Finding the global minimum of Optimization Functions via Genetic Algorithm and comparison to Hill Climbing and Simulated Annealing algorithms

Habasescu Andrei E2

29^{th} of November 2022

Abstract

In this paper we observe the particularities of a Genetic Algorithm and the results of such algorithm to find the global minimum of Optimization Functions, namely **Rastrigin's**, **DeJong's**, **Schwefel's** and **Michalewicz's**[**2**] functions.

In order to apply the the Genetic Algorithm[**1**] to find the global minimum of Optimization Functions we will be using a set of functions particular to this type of algorithm *(Selection, Cross-Over, Mutation)* while also applying these operations on a Population made of 100 individuals, each containing a Bit String made of bits which represent our candidate solutions in binary, while also storing their Decimal Values. As we are using Optimization Functions we will run the algorithm on 5, 10 and 30 dimensions, each instance of the Genetic Algorithm going through 1000 iterations, the overall precision of the algorithm is 10^{-5}.

By the end of this paper we will see that Genetic Algorithms are overall more efficient than any of the heuristic methods such as Hill Climbing and it's variations based on improvement (worst, first, best) as well as the metaheuristic method Simulated Annealing. We will also see that Genetic Algorithms are computing faster and have much better precision than the algorithms mentioned above, fact proven by the data displayed in the Results Section.

1 Introduction

In the following report we will display the results obtained by a Genetic Algorithm while finding the global minimum of Optimization Functions (DeJong's, Schwefel's, Rastrigin's, Michalewicz's) on 5, 10 and 30 dimensions and 1000 iterations. We will also break down the obtained data to determine in what departments does the Genetic Algorithm excel compared to Heuristic Algorithms which have ran on the same Optimization Functions (Time Efficiency, Accuracy of the Results, Deviation).

We will have a population of 100 individuals which will undergo several modifications through the Genetic Algorithm, such as Mutation, Cross-Over and Selection, having a precision of 10^{-5}.

2 Methods

In order to run the Genetic Algorithm run Optimization Functions we will generate strings of bits which will represent possible solutions to the said functions. In order to do so we will have to calculate the bit length of one such solution with the formula:

$$number\ of\ bits = log_2(10^5 \cdot\ (upper\_limit\ -\ lower\_limit))$$

A single candidate is required to be of 5, 10 or 30 dimensions, therefore the length of one such candidate's bit string will be equal to :

$$candidate\ binary\ length = number\ of\ bits \cdot\ dimensions$$

A population has 100 individuals, therefore we will store all of the candidates in a vector and will begin our operations on the original population in order to arrive by the end of 1000 iterations to a plausible solution. In order to generate

2.1 Selection

The selection is a function of the Genetic Algorithm which evaluates each individual of the population and assigns it value in the range *[0;1]* based on how close it is to the the global minimum of the Optimization Function analysed, such value is computed by the Fitness Function. The higher the fitness, the larger the value assigned is.

Once we compute the fitness of each individual, we will place them on a Wheel Of Fortune, which will help us select values to be passed to the next generation of the population based on how fit they are for the analysed Optimization Function. In order to pick a number in range of *[0;1]* we will be using a pseudo-random number generator, in our case the *64-bit version of the Mersenne Twister 19937 Generator.*

Using an auxiliary population we pass the individuals which have been selected by the Wheel of Fortune, on which we will run the operations of Cross-Over and Mutation to create a individuals whose new values might be closer to the desired result.

The best value of the original population will be stored on the first position while also creating another 4 copies on the following position, so as to induce an improvement to the overall quality of the next population through Genetic Operations on the said best candidates by crossing them with other individuals or mutating them to reach a possible new optimum.

2.2 Mutation

Once we have generated a new population which inherited data from the original population through Selection, we have to modify it so as to avoid repetition while also looking for new possible solutions. In order to do so we will try to invert each bit of a bit string with a base chance of 0.05

I have implemented a special adaptive function so as to reduce the odds of mutation based on how close it is to a global or local minimum, determined by the Fitness function. It is as follows :

$$chance = mutation\ probability - (fitness^2 \cdot 0.8 \cdot dimensions/30)$$

This function has an exponential graph, therefore the closer our individual is to the desired result, the smaller our mutation chance becomes, as such we will jump less between values and eventually zero in on a possible solution.

A pro of this method is the rapid narrowing of the range to one relatively close to the desired solution while also impeding regression by reducing the mutation chance.

The cons of the function is that it does not always allow reaching the true global minimum as very specific parameters have to implemented for each function in order to do so, since in some cases the mutation is stunted before the optimum is reached.

Wheel of Fortune[**3**]

```
1.    Evaluate P                for i:=0 to POP_SIZE
2.                                  eval[i] = f( P(i) )
3.    Total fitness             for i:=0 to POP_SIZE
4.                                  T += eval[i]
5.    Individual sel.prob.      for i:=0 to POP_SIZE
6.                                  p[i] = eval[i] / T
7.    Accumulated sel.prob.                     q[0] = 0
8.                              for i:=0 to POP_SIZE
9.                                  q[i+1] = q[i] + p[i]
10.    Selection                for i:=0 to POP_SIZE
11.                                   uniform random r in (0,1]
12.                             select for the next generation the individual j
13.                                   for which q[j] ¡ r ¡= q[j+1]
```

2.3 Cross-Over

The Cross-Over function is responsible for diversifying the population by exchanging portions of bit strings between individuals, on in the Genetic Algorithm lingo - exchanging portions of chromosomes, by selecting a random position in the bit string (locus) and a random other individual of the population. Starting the selected locus we will swap each bit (gene) till the very end of the chromosome.

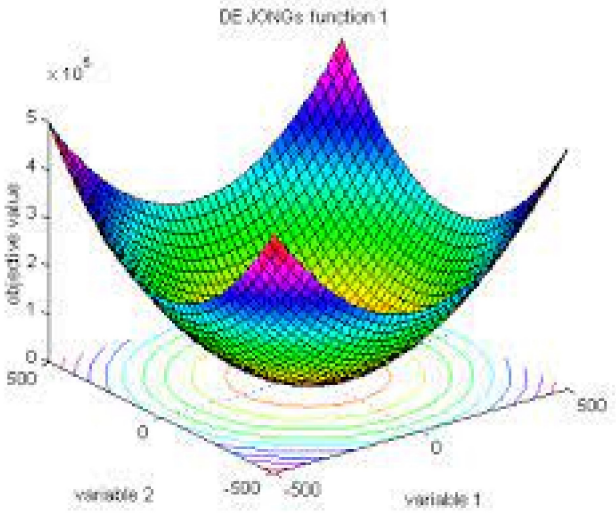
This function yields new individuals which may be a potential new best individual of the population. The chance of Cross-Over is 40

2.4 Succession

Once modified, the auxiliary population will pass it's data through selection to a new population, therefore making a second generation, on which later on we will repeat the succession of steps in order to obtain a new population.

3 Optimization Functions

3.1 DeJong's Function

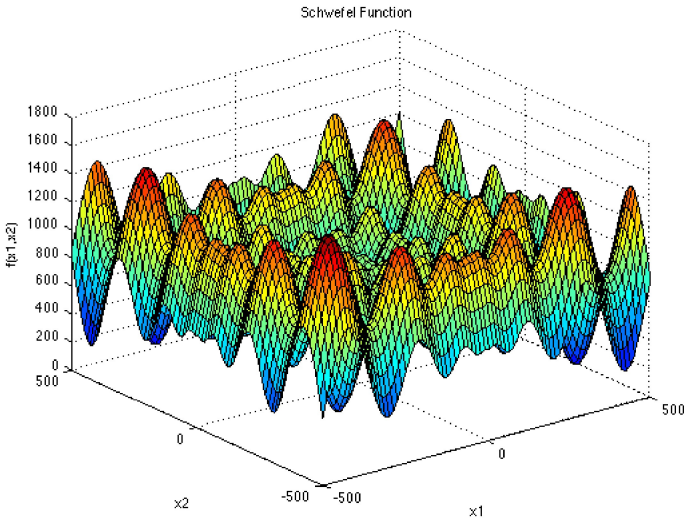


[2]

$$f(x) = \sum_{i=1}^{n} {x_i}^2,\ x_i \in [-5.12; 5.12]$$

Global Minimum : $f(x) = 0,\ x_i = 0, \forall i \in [1; n]$

3.2 Schwefel's Function

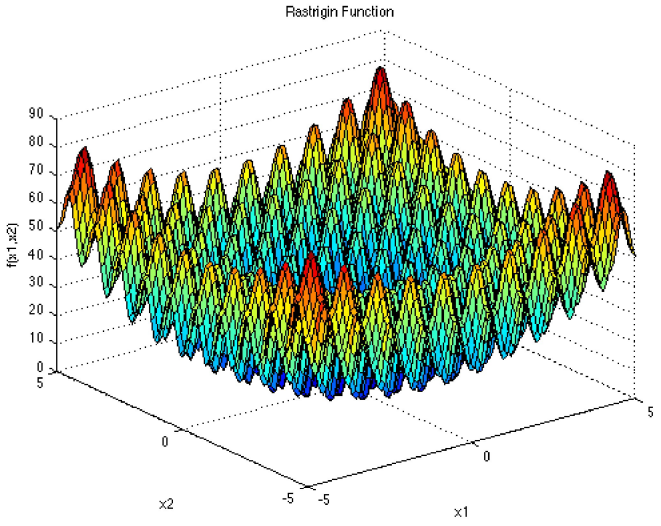


[2]

$$f(x) = \sum_{i=1}^{n} -x_i \cdot sin\left(\sqrt{|x_i|}\right), x_i \in [-500; 500]$$

Global Minimum : $f(x) = -n \cdot 418.9829, x(i) = 420.9687, \forall i \in [1; n]$
$n = 5 : f(x) = -2094.91450$
$n = 10 : f(x) = -4189.82900$
$n = 30 : f(x) = -12569.48700$

3.3 Rastrigin's Function

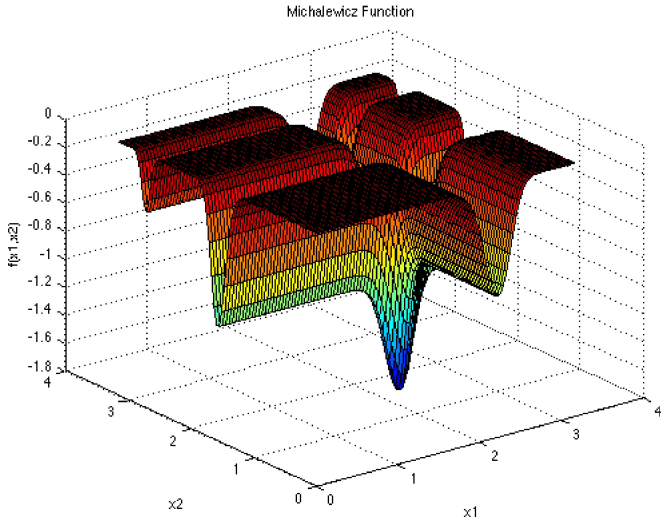


[**2**]

$$f(x) = 10 \cdot n + \sum_{i=1}^{n} \left({x_i}^2 - 10 \cdot cos(2 \cdot \pi \cdot x_i) \right), x_i \in [-5.12; 5.12]$$

Global Minimum : $f(x) = 0, \; x_i = 0, \forall i \in [1; n]$

3.4 Michalewicz's Function



[**2**]

$$f_{12}(x) = -\sum_{i=1}^{n} sin(x_i) \cdot \left(sin\left(\frac{i \cdot {x_i}^2}{\pi}\right)\right)^{2 \cdot m}, x_i \in [0; \pi]$$

Global Minimum :

$n = 5 : f(x) = -4.68765$

$n = 10 : f(x) = -9.66015$

$n = 30 : f(x) = -29.63088$

4 Results

4.1 30 Dimensions

30 Dim	Best	Avg	Worst	St. Dev	HCB (Best)	HCB (Avg)	SA (Best)	Sa (Avg)
DeJong	0	0	0	0	0	0	0	0
Schwefel	-12568.8	-12298.1	-10958.7	387.8329348	-11557.23535	-11320.47005	-12568.10547	-12331.40199
Rastrigin	11.6874	12.94817	16.6762	1.250170139	29.83136	27.10301633	14.9031	23.961427
Michalewicz	-28.7824	-28.44834	-25.5895	0.686166086	-27.36428	-26.94174278	-28.68177	-27.89168633

AVG TIME	DeJong	Schwefel	Rastrigin	Michalewicz
HCB	142	539	319.8	717
SA	14.1	14.2	17.8	96.73
GA	60.3	50.93	30.5	43.3

4.2 10 Dimensions

10 Dim	Best	Avg	Worst	St. Dev	HCB (Best)	HCB (Avg)	SA (Best)	Sa (Avg)
DeJong	0	0	0	0	0	0	0	0
Schwefel	-4155.18	-4087.039	-3650.1	142.9996874	-4189.51465	-4070.57417	-4189.60352	-4133.251319
Rastrigin	2.47636	2.757126667	4.75068	0.65198596	2.2308	4.047589	2.2308	6.715594333
Michalewicz	-9.04071	-9.036507333	-8.97769	0.0159734	-9.59114	-9.362743	-9.53828	-9.206285

AVG TIME	DeJong	Schwefel	Rastrigin	Michalewicz
HCB	3	30.13	10.3	23.1
SA	10.1	9.46	10.23	28.3
GA	22.5	19.43	15.6	29.16

4.3 5 Dimensions

5 Dim	Best	Avg	Worst	St. Dev	HCB (Best)	HCB (Avg)	SA (Best)	Sa (Avg)
DeJong	0	0	0	0	0	0	0	0
Schwefel	-2094.71	-2081.014	-2060.47	17.06086338	-2094.81	-2094.79	-2094.8103	-2077.508036
Rastrigin	0	0.066500567	1.00004	0.253076891	0	0.464314667	0.99497	3.627692333
Michalewicz	-4.65689	-4.656889333	-4.65687	0.00001	-4.68757	-4.686416667	-4.68766	-4.576512333

AVG TIME	DeJong	Schwefel	Rastrigin	Michalewicz
HCB	2.26	5.8	3.4	6.73
SA	6.26	4.13	7	15
GA	12.03	11.6	13.5	16.9

5 Conclusion

Going off by the data obtained we can say that genetic Algorithms hold much potential as opposed to the general Heuristic Algorithms such as Hill Climbing or even Simulate Annealing as it yields decent results within a reasonable timespan. Genetic Algorithms perform well on functions which require a large amount of computation such as Michalewicz, as it offers good results within a short period of time compared to the other aforementioned algorithms. A big upside of the GA is that they have a very small deviation compared to the other algorithms, and even though they might not offer the best absolute best results due to the constraints of implementation (limited no. of iterations, smaller sizes of the population, high mutation/cross-over rates) it can still deliver fairly good and consistent solutions to the Optimization Functions.

6 References

[1]https://en.wikipedia.org/wiki/Genetic_algorithm
[2]http://www.geatbx.com/docu/fcnindex-01.html - Optimization Functions' Definitions
[3]https://profs.info.uaic.ro/~eugennc/teaching/ga/ - GA Materials and Pseudocode []https://www.youtube.com/watch?v=uQj5UNhCPuo - Graphic Explanation of the Structure of a GA