Finding the function minimum of standard benchmark functions using a Genetic Algorithm and comparing it to Hill Climbing and Simulated Annealing

Pasat Tudor Cosmin

November 23, 2021

1 Abstract

Presented in this report is a genetic algorithm designed to find the function minimum. We are going to explain how it works in finding the minimum of a function and in the end show the results of some experiments performed on four standard benchmark functions, tested on 3,10 and 30 dimensions and compare them to three other iterative algorithms, presented in a previous report. These experiments led to finding almost perfect results, obtained in a good amount of time.

2 Introduction

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. Genetic algorithms are commonly used to generate high-quality solutions to optimisation and search problems by relying on biologically inspired operators such as mutation, crossover and selection.
The process of natural selection is performed by choosing the fittest individuals from a population, which can produce descendants that inherit the characteristics of their parents and replace them in the new generation. There are five steps to follow in a genetic algorithm: Initial population, Fitness function, Selection, Crossover and Mutation.
In the methods section we present the structure of the algorithm, describe all the functions we used, how we modified them to obtain better results and the values of all the parameters we used and how we chose them.
In the "Functions" section you are going to find the definition and all the global

minimums of the test functions, four standard benchmark functions, De Jong's function, Schwefel's function, Rastrigin's function and Michalewicz's function.
Moving forward we presented in the section "Experiments" four tables, one for each function, containing the dimensions and all the results of the tests we performed on the genetic algorithm, and also the solutions we got from Hill Climbing and Simulated Annealing in the last raport. After that we will compare all the methods and in the end we will do a comparison between the genetic algorithm and the other methods.

3 Method

We represented the entire population in a vector of vectors of bits, which we first generated at random. For the main function of the algorithm we used the common structure of a genetic algorithm. The algorithm stops after goes through a number of generation, which it gets as input. The other functions of the algorithm are implemented as follows.
For selection, we used Holland's roulette wheel method, where each individual gets a number of descendants that is proportional with their fitness divided by the fitness of the entire population. To improve the algorithm we added elitism for the selection process. This way we chose the best individuals from the population and guaranteed them a place in the next generation. We used the fitness function to measure de quality of the chromosomes and to make sure that the best individuals have a higher chance of getting selected we also used a selection pressure for calculating the fitness of each individual. On the next line is presented the fitness function, where g is the function tested, minimum and maximum are the global extrema of this function and the exponent is the selection pressure.

$$f(x) = \left(\frac{maximum - g(x)}{maximum - minimum + 0.000001} + 0.01\right)^{selectionPressure}$$

Then we have the mutation and the crossover functions. For the mutation process we choose a random number from $[0, 1)$ for every bit of each individual and if it's smaller than 0.01 than we mutate it. In the crossover function we assigned each individual a random number from $[0, 1)$ and then we ordered the individuals by these numbers. Then we pair the individuals 2 by 2, but only the ones that have an assigned number smaller than 0.6. If there is an odd number of individuals like this we have a 50-50 chance of either forgeting it or choosing the next individual. For the actual crossover process we use only one cutting point that we choose at random. Also the descendants obtain from this process are replacing their parents in the curent population.

4 Functions tested

To test the methods presented above we used four standard benchmark functions, defined as seen below.

4.1 De Jong's function

$$f(x) = \sum_{i=1}^{n} x_i^2, i = 1 : n, x_i \in [-5.12, 5.15]$$

Global minimum: f(x) = 0, x(i) = 0, i = 1:n

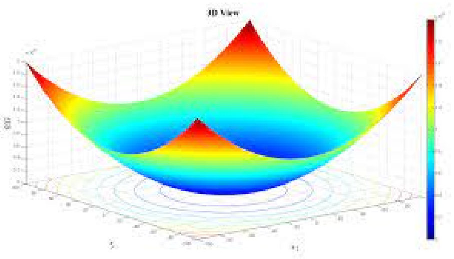

Figure 1: De Jong's function: `https://www.al-roomi.org/component/tags/tag/1st-de-jong-s-function`

4.2 Schwefel's function

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

Global minimum: n=5, f(x) = -2094.91, x(i) = 420.9687, i = 1:n
n=10, f(x) = -4189.82, x(i) = 420.9687, i = 1:n
n=30, f(x) = -12569.48, x(i) = 420.9687, i = 1:n

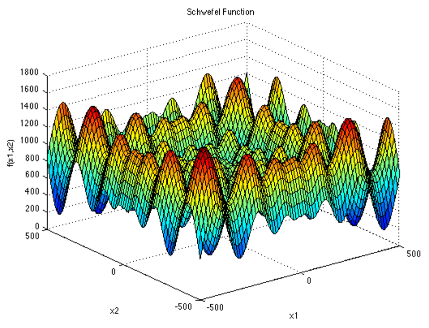

Figure 2: Schwefel's function: `https://www.sfu.ca/~ssurjano/schwef.html`

4.3 Rastrigin's function

$$f(x) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot cos(2\pi x_i)\right], i = 1:n, A = 10, x_i \in [-5.12, 5.15]$$

Global minimum: f(x) = 0, x(i) = 0, i = 1:n

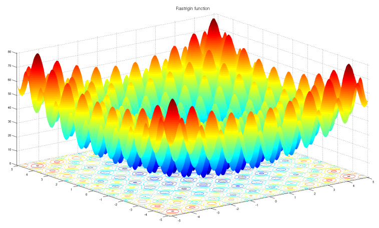

Figure 3: Rastrigin's Function: `https://en.wikipedia.org/wiki/Rastrigin_function`

4.4 Michalewicz's function

$$f(x) = -\sum_{i=1}^{n} sin(x_i) \cdot \left(sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{2 \cdot m}, i = 1:n, m = 10, x_i \in [0, \pi]$$

Global minimum: n=5, f(x) = -4.687, x(i) = ???, i = 1:n
n=10, f(x) = -9.66, x(i) = ???, i = 1:n
n=30, f(x) = -28.98, x(i) = ???, i = 1:n

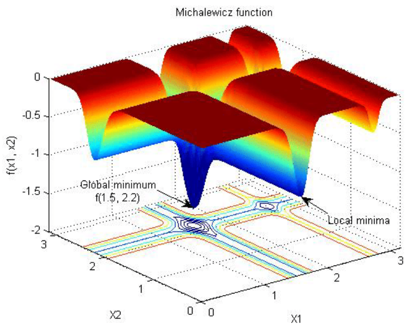

Figure 4: Michalewicz's function: `https://www.researchgate.net/figure/12-A-2D-test-function-known-as-the-Michalewicz-function-Michalewicz-1998_fig6_277197330`

5 Experiment

The experiments were performed on an Intel i5 processor.
We tested the algorithm using these parameters:

Sample size	30
Dimensions	5,10,30
Precision	10^{-5}
Population size	200
Number of generations	5000
Elitism percentage	7%
Selection pressure	4
Mutation probability	1%
Crossover probability	60%

5.1 De Jong's function

	GA				FIHC		BIHC		SA	
D	Minimum	Average	Time	SD	Average	Time	Average	Time	Average	Time
5	0	0	10s	0	0	53s	0	54s	0.00006	4s
10	0	0	16s	0	0	411s	0	319s	0.00005	6s
30	0	0	41s	0	0	724s	0	6651s	0.00026	13s

5.2 Schwefel's function

	GA				FIHC		BIHC		SA	
D	Minimum	Average	Time	SD	Average	Time	Average	Time	Average	Time
5	-2094.91	-2094.9	12s	0.03	-2094.29	182s	-2094.36	146s	-2094.38	4s
10	-4189.72	-4189.52	21s	0.12	-4103.54	1137s	-4098.97	978s	-4141.62	6s
30	-12568.4	-12563.3	54s	12	-11908.5	1189s	-12063	2026s	-11323.7	14s

5.3 Rastrigin's function

	GA				FIHC		BIHC		SA	
D	Minimum	Average	Time	SD	Average	Time	Average	Time	Average	Time
5	0	0	10s	0	0.0002	83s	0.0412	65s	2.6679	3s
10	0	0.281281	16s	0.59	3.9351	521s	4.09	417s	9.5568	5s
30	3.49631	10.8633	42s	4.33	35.2302	1283s	29.2769	8327s	33.971	11s

5.4 Michalewicz's function

	GA				FIHC		BIHC		SA	
D	Minimum	Average	Time	SD	Average	Time	Average	Time	Average	Time
5	-4.68766	-4.66205	10s	0.05	-3.53	70s	-3.68	49s	-3.59781	4s
10	-9.65524	-9.53138	17s	0.10	-8.47	463s	-8.94	350s	-8.04701	6s
30	-28.7051	-28.0803	42s	0.37	-23.511	9081s	-25.985	6829s	-26.0086	12s

5.5 Discussion

We tested different values for each parameter, in order for the algorithm to give more precise results. For the elitism percentage we tried different values between 5% and 10% and concluded until 7% the results got better and better, but after that they started being worse. Using elitism changed our average solution for Rastrigin's function from 67 to under 30. For the selection pressure, we tested values between 1 and 6 and in the end we found 4 to be the best one. We started with a population of 100 individuals and 1000 generations and we increased each one of them until we got to a population of 200 and 5000 generations where we stopped. From this point the difference between the results wasn't big enough to justify the increase in run time. We tested this first on Rastrigin's function on 30 dimensions, where we first got a average global minimum of 26.85 on 30 runs and after modifying these 2 parameters, we got to an average global minimum of 10.86. For the mutation probability we used 1% and for crossover 60%. These values were recommended by our teacher in class and we couldn't find better values than this.

5.6 Comparison

Now we start comparing the results displayed above. Each table contains the statistics for the four benchmark functions , De Jong's function, Schwefel's function, Rastrigin's function and Michalewicz's function. On the first column we find the dimension tested, on the second the Genetic Algorithm with the minimum solution, the average, the standard deviation and the average time out of all the runs, on the third column we have the First Improvement Hill Climb, than Best Improvement Hill Climb and on the last the Simulated Annealing, which each have their average solution and time.
Starting with the 5-dimensional version of the functions, we can see that GA found the function minimum almost every time, the standard deviation being equal or very close to 0, while HC and SA gave decent solutions, but not as perfect. SA couldn't find the optimum solution for Rastrigin's function, but it also has the smallest running time.
Next we have the 10-dimensional version, the GA found again the minimum or a close value to the function minimum, with only one exception, Rastrigin's function where we got an average of 0.28 when the minimum is 0. This result is still a good one comparing it to the other methods where FIHC got a 3.95.

BIHC has a 4.09 minimum and SA has the worst solution of 9.55. Also the time for GA is only 16 seconds, while the others can get up to 9 minutes,
Moving on the 30-dimensional version we can clearly see the difference between algorithms. The Rastrigin's function is a complex function that gives problems to all the methods. While the BIHC gives an average solution of 29.27 with a run time of over 2 hours and SA an average of 33.97 with a run time of 11 seconds, GA finds an average minimum of 10.86 in only 42 seconds.
Overall, GA gives the best results in a decent amount of time. BIHC gave decent results, but it was very time consuming, getting even to a few hours for one run, while SA would give a result in a few seconds, but it was a little worse. GA has even better results than BIHC, being almost perfect, with one exception, while having a run time close to the one of SA.

6 Conclusions

This is all we have to say about the genetic algorithm, tested on the standard benchmark functions , De Jong's function, Schwefel's function, Rastrigin's function and Michalewicz's function. We presented the structure of the algorithm, how it works, the parameters we chose and all the different modifications that we made to find as good of a solution as possible. From the tests we made, it was obvious that the genetic algorithm it's the best method to use out of all the ones presented, since gives an almost perfect solution for all the functions, which are also always better than the ones of the Hill Climbing and Simulated Annealing.

References

[1] Course Page
Genetic Algorithms: https://profs.info.uaic.ro/~eugennc/teaching/ga

[2] Wikipedia Commons
Genetic Algorithm: https://en.wikipedia.org/wiki/Genetic_algorithm
Elitism: https://ro.wikipedia.org/wiki/Elitism
Selection pressure: https://en.wikipedia.org/wiki/Selection_(genetic_algorithm)

[3] Functions
Function definitions:
De Jong's function: http://www.geatbx.com/docu/fcnindex-01.html#P89_3085
Schwefel's function: http://www.geatbx.com/docu/fcnindex-01.html#P150_6749
Rastrigin's function: http://www.geatbx.com/docu/fcnindex-01.html#P140_6155
Michalewicz's function: http://www.geatbx.com/docu/fcnindex-01.html#P204_10395

[4] Others
Genetic algorithms article: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
Genetic Programming Theory and Practice VI: https://books.google.ro/books?id=AnkveR_LlUMC&pg=PA101&lpg=PA101&dq=elitism+population+percentage&source=bl&ots=XkgtpxwAxL&sig=ACfU3U3a8cinP3rPDRWBlsA4od8RphPZcA&hl=ro&sa=X&ved=2ahUKEwib5ezZgqn0AhWagv0HHZtYDUQQ6AF6BAg1EAM#v=onepage&q=elitism%20population%20percentage&f=false