H2 & H2P

Opris Vlad, 2A3

December 6, 2023

Abstract

This report studies the behaviour of a genetic algorithm designed to solve a optimization problem, this being finding the minimum of a function. The report explains how the the algorithm works and show the results of experiments performed on four benchmark functions (Rastrigin, De Jong 1, Schwefel, Michalewicz) and then compare the results with some previously analyzed heuristic algorithms: Hill Climbing and Simulated Annealing. The result will reveal some insights and conclusions about genetic algorithms in contrast with other methods.

1 Introduction

Genetic algorithms draw inspiration from the principles of natural selection and genetics to iteratively evolve potential solutions and converge towards an optimal solution. This algorithm resembles the process of natural selection, by selecting for reproduction the fittest solutions in order to potentially produce better solutions for the next generation. By mimicking the mechanisms of genetic evolution through biologically inspired operators such as selection, crossover and mutation, a genetic algorithm can navigate vast solution spaces to efficiently uncover the global minimum.

The method section will explain how the algorithm works and how these operators are actually implemented. In the section for experiments, there are present four tables one for each function (Rastrigin, De Jong 1, Schwefel, Michalewicz) and dimensions tested, containing the results of the tests and also the solutions from Hill Climbing and Simulated Annealing. In the end, a comparison between all the methods will be made.

2 Methods

The algorithms follows the structure of a standard genetic algorithm. The first step is to generate a random population of candidate solutions (in our case an array of bitstrings) that will evolve over multiple generations. Roulette wheel method is used for the selection step. This method follows the idea that each individual gets a number of descendants proportional with their fitness. Elitism is also implemented along the selection to ensure that the best solutions will pass to the next generation. The role of a fitness function is to measure the quality of the chromosomes and make sure the best solutions get a high score, thus having a higher change of getting selected. The fitness function goes as follows

$$f_1(x) = \left(1 + \frac{gmax - g(x)}{gmax - gmin + \epsilon}\right)^p \; or \; f_2(x) = \frac{1}{g(x) + \epsilon}$$

g is the benchmark function; gmax, gmin are the maximum and mimimum values of g gotten from the current population and p represents the selection pressure parameter. The best results were obtained when f_1 was used for Rastrigin, Schwefel, Michalewicz and f_2 for De Jong.

The next components of the algorithm are crossover and mutation. The crossover of the population works as follows: in every generation an individual gets a change to be selected for the crossover operation according to a crossover probability. For the actual crossover, one-point crossover is used. This results in two offspring which are added to the population. The selection process will subsequently take care to restore the size of the population to a base value. Regarding mutation, every bit of each individual mutates/flips according to a mutation probability. An addition to the mutation process would be that every few hundred generation, random individuals would have a mutation probability of 0.5, namely replace them with new random solutions (partial reset). The algorithm ends when it performs a predetermined number of generations.

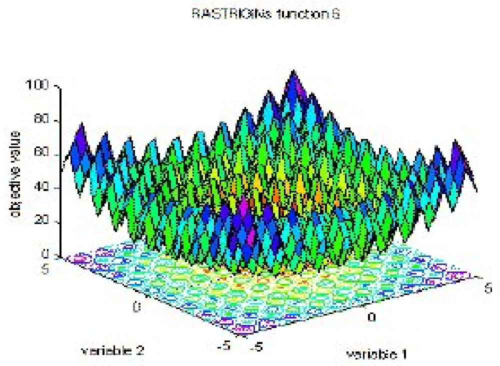
3 Experiment results

3.1 Parameters

Tests	30
Precision	10^{-5}
Population size	200
Generations	2000/5000
Selection pressure	4
Elitism	10%
Crossover prob.	0.6
Mutation prob.	0.01

The GA time measured is per test. For 5, 10 dimensions the number of generations was set to 2000 and for 30 the number was set to 5000.

3.2 Rastrigin's function



$$f(x) = 10d + \sum_{i=1}^{d}(x_i^2 - 10cos(2\pi x_i)), x_i \in [-5.12, -5.12], f(x) \geq 0$$

	GA				HCBI		HCFI		SA	
Dim	Min	Mean	Sd	Time	Mean	Time	Mean	Time	Mean	Time
5	0.00000	0.04119	0.22	3s	0.00000	2m	0.84519	4.5m	1.39972	50s
10	0.00000	0.74150	0.81	7s	2.40042	7.8m	7.72923	10.1m	2.65548	68s
30	4.94342	16.9252	7.59	49s	28.06330	6.6m	52.55050	21.4m	20.78260	2.3m

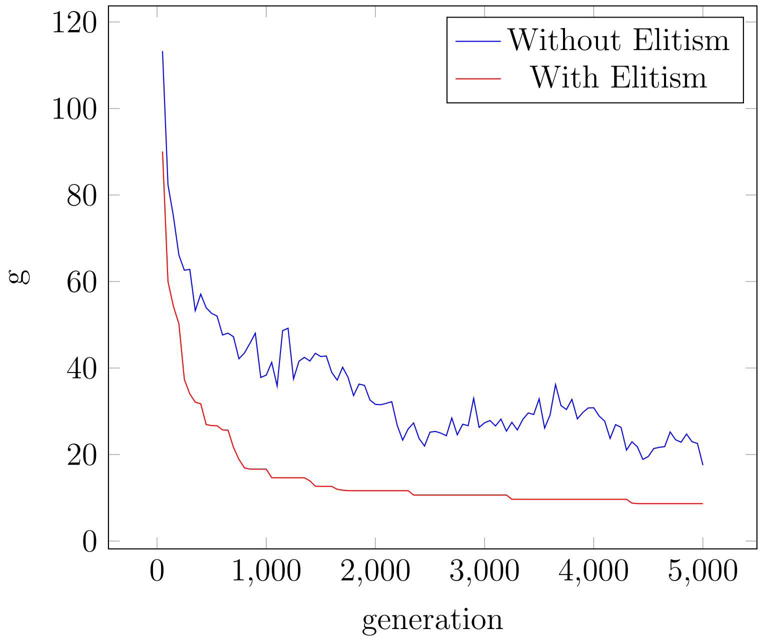
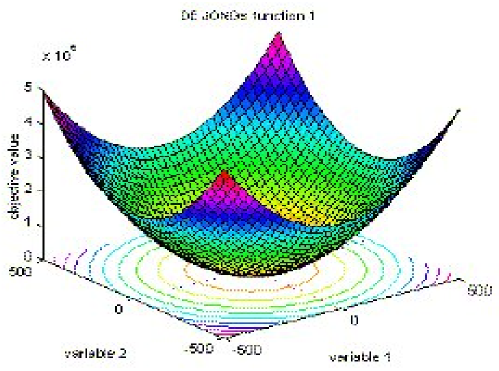


Figure 1: Rastrigin 30 Dim.

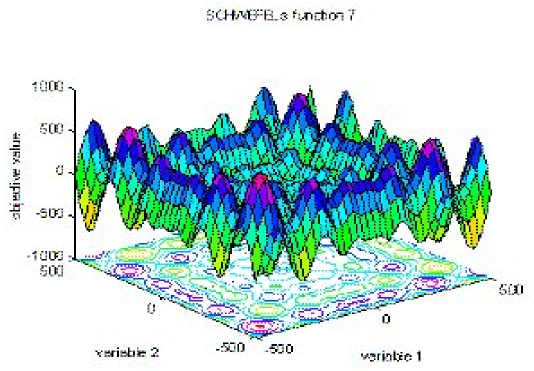
3.3 De Jong's 1 function



$$f(x) = \sum_{i=1}^{d} x_i^2,\ x_i \in [-5.12, -5.12],\ f(x) \geq 0$$

	GA				HCBI		HCFI		SA	
Dim	Min	Mean	Sd	Time	Mean	Time	Mean	Time	Mean	Time
5	0.00000	0.00000	0.00	3s	0.00000	30s	0.00000	16s	0.00000	25s
10	0.00000	0.00000	0.00	6s	0.00000	2m	0.00000	1m	0.00000	29s
30	0.00000	0.00000	0.00	11s	0.00000	1.7m	0.00000	54s	0.00000	44s

3.4 Schwefel's function



$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i sin(\sqrt{|x_i|}), x_i \in [-500, -500], f(x) \geq 0$$

	GA				HCBI		HCFI		SA	
Dim	Min	Mean	Sd	Time	Mean	Time	Mean	Time	Mean	Time
5	0.00000	0.06991	0.08	4s	0.00812	8.2m	20.0634	8.5m	1.27330	57s
10	0.20996	0.04887	0.14	9s	27.0088	15.3m	153.6540	18.3m	6.02869	1.3m
30	0.94238	3.87168	8.59	66s	1232.48	11.8m	1455.04	29.7m	210.689	2.6m

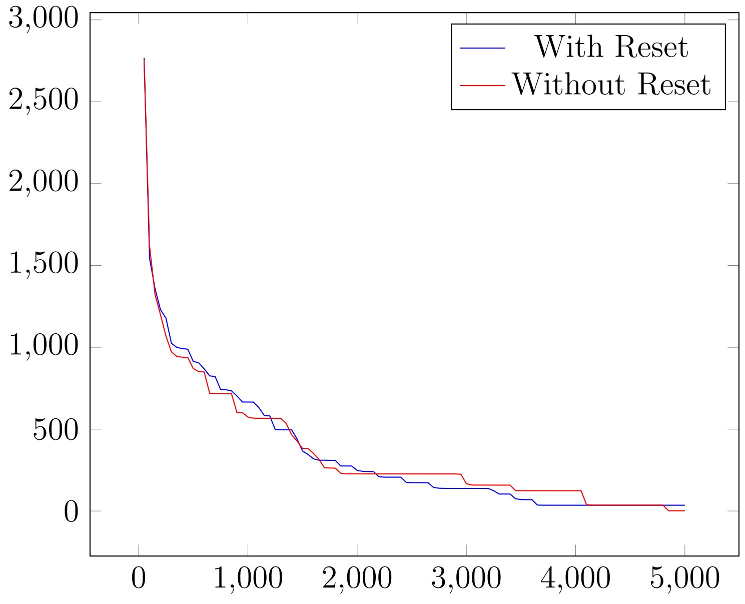
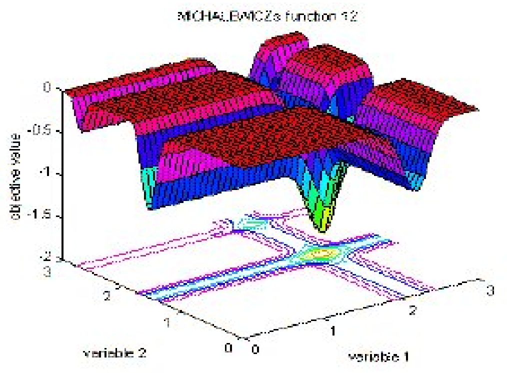


Figure 2: Schwefel 30 Dim.

3.5 Michalewicz's function



$$f_d(x) = -\sum_{i=1}^{d} sin(x_i) sin^{20}\left(\frac{ix_i^2}{\pi}\right), x_i \in [0, \pi], f_5(x) \geq -4.68765, f_{10}(x) \geq -9.66015$$

	GA				HCBI		HCFI		SA	
Dim	Min	Mean	Sd	Time	Mean	Time	Mean	Time	Mean	Time
5	-4.68765	-4.68140	0.02	4s	-4.68765	4m	-4.68617	5m	-4.63777	1.6m
10	-9.65782	-9.52960	0.08	19s	-9.53463	15m	-9.19747	14m	-9.37160	2.5m
30	-28.37330	-27.79300	0.04	56s	-27.00870	12m	-24.4564	24.8m	-28.17060	5.6m

4 Comparison

GA uses a population of potential solutions and evolves them over generations through selection, crossover, and mutation. HC Iteratively moves towards the direction of increasing elevation, making small steps to find the peak of the mountain. Simulated Annealing randomly explores the solution space, allowing for uphill/downhill moves with a certain probability, gradually decreasing the probability of accepting moves.

Immeadiately after implementing the GA and then testing it, it could be observed that they were a lot of instances where the best solution from current population was worse than the value belonging to the previous generation. HC does not show this behaviour because it always keeps the best value. SA may create this instance, but this behaviour is more and more inhibited over time.

In the early stage of the algorithms HC, SA start with better solution because they compare the solutions after every iteration and eventually reach a intermediate point or either a local optima. GA starts worse but the optimizing is a steady process and from the tests the case of getting stuck in a local optima is not that common, consequently from crossover and mutation.

GA emphasizes exploration through the maintenance and evolution of a diverse population. HC primarily focuses on exploitation by moving towards the current best solution. SA balances exploration and exploitation by allowing occasional uphill moves, preventing the algorithm from getting stuck in local optima.

5 Conclusion

Genetic algorithms, Hill climbing, and Simulated Annealing are all optimization algorithms, but they differ in their approaches and characteristics. From the performed experiments, it can be concluded that the genetic algorithm both in terms of quality of the results and time; it's the most suitable method out of the three to solve this problem. The choice between these algorithms depends on the nature of the optimization problem and the trade-off between exploration and exploitation.

6 Adaptive Genetic Algorithm

In a regular GA the parameters/operators do not change between the generations. If the algorithm changes it's behaviour over a run and adapts its parameters in beneficial way, this can have a big impact on performance. Therefore, in an Adaptive Genetic Algorithm parameters are not constant, but change in correspondence with the current state of the population or the generation.

Adaptive Mutation

The weak point of regular GA is the total randomness of mutation, which is applied equally to all chromosomes, irrespective of their fitness. Thus a very good chromosome is equally likely to be disrupted by mutation as a bad one. On the other hand, bad chromosomes are less likely to produce good ones through crossover. They would benefit the most from mutation.

The use of adaptive mutation to solve this problem turns out to be a good approach. The adaptive mutation is implemented as follows. Firstly the average fitness is calculated in the evaluation step and based on this a different mutation probability is applied. Namely a solution with the fitness less than average will have a a higher mutation rate because this would likely increase its quality. In contrast on a better solution(fitness greater than average) will be applied a lower mutation rate to avoid disruptions.

Gray encoding

Gray code can be used to encode individuals in the population. This encoding has certain advantages over the binary encoding. In Gray code, consecutive numbers have bitwise representations that differ by only one bit. This property can be advantageous in GAs because small changes in the genotype are more likely to result in small changes in the phenotype. Furthermore, gray code tends to have a smaller average Hamming distance between consecutive values compared to traditional binary encoding. This can be beneficial as it may reduce the probability of disruptive changes during crossover operations.

6.1 Parameters and Results

Tests	30
Precision	10^{-5}
Population size	200
Generations	1000/2000
Selection pressure	4
Elitism	10%
Crossover prob.	0.6
Mutation prob.	0.005/0.01

The AGA time measured is per test. For 5 dimentions the number of generations was to 1000 for 10, 30 the number was set to 2000.

6.2 Rastrigin's function

	AGA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.00000	0.00	1s
10	0.00000	0.00000	0.00	3s
30	0.00000	0.71123	0.93	20s

	GA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.04119	0.22	3s
10	0.00000	0.74150	0.81	7s
30	4.94342	16.9252	7.59	49s

6.3 De Jong's 1 function

	AGA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.00000	0.00	1s
10	0.00000	0.00000	0.00	3s
30	0.00000	0.00000	0.00	9s

	GA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.00000	0.00	3s
10	0.00000	0.00000	0.00	6s
30	0.00000	0.00000	0.00	11s

6.4 Schwefel's function

	AGA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.00345	0.00	2s
10	0.00000	0.01077	0.01	9s
30	0.114258	0.81429	0.30	27s

	GA			
Dim	Min	Mean	Sd	Time
5	0.00000	0.06991	0.08	4s
10	0.20996	0.04887	0.14	9s
30	0.94238	3.87168	8.59	66s

6.5 Michalewicz's function

	AGA			
Dim	Min	Mean	Sd	Time
5	-4.68765	-4.68765	0.00	2s
10	-9.65524	-9.47827	0.12	7s
30	-28.93800	-27.926	0.43	22s

	GA			
Dim	Min	Mean	Sd	Time
5	-4.68765	-4.68140	0.02	4s
10	-9.65782	-9.52960	0.08	19s
30	-28.37330	-27.79300	0.04	56s

6.6 GA vs AGA

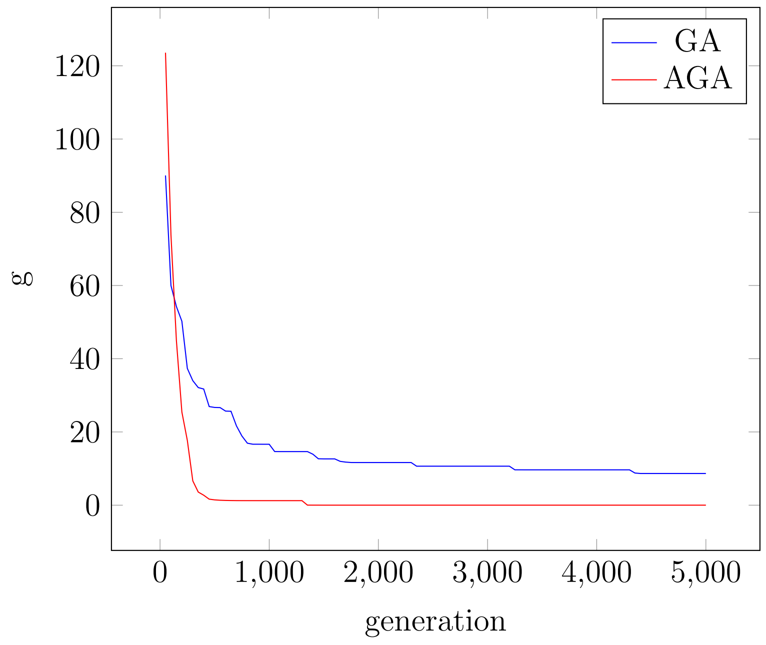


Figure 3: Rastrigin 30 Dim

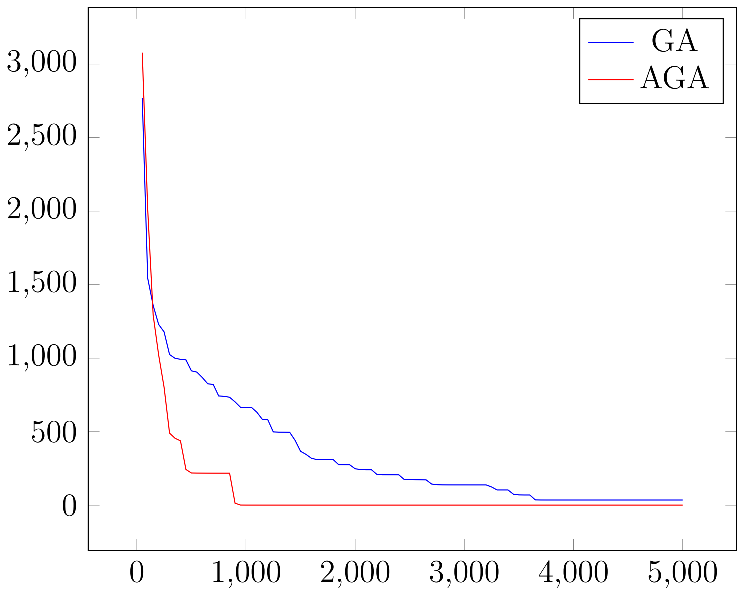


Figure 4: Schwefel 30 Dim

References

[1] Course page
`https://profs.info.uaic.ro/~eugennc/teaching/ga/`

[2] Wikipedia
`https://en.wikipedia.org/wiki/Genetic_algorithm`
`https://en.wikipedia.org/wiki/Gray_code`

[3] Function definitions
http://www.geatbx.com/docu/fcnindex-01.html