T2 - Numerical optimization using Genetic Algorithms

Oloieri Alexandru

November 25, 2019

Abstract

This paper analyzes the behaviour and performance of Genetic Algorithms when dealing with a complex problem: finding the global minima/maxima of a mathematical benchmark function, and compares the obtained results with the results of a well known trajectory-based method: Hill Climbing Best Improvement.

1 Introduction

In computer science, there are some problems which require a lot of resources (time, memory) in order to be solved (usually these are called hard problems), and, unfortunately, classical algorithms are too weak for solving them. One way to solve this kind of problems is using metaheuristics, because even if they don't produce an optimal result on each run, the results they produce are usable and most of the times their running time is low compared to other algorithms that would find the optimal result, but realistically speaking, they would never end for large instances of the problem.

A well known hard problem is finding the global minimum/maximum of a mathematical benchmark function which is defined on an interval, because a computer cannot run an algorithm that evaluates the function in each point from the function domain, and then chooses the best one (because there are an infinity of points). This paper analyzes how good are Genetic Algorithms behaving in practice, when dealing with this problem, more exactly, when trying to find the global minimum of some functions with more than one dimensions (functions which have an argument of type $x = (x_1, x_2, ..., x_n)$).

2 Genetic Algorithm Description

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. [1] Genetic algorithms are a population-based approach, as they maintain and improve multiple candidate solutions, often using population characteristics to guide the search and relying on bio-inspired operators such as mutation, crossover and selection. [1] [2] In this experiment the size of the populations at each iteration will be equal to $populationSize = 100$.

2.1 Representation of the solutions

The solutions will be represented as binary arrays. Let's say we want to represent every real number with d digits for precision. Then we will divide the function domain $[leftMargin, rightMargin]$ in $Cnt = (rightMargin - leftMargin) * 10^d$ equal intervals. For a dimension, to represent Cnt different values we need $len = \lceil log2(Cnt) \rceil$ bits, so for N dimensions, for each chromosome, we will need $len \cdot N$ bits. In this experiment the precision will be $d = 5$.

2.2 Decodification of the solutions

We will have to decode the solutions to be able to evaluate them. If we represented real numbers in the manner described at 2.1, then for a function with N dimensions, the following domain: $[leftMargin, rightMargin]$, and $binary_i$ being the binary representation of the i^{th} dimension, the real value for each dimension will be:

$$realValue_i = leftMargin + decimal(binary_i)/(2^{len} - 1) \cdot (rightMargin - leftMargin)$$

2.3 Crossover

Crossover (also called recombination) is a genetic operator used to combine the genetic information of two parents to generate new offspring. [4] During an iteration each chromosome will have a crossover probability p_i, and those who have $p_i < P_c$, where P_c is the crossover probability constant, will take part in crossover.

2.4 Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.[3] This will be implemented in the following way: each bit from the cromosome will have its value changed with a probability equal to P_m.

2.5 Selection

During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected.[1] Since all the benchmark functions used in the experiemnt evaluate only to positive values, the fitness function that will be used is:

$$fitness_i = 1/(f(chromosome_i) + 0.01)$$

2.6 Elitism

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as elitist selection and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next. [1] This will be achieved by copying the best k_e candidates from the current generation to the next one (during the selection stage).

2.7 Final Algorithm

Algorithm 1 Genetic Algorithm Scheme

```
1: $population \leftarrow randPopulation()$;
2: for $i \leftarrow 2; i \leq generationsCount; ++i$ do
3:     $mutation()$;
4:     $crossOver()$;
5:     $population \leftarrow selection(population)$;
6: $return\ best(population)$;
```

3 Functions

Rastrigin's function:

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

Schwefel's function:

$$f(x) = 418.9829 \cdot n - \sum_{i=1}^{n} \left[x_i \cdot sin(\sqrt{|x_i|})\right], x_i \in [-500.0, 500.0]$$

Griewank's function:

$$f(x) = 1 + \sum_{i=1}^{n} \left[x_i^2/4000\right] - \prod_{i=1}^{n} \left[cos(x_i/\sqrt{i})\right], x_i \in [-600.0, 600.0]$$

Zakharov's function:

$$f(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5 \cdot i \cdot x_i)^2 + (\sum_{i=1}^{n} 0.5 \cdot i \cdot x_i)^4, x_i \in [-5.0, 10.0]$$

For the experiment to have varied and relevant results, appropriate functions have to be chosen (with different types of plots and different properties). For that, in this experiment three non-convex and multimodal functions, with a lot of local minimas (Rastrigin's function, Schwefel's functions and Griewank's function) and one unimodal and convex function, with no local minimum except the global one (Zakharov's function) will we used.

4 Experiment

4.1 Suggested parameters analysis

The suggested parameters for running the genetic algorithms are:

- $populationSize = 100$
- $generationsCount = 1000$
- $crossOverProbability(P_c) = 0.3$
- $mutationProbability(P_m) = 0.01$

dim	runs	generationsCount	best	worst	mean	stDev	time(s)
5	15	1000	6.18	20.05	14.89	3.56	4.8

Figure 1: Rastrigin's function results with suggested parameters

The results for the Rastrigin's function with 5 dimensions are pretty bad, as even the Naive Hill Climbing provides better results (even though that strategy gets stuck in a local minima). In my opinion this is due to the values provided by the fitness function, because they are very close one to each other, and the best chromosomes don't have significant better chance to be selected (the selection pressure is low, and this is a disadvantage for fit candidates). This together with the high rate of mutation makes the algorithm to sometimes "lose" the best candidades. But the benefit of metaheuristics is that we can experiment with several values for parameters, and then choose the ones that fit the best our needs, so an idea is to modify some of the parameters in trying to get the best possible results.

4.1.1 Generations Count

After increasing the generations count to 7500, the running time increased, but the results were better, because even if the mutation affected the best chromosomes, there were more iterations when the GA could improve the results.

dim	runs	generationsCount	best	worst	mean	stDev	time(s)
5	15	1000	5.05	12.03	9.19	2.23	73.664

Figure 2: Rastrigin's function results with increased generations count

dim	runs	generationsCount	best	worst	mean	stDev	time(s)
5	15	1000	0.00	4.46	1.06	1.59	44.13

Figure 3: Rastrigin's function results with some optimizations

4.1.2 Mutation probability and Elitism

To further optimize the algorithm, the mutationProbability was lowered to $P_m = 0.001$, and also the strategy described at **2.6** (Elitism, $k_e = 6$) was integrated in the program, and this led to very close to optimal results.

4.2 Parameters used in the experiment

As the goal of using Genetic Algorithms is to get best possible results, some of the suggested parameters will be modified to increase the selection pressure and to ensure that, for most of the times, the quality of the best candidate will not decrease between generations.

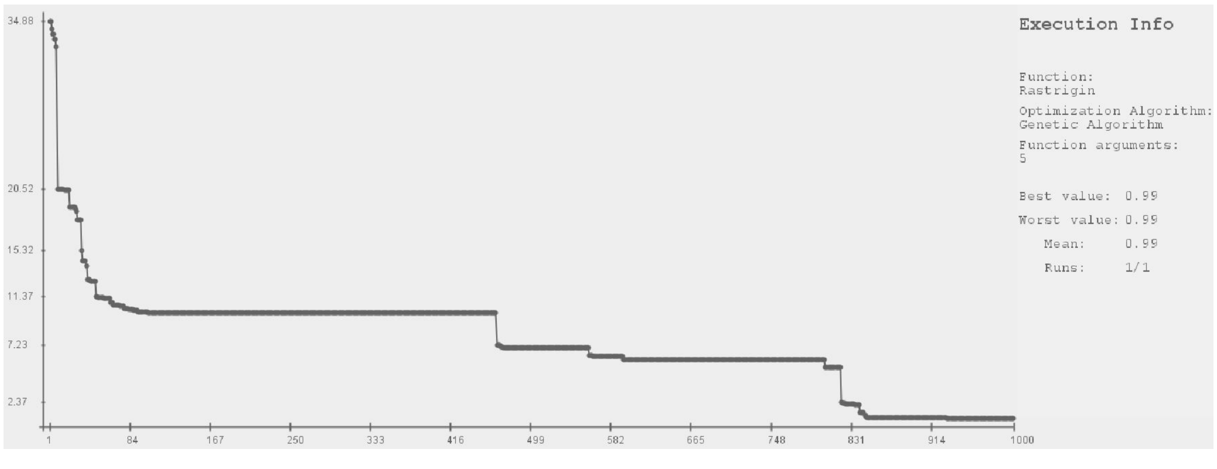
- $populationSize = 100$
- $generationsCount = 10.000$ for 10 and 30 dimensions, $generationsCount = 7.500$ for 5 dimensions
- $P_m = 0.001$
- $crossOverProbability(P_c) = 0.3$
- $k_e = 6$ (best 6 chromosomes will be copied from the current generation to the next one)

For each function and for each number of dimensions, the Genetic Algorithm described at **Section 2** will be run 30 times using the above parameters.

5 Results

dim	generationsCount	best	worst	mean	stDev	time(s)
5	7500	0.00	5.62	1.53	1.68	76.094
10	10000	0.00	8.63	3.86	2.84	197.45
30	10000	11.16	42.19	23.6	7.41	569.203

Figure 4: Rastrigin's function results

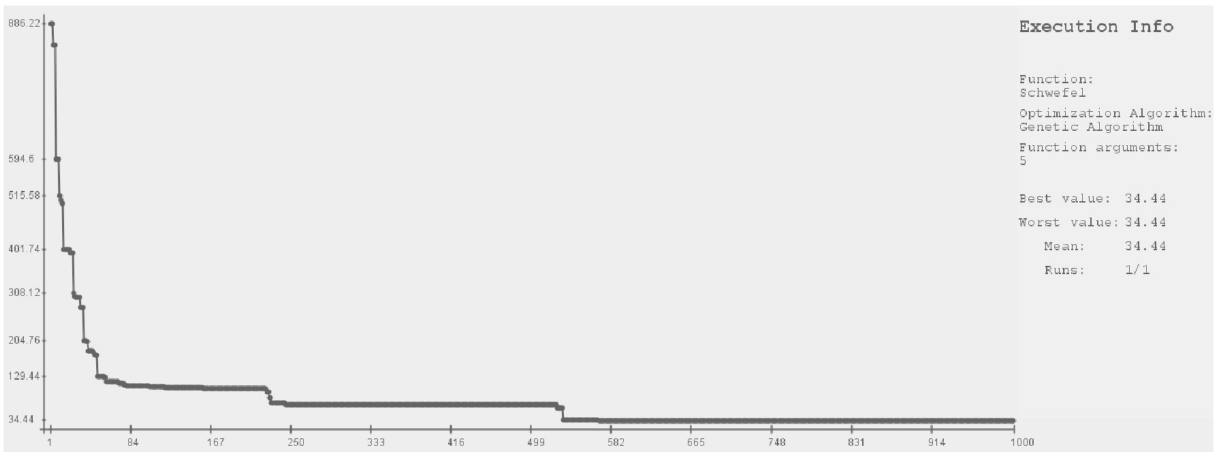


```
Execution Info

Function:
Rastrigin
Optimization Algorithm:
Genetic Algorithm
Function arguments:
5

Best value:  0.99
Worst value: 0.99
    Mean:    0.99
    Runs:    1/1
```

Figure 5: Rastrigin's function 5 dimensions, 1000 generations

dim	generationsCount	best	worst	mean	stDev	time(s)
5	7500	0.103	0.51	0.27	0.12	93.81
10	10000	0.106	34.73	2.76	8.53	248.14
30	10000	36.0	727.45	246.18	144.03	723.13

Figure 6: Schwefel's function results



Execution Info

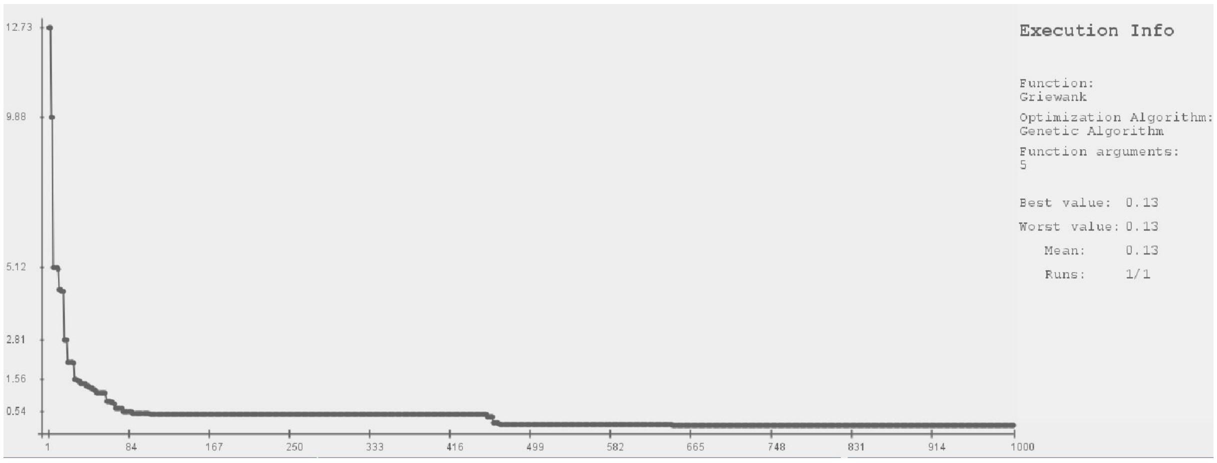
Function:
Schwefel
Optimization Algorithm:
Genetic Algorithm
Function arguments:
5

Best value: 34.44
Worst value: 34.44
Mean: 34.44
Runs: 1/1

Figure 7: Schwefel's function 5 dimensions, 1000 generations

dim	generationsCount	best	worst	mean	stDev	time(s)
5	7500	0.007	0.16	0.04	0.034	80.95
10	10000	0.00	0.19	0.07	0.04	212.63
30	10000	0.10	0.25	0.08	0.05	622.93

Figure 8: Griewank's function results



Execution Info

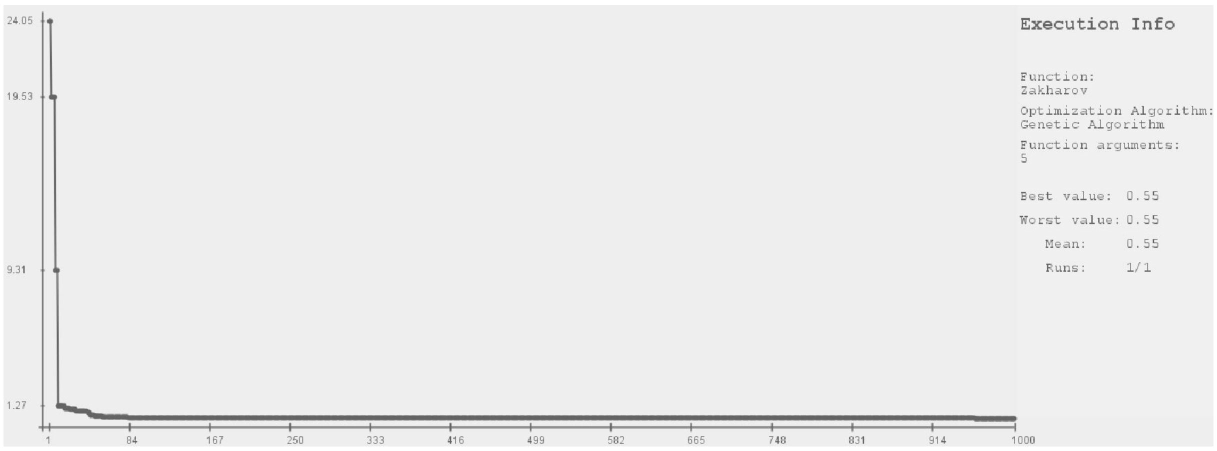
Function:
Griewank
Optimization Algorithm:
Genetic Algorithm
Function arguments:
5

Best value: 0.13
Worst value: 0.13
Mean: 0.13
Runs: 1/1

Figure 9: Griewank's function 5 dimensions, 1000 generations

dim	generationsCount	best	worst	mean	stDev	time(s)
5	7500	0.03	11.79	3.86	3.67	63.91
10	10000	1.13	26.43	12.59	6.39	155.074
30	10000	38.56	71.15	57.12	8.84	464.87

Figure 10: Zakharov's function results



Execution Info

Function:
Zakharov

Optimization Algorithm:
Genetic Algorithm

Function arguments:
5

Best value: 0.55
Worst value: 0.55
Mean: 0.55
Runs: 1/1

Figure 11: Zakharov's function 5 dimensions, 1000 generations

5.1 Interpretation

5.1.1 Rastrigin's, Schwefel's and Griewank's functions

The obtained results for these functions are very close to the global optimum, and this is mainly due to the optimizations added to the standard Genetic Algorithm (Elitism and a smaller value for mutation probability), which guaranteed that the best solution quality will not decrease from one generation to another.

An interesting thing to observe is that for Griewank's function, the results are very close to global minimum for all the dimensions (whereas for most functions the global minima increases as we increase the number of dimensions). That's because the way we chose the fitness function ($1/f(x)$) really benefits the functions which have large values in function codomain, and the function domain is $[-600.0, 600.0]$, so $f(x)$ will be pretty large, so during the selection phase best candidates will have a very high chance to be selected. This together with the fact that elitism was integrated in the algorithm almost guarantees that we will get good results (results close to the optimum).

As *generationsCount* is fixed (7.500 for 5 dimensions and 10.000 for 10 and 30 dimensions), the running time of the algorithms are very similar, the small differences are only caused by the complexity of the operations needed for evaluating a function. (for example, computing the square root of some real numbers, the sine of a value etc.)

5.1.2 Zakharov's function

The way the GA was implemented doesn't guarantee that for convex functions it will find the global minima on each run (unlike the Hill Climbing or Simulated Annealing strategies). This is because the mutations may modify some bits of the best candidates, so they cannot converge to the global minima. Since the values of the function evaluations are small and close to each other (so there is a small selection pressure), during selection we may miss choosing the best chromosomes. The running times are a little bit smaller than the running time of the other functions because this function only contains basic mathematical operations: addition and multiplication (so we need less computational effort). Something very interesting is that the best candidate improved a lot during the first 50/60 iterations, and then there was barely any improvement, and I think that's mainly because the genetic operators (mutation and crossover) affected the best candidates, so no chromosome could converge to the global minima.

5.1.3 Comparison with Hill Climbing Best Improvement

- Multimodal functions

algorithm	dim	generationsCount	best	worst	mean	stDev	time(s)
Genetic Algorithm	30	10000	11.16	42.19	23.6	7.41	569.203
HC Best Improvement	30	-	26.26	40.83	34.04	3.92	399.78

Figure 12: Rastrigin's function results comparison

algorithm	dim	generationsCount	best	worst	mean	stDev	time(s)
Genetic Algorithm	30	10000	36.0	727.45	246.18	144.03	723.13
HC Best Improvement	30	-	723.32	1817.14	1498.29	216.901	974.66

Figure 13: Schwefel's function results comparison

For non-convex and unimodal functions the Genetic Algorithm found way better results than the Hill Climbing one, because most of the times, due to the greedy strategy it uses, the Hill Climbing algorithm gets stuck in local minimas, whereas the Genetic Algorithm not only maintains more than one solution, but also uses the genetic operators described at **2.3** and **2.4** which will cause a better exploration and exploitation of the function domain. The running time of the Genetic Algorithm is almost constant for a fixed *populationsCount*, in contrast to the running time of the Hill Climbing strategy, which will vary based on the function graph and the quality of the neighbors (the number of times we find a better candidate during iterations may be very large).

- Unimodal functions

algorithm	dim	generationsCount	best	worst	mean	stDev	time(s)
Genetic Algorithm	20	10000	28.36	65.14	39.78	9.28	164.03
HC	20	-	0.00	0.00	0.0	0.0	52.58

Figure 14: Zakharov's function results comparison

The strategy of the Hill Climbing algorithms is perfect for this kind of functions because it doesn't matter what is the starting point, it will always converge to the global minima. This is not the case for Genetic Algorithms, and this can be observed from the results: because the number of generations was 10000 it had a higher running time, and the results were worse.

6 Conclusions

Even though they are complex, when used with wrong parameters, Genetic Algorithms can have worse results than simpler optimization strategies, as observed in section **4.1**, so with a little bit of analysis suitable/optimal parameters can be chosen, and when this happens, the results are better than the ones provided by other strategies like Hill Climbing or Simulated Annealing, with comparable running times that can also be adjusted when increasing or decreasing the value of *populationsCount*. So this metaheuristic is a good option for the cases when we need both accuracy (get the best possible results) and efficiency (the execution time to be as low as possible), but it must be taken into account that it doesn't benefit specific types of functions (which happens for Hill Climbing algorithms and unimodal functions), so there are cases when the results are worse, and other strategies should be used.

Consequently, Genetic Algorithms are powerfull algorithms, with relative low running time, that when used correctly can generate better results than other metaheuristics.

References

[1] More about GAs in computer science: `https://en.wikipedia.org/wiki/Genetic_algorithm`

[2] More about metaheuristics in computer science: `https://en.wikipedia.org/wiki/Metaheuristic`

[3] More about mutation in GAs: `https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)`

[4] Crossover in GAs: `https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)`

[5] Information about functions used in this experiment: `https://www.sfu.ca/~ssurjano/optimization.html`

[6] More details about the implementation of genetic algorithms: `https://profs.info.uaic.ro/~pmihaela/GA/laborator3.html`

[7] The github repository of the application used in this experiment: `https://github.com/OloieriAlexandru/Numerical-Optimization-Platform/tree/T2`