II Workshop on Modeling and Simulation for Science and Engineering (II WMSSE) IOP Publishing
Journal of Physics: Conference Series **1448** (2020) 012020 doi:10.1088/1742-6596/1448/1/012020

Genetic algorithms for mathematical optimization

J M García[1], **C A Acosta**[1], **and M J Mesa**[2]
[1] Grupo de Estudio y Desarrollo de Software, Universidad del Quindío, Armenia, Colombia
[2] Corporación Universitaria Empresarial Alexander Von Humboldt, Armenia, Colombia

E-mail: jmgarcia@uniquindio.edu.co

Abstract. The inability to find the solution in engineering problems has led to a large part of the scientific community developing indirect and alternative techniques to find optimization problem-solving. Genetic algorithms are looking for models based on the natural and genetic selection process, which optimizes a population or set of possible solutions to deliver one that is optimal or at least very close to it in the sense of a fitting function. In this work, we derive and evaluate a method based on genetic algorithms to find the relative maximum of differentiable functions that are difficult to find by analytical methods. We build a library in Python that includes different components from genetic algorithms. The test problems include finding the maximum or minimum of functions in one and two dimensions.

1. Introduction

John Holland developed genetic algorithms (GAs) in the 1960s [1-3]. They are algorithms based on natural selection and natural laws of genetics, which aims to solve optimization problems. These algorithms have the following iterative process to find the optimal solution [4].

- Properly represent the encoding of the problem. Most of the problems use binary encoding.
- Evaluate each individual with a fitness function or target function, which determines the value or performance of each solution.
- Choose a configuration selection strategy, which will be in charge of the construction of the new population (new generation).
- Choose a mechanism to implement the genetic crossover operator.
- Build a mechanism to implement the genetic mutation operator.

However, in mathematics and engineering, it is very likely to encounter problems that cannot be solved by a direct method. Gas may be the solution for those problems with the right encoding, they can come to several solutions very close to the optimal.

GAs have been traditionally used in different branches of engineering in optimization problems, as shown by Bhoskar [5], Khan [6], and Shi [7]. Recently, GAs have become a high-complexity search technique and an optimization math problem tool. Authors such as Kiyoumarsi [8], McCall [9], and Assis [10] have used this technique to solve mathematical problems.

In this work, we evaluated and developed a method based on GAs to find the relative maximum of differentiable functions that are difficult to find by analytical methods. To derive the method, we first defined and implemented in Python the building blocks of a GAs, considering the encoding and the fitting function. Then, to validate the method we proposed the optimization of tow test functions.

2. Methods

2.1. Maximum and minimum values of a function

A function f has a relative maximum value in the number c if there is an open interval containing c where f it is defined, such that $f(c) \geq f(x)$ for everything x in the interval. On the other hand, f has a relative minimum value in the number c if there is an open interval containing c where f it is defined, such that $f(c) \leq f(x)$ for everything x in the interval [11].

However, in the study of the calculus, we can find that these relative maximum or minimum values are usually obtained with the help of the derivative of the function, finding what is known as hotspots, these critical points are the points where the derivative is it does zero. However, the definition of a relative maximum does not make a warning about if f function is differentiable and even more if f is continuous. Then we can find two drawbacks: First, we can have functions that are continuous and differentiable at all points but, it is very difficult to find the derivative in analytical form. Secondly and in the worst case, we can find functions that are continuous at all points in the range but not differentiable at some or all points, such as the Riemann and Weierstrass functions [12].

We will take two functions as an example, the first case, we will use simple genetic algorithm (AG) to find the maximum of one function in $\mathbb{R}^2$ and in the second case, we will find the maximum of one function in $\mathbb{R}^3$.

To illustrate the method, we will take the following example: Find the maximum of the function f in the interval $(0;\ 1)$:

$$f(x) = e^{0.005\sin(x) - \sinh\left(\frac{\sin(10.005x)x}{2}\right)} \sin(50.5x)\cos(x) + \cos(0.325x) \tag{1}$$

Equation (1) is continuous in the range $(0;\ 1)$, because it is the addition and multiplication of an exponential function and several sine and cosine trigonometric functions. However, it is a very complex function of deriving, so finding the maximum value is relatively complex by analytical methods.

2.2. Simple genetic algorithm implementation

To find the maximum value of the Equation (1), we will use the coding model posed by Michalewicz [13], where from an interval a list of zeros and ones that will serve as individuals or chromosomes of the form of Equation (2):

$$(b_1, b_2, \ldots b_n)_2 = \left(\sum_{i=0}^{N} b_i 2^i\right) = x' \tag{2}$$

Each $b_i \in \{1,0\}$ is a chromosome gene, for this example we take a chromosome size of $N = 40$. Then with this result, we convert x' at a value in the interval $(0;\ 1)$, as shown Equation (3):

$$x = L_{low} + x' \frac{L_{up}}{2^N - 1} \tag{3}$$

There, $L_{low} = 0$ is the lower limit of the interval, $L_{up} = 1$ it is the upper limit. The Equation (2) and Equation (3) ensure that any combination of ones and zeros in size N always in this interval, ensuring that the chromosome: $000000\ldots000$ represent the $L_{low} = 0$ and chromosome $111111\ldots111$

represent $L_{up} = 1$, so that any other combination of ones and zeros will always be in the range (L_{low}, L_{up}).

Fixed the coding problem, we proceed to configure the GAs, which will have an initial population of 300 individuals or chromosomes with a maximum number of iterations of 200. A selection scheme per tournament and the recombination to a chosen point in shape mutation is made by randomly choosing a gene and changing its value from 1 to 0 or from 0 to 1 as the case may be, with a probability of 0.01. For the stop criterion, a maximum number of iterations was taken, which was 200. 100 runs of the AG were performed, where this return in most cases the following solution:

$$\text{Optimal\_chromosome} = 01100111100011101100010100110110101101111 \tag{4}$$

This optimal chromosome in Equation (4), codifies the solution $x = 0.40452225290155597$ with the Equation (2) and Equation (3); its performance or fitness is $y = 2.072060602212556$, which, can be observed as one of the green point in the Figure 1. These values compared to the actual values obtained by traditional methods: $x = 0.4048326421$ and $y = 2.07219397877154$, have a total error of 3.1038×10^{-4}. Figure 1 shows the maximum value found and other approximate solutions, which the GA found.

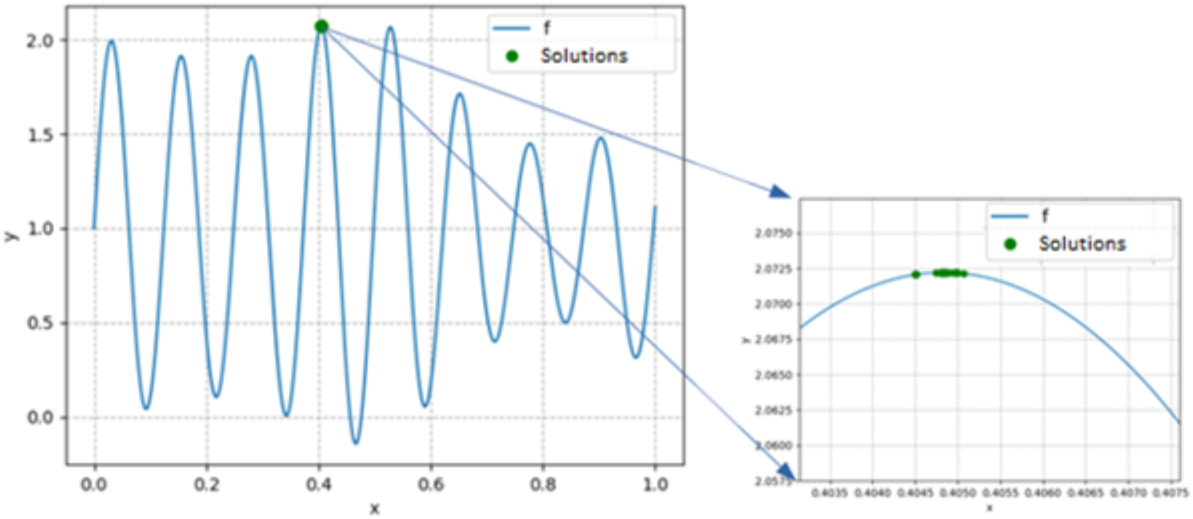

Figure 1. Function f from Equation (1) in the interval $(0,1)$, green dots represent the best solutions found.

Figure 1 shows that the function f has eight maximums reactive in interval (0; 1), in fact, two of them have very similar values, which meant that in some occasions the AG would find the wrong maximum value.

The next problem is to find the maximum of one function g in $\mathbb{R}^3$, at intervals $x \in (-5;\ 5)$ and $y \in (-5;\ 5)$. The function g is defined in Equation (5).

$$g(x,y) = (3/2)e^{\frac{1}{1+(x-1)^2+(y-1)^2}} - (5/2)e^{\frac{1}{1+(1/4)(x+1/2)^2+(1/36)(y-1)^2}} + 2e^{\frac{1}{1+(x-2)^2+(y-2)^2}} + 2e^{\frac{1}{1+(x-1)^2+(y+1)^2}} \tag{5}$$

In this case, each chromosome must have a multiple size of 2, because each chromosome must now account for two components (x, y) for being a function in $\mathbb{R}^3$. In addition, in the target function it receives a two-component vector as a parameter. However, the methodology is the same; a population of 300 individuals was used, a chromosome size of 50 (this number must now be even) and a maximum number of cycles of 1000, otherwise the same algorithm settings were used for the function in $\mathbb{R}^2$. The most common result in 100 runs was Equation (6).

$$\text{Optimal\_Chromosome} = 1011001100111000110001110101100001000111101001010 \quad (6)$$

The solution in Equation (6), corresponds to the point on the plane $(x, y) = (2.00085124, 1.89686467)$, which has a performance of $z = 6.043504718396552$. Figure 2 shows the best solution and some others very close.

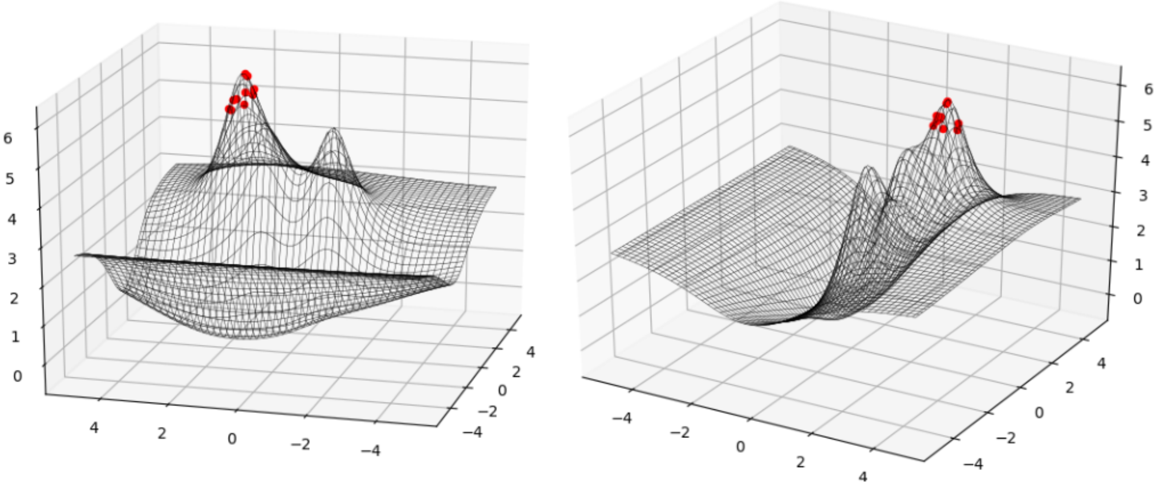

Figure 2. The function g in Equation (5), the red dots represent the best solutions found.

Figure 2 shows two perspectives of the function g, in which we can see two locals maximum in the domain $(-5;\ 5) \times (-5;\ 5)$. In addition, of the optimal solution, the AG found other solutions, which are the other red dots in Figure 2. This particular problem was a more complex problem than the previous one, sometimes the AG found the smallest of the two local highs in the region; usually always towards rapid convergence to find this value or very similar values.

3. Conclusions

In both examples, the algorithm did not spend too much time finding the optimal solution, usually taking two to three minutes to complete the 200 cycles. It is interesting to do the exercise of saving the other solutions that are obtained in each generational cycle, this shows how the algorithm is evolving and how these solutions approach until finding the optimal final solution. These "pre-five" solutions can become useful in engineering, because in many cases, the optimal solution is not feasible or cannot be implemented, due to possible constraints.

Using AG to find the maximum or minimum of real variable functions is just the beginning of the great potential that these algorithms have for solving problems in mathematics. In our case, we show examples that do not require a large computing power; however, GAs can be used in many other areas of mathematics, for example by finding system coefficients of linear equations or solving linear programming problems.

The Python language is an excellent choice for implementing AG. SciPy libraries have tools for handling values, matrices and charts. In addition, the native Python language has excellent list management, which fits very well with the concepts of gene, chromosome, population, etc. In general, the implementation of these concepts is simple, and the handling of objects allows such algorithms to be developed with ease.

Acknowledgments

This work was supported by the Universidad del Quindío, Colombia.

References

[1] Mitchell M 1998 *An introduction to genetic algorithms* (London: MIT press)
[2] Koza J R 1994 *Genetic programming II: Automatic discovery of reusable subprograms* (Cambridge: MIT press)
[3] Goldberg D E 2014 *Genetic algorithms in search, optimization, and machine learning* (London: Addison-Wesley)
[4] García J M, Acosta C A & Hoyos E A 2006 Librería de funciones abstractas para la construcción de algoritmos genéticos con programación funcional *Revista de Investigaciones Universidad del Quindío* **16** 123
[5] Bhoskar M T, Kulkarni M O K, Kulkarni M N K, Patekar M S L, Kakandikar G M, & Nandedkar V M 2015 Genetic algorithm and its applications to mechanical engineering: A review *Materials Today: Proceedings* **2(4-5)** 2624
[6] Khan M Z R and Bajpai A K 2013 Genetic algorithm and its application in mechanical engineering *International Journal of Engineering Research & Technology* **2(5)** 677
[7] Shi L, Da L, Fu H 2005 An application of genetic algorithm in engineering optimization *Current Trends in High Performance Computing and Its Applications* (Berlin: Springer)
[8] Kiyoumarsi F 2015 Mathematics programming based on genetic algorithms education *Procedia-Social and Behavioral Sciences* **192** 70
[9] McCall J 2005 Genetic algorithms for modelling and optimisation *Journal of Computational and Applied Mathematics* **184(1)** 205
[10] De Assis L S, Junior J R, Tarrataca L, Fontoura A R, and Haddad D B 2019 Efficient Volterra systems identification using hierarchical genetic algorithms *Applied Soft Computing* **2019** 105745
[11] Leithold L 1998 El cálculo (México: Oxford University Press)
[12] Zähle M and Ziezold H 1996 Fractional derivatives of Weierstrass-type functions *Journal of Computational and Applied Mathematics* **76(1-2)** 265
[13] Michalewicz Z 2013 *Genetic algorithms+data structures = evolution programs* (Charlotte: Springer Science & Business Media)