


Figure 3.5: Generalization of Rastrigin's function and 2 dimensions plot.

### 3.2.1 Implementation and testing

The genetic algorithm will be coded in python. Now, the program will be tested to verify that it works correctly and that it gets right solutions. The implementation can be found by clicking [here](#).

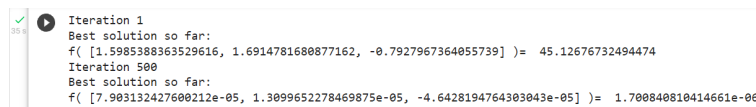
```

Iteration 1
Best solution so far:
f( [0.2106434738836569, 2.40127521819195, 1.2210211946432725] )= 41.17992558288252
Iteration 300
Best solution so far:
f( [0.0012209066608210861, 0.045247648204208346, 0.0006104532531026052] )= 0.40383292836840745

```

Figure 3.6: GA's output for Rastrigin's function.

Figure 3.6 shows output of the execution of the genetic algorithm to optimize Rastrigin's function. It shows the best solutions found after **300 iterations**, each chromosome having **three values of 20 bits each** and a **population of 100 chromosomes**. It can be observed that it gets a point really close to the optimal point which is  $f(0, 0, 0) = 0$ .



```

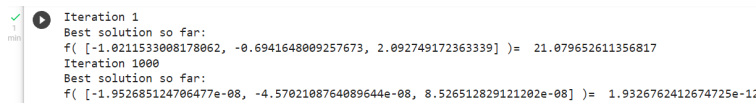
Iteration 1
Best solution so far:
f([1.5985388363529616, 1.6914781680877162, -0.7927967364055739]) = 45.12676732494474
Iteration 500
Best solution so far:
f([7.903132427600212e-05, 1.3099652278469875e-05, -4.6428194764303043e-05]) = 1.700840810414661e-06

```

Figure 3.7: GA's output for Rastrigin's function with increased parameters.

Figure 3.7 shows the best solutions found after **500 iterations**, each chromosome having **three values of 40 bits each** and a **population of 100 chromosomes**. It can be observed that the precision improved drastically compared to the last run, approaching even more to the global optimal point.

Lastly, the last run will be tested for **1000 iterations** to see how close to the global point it gets.



```

Iteration 1
Best solution so far:
f([-1.0211533008178062, -0.6941648009257673, 2.092749172363339]) = 21.079652611356817
Iteration 1000
Best solution so far:
f([-1.952685124706477e-08, -4.5702108764089644e-08, 8.526512829121202e-08]) = 1.9326762412674725e-12

```

Figure 3.8: GA's output for Rastrigin's function with increased parameters and increased iterations.

Figure 3.8 shows the output after 1000 iterations using the more precise parameters. It can be seen that this time, the solution is way closer to the global minimum point.



## 4 Conclusions

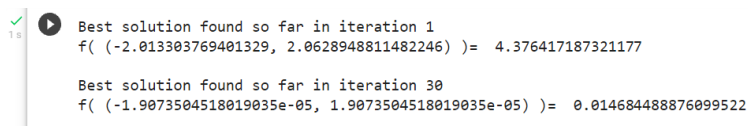
The development of this practice resulted in a better understanding of genetic algorithms, it was understood the main parts of a genetic algorithm, and got familiar with them.

It could be concluded that genetic algorithms are very effective at optimization, in this case, other computational methods could have taken way more resources to solve, however, we could observe, that we could get values really close to the global optimal point

It could also be concluded that the different parameters affect drastically the number of iterations taken to achieve the best solution and that iterations is usually a determinant factor when trying to get more precise results.

### 4.1 Experimentation

On the tests run during this practice, the parameter of mutation was set to a probability of 50% of the times to make a mutation, however, experimenting on Ackley's GA, setting this value to 100%, it is interesting to see how that convergence of the algorithm improved, in this case, 20 bits are used for each dimension of the chromosome, and a population of 100 chromosomes are used, but, even when the algorithm starts pretty far from the global solution, an increased rate of mutation improves the speed of convergence.



```

Best solution found so far in iteration 1
f( (-2.013303769401329, 2.0628948811482246) )=  4.376417187321177

Best solution found so far in iteration 30
f( (-1.9073504518019035e-05, 1.9073504518019035e-05) )=  0.014684488876099522

```

Figure 4.1: GA for Ackleys function with increased mutations.

In figure 4.1 it can be observed that even when the algorithm started with big fitness values, at the end, it reaches points fairly close to the optimal global point. Comparing with figure 3.2 it is better than using 50% of probability of mutation.

## 5 Bibliography

- Sastry K., Goldberg D., Kendall G. (2005) Search Methodologies. Springer, Boston, MA.
- Coley David. (2009). An introduction to genetic algorithms for scientists and engineers. New Jersey. University of Exeter World Scientific.
- Surjanovic Sonja, Bingham Derek (2013). Virtual Library of Simulation Experiments. Burnaby, Canada. Simon Fraser University.