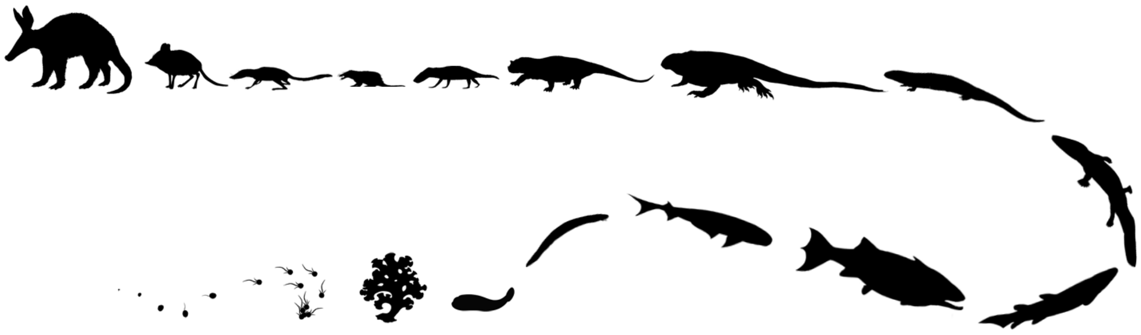
Instituto Politécnico Nacional
Escuela Superior de Cómputo

Evolutionary Computing
(Evolutionary Computation)



Practice No. 2:

Student: García Medina Juan Carlos
Professor: Dr. Rosas Trigueros Jorge Luis

Date Practice: 12-Feb-2019
Date Report: 20-Feb-2019

THEORETICAL FRAMEWORK

Genetic Algorithms

Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems.

Holland's 1975 book Adaptation in Natural and Artificial Systems presented the genetic algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics-inspired operators of crossover, mutation, and inversion. Each chromosome consists of "genes" (e.g., bits), each gene being an instance of a particular "allele" (e.g., 0 or 1). The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single-chromosome ("haploid") organisms; mutation randomly changes the allele values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome, thus rearranging the order in which genes are arrayed. (Here, as in most of the GA literature, "crossover" and "recombination" will mean the same thing.)

Holland's introduction of a population-based algorithm with crossover, inversion, and mutation was a major innovation. (Rechenberg's evolution strategies started with a "population" of two individuals, one parent and one offspring, the offspring being a mutated version of the parent; many-individual populations and crossover were not incorporated until later. Fogel, Owens, and Walsh's evolutionary programming likewise used only mutation to provide variation.) Moreover, Holland was the first to attempt to put computational evolution on a firm theoretical footing (see Holland 1975). Until recently this theoretical foundation, based on the notion of "schemas," was the basis of almost all subsequent theoretical work on genetic algorithms. (1)

Resources and tools

1. Computer with python 2: To run the program, python 2 was used since some parts of the code were not able to run directly on python 3.

2. Tools

   - Tkinter: The Tkinter module ("Tk interface") is the standard Python interface to the Tk GUI toolkit. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

DISCUSSION

The next part is focused on the tests made in the laboratory of a genetic algorithm using the theory seen in class with some operations over the genes.

The main goal is to find a minimum of the function $f(x) = x^2 \cos x$

Originally the chromosomes are 4 bits long, however, with this number it does not finds meaningful results, so I set it up to 32.

```
from Tkinter import *
import math
import random
#Chromosomes are 4 bits long
L_chromosome = 32
N_chains = 2 ** L_chromosome
#Lower and upper limits of search space
a = -20
b = 20
crossover_point = L_chromosome / 2
#...
```

As it is showed in figure 1

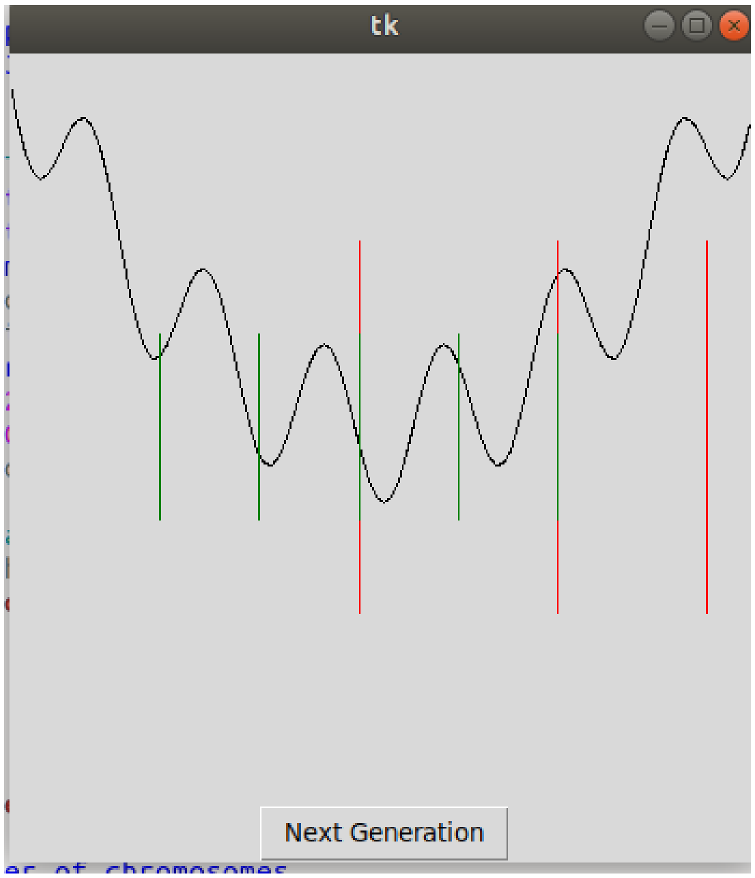


Figure 1: Figure showing initial congiguration

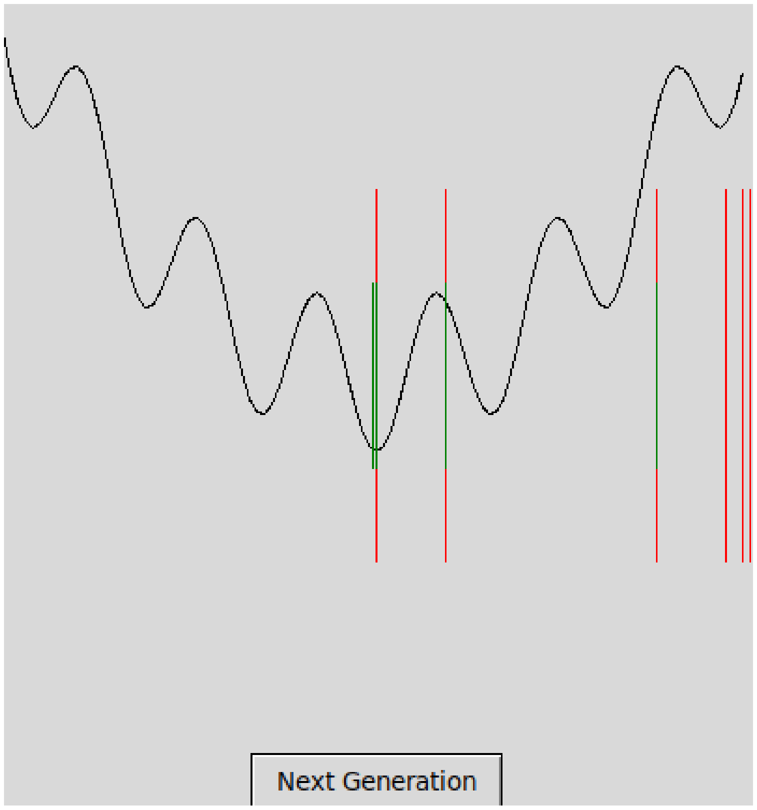
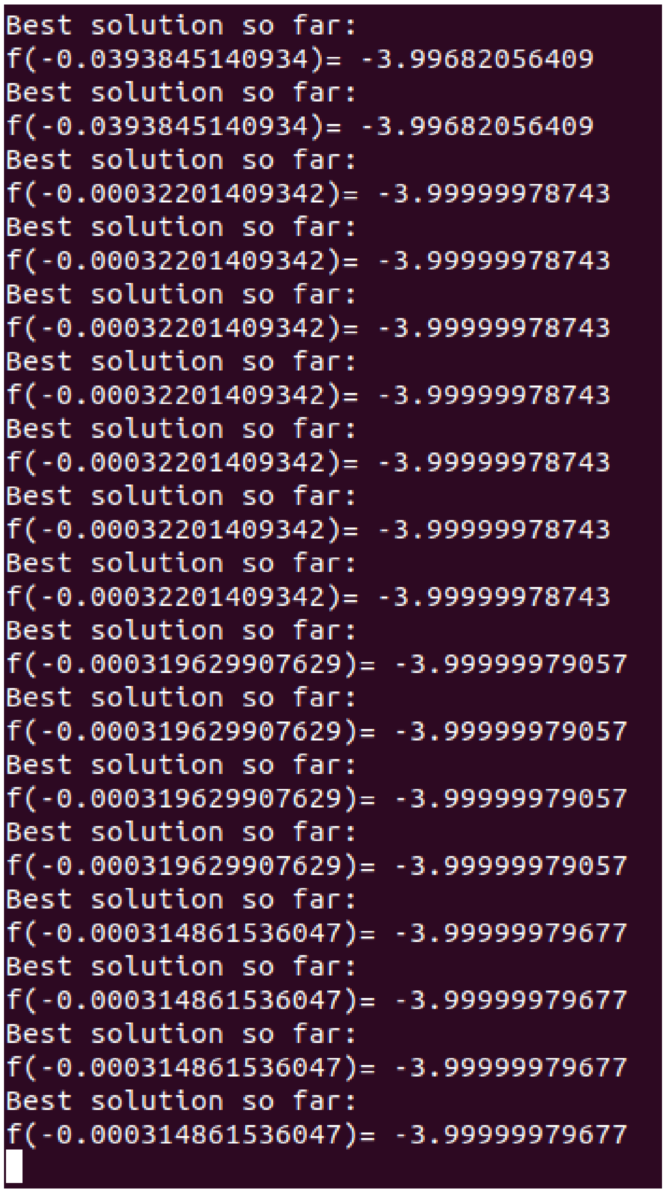


Figure 2: Behavior when probability is changed.

Changing the probability of mutation

With a low `prob_m` it seemed to show a very stationary behavior and setting the highest probability possible: i.e: *1.0* it showed a better behavior:

```
#...
#Number of chromosomes
N_chromosomes = 10
#probability of mutation
prob_m = 1.0
#...
```



```
Best solution so far:
f(-0.0393845140934)= -3.99682056409
Best solution so far:
f(-0.0393845140934)= -3.99682056409
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.00032201409342)= -3.99999978743
Best solution so far:
f(-0.000319629907629)= -3.99999979057
Best solution so far:
f(-0.000319629907629)= -3.99999979057
Best solution so far:
f(-0.000319629907629)= -3.99999979057
Best solution so far:
f(-0.000319629907629)= -3.99999979057
Best solution so far:
f(-0.000314861536047)= -3.99999979677
Best solution so far:
f(-0.000314861536047)= -3.99999979677
Best solution so far:
f(-0.000314861536047)= -3.99999979677
Best solution so far:
f(-0.000314861536047)= -3.99999979677
```

Figure 3: Results after some generations.

Changing the number of chromosomes

If we change the N_chromosomes value to a higher value, then we will have more chances to find the optimal solution in less steps, since population is larger and simultaneously are searching for the solution:

```
#...
#Number of chromosomes
N_chromosomes = 100
#...
```

As you can see, in Figure 4 it shows a hudge quantity of lines, just after changing the number of chromosomes up to 100, therefore we can now see that those lines represent the chromosomes.

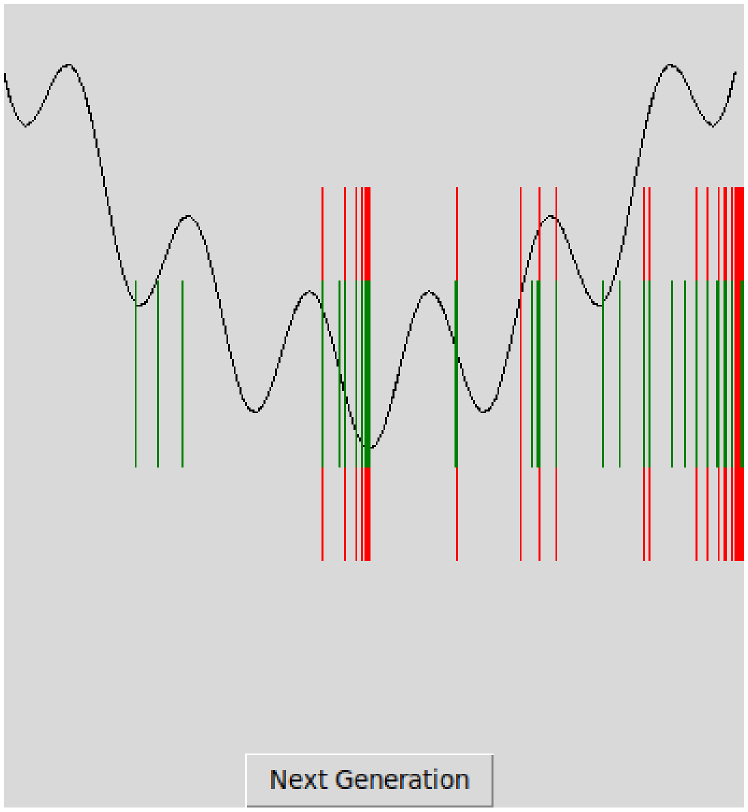


Figure 4: Hundred chromosomes searching in the space

CONCLUSION AND RECOMMENDATIONS

In this practice we were able to interact with a genetic algorithm, and we saw how changing some parameters were determinant with the behavior and the time to find the optimal solution. The accuracy of the algorithm choosing good values for the

paramteres was too good, in fact, in some cases we could see the exact global solution. I would recommend for further practices the next points:

- To use an alternative for the tkinter library
- To give a quick explanation of the code before staring the practice to have more context about it.

REFERENCES

[1] Mitchell, M., 1999, An introduction to genetic algorithms: MIT press.