Genetic algorithms in minimizing functions

Robert Gaina

November 30th, 2020

Abstract

In optimization problems we are looking for either the largest or the smallest value that a function can take. Genetic algorithms mantain a population of representations of candidate solutions which are improved over generations by mutation and crossover; the selection of the next generation is done based on the quality of the individual. Comparative tests are performed upon different functions on different sizes with a focus on the atomic operators in Genetic Algorithms and the impact of scalability in population, respectively number of generations. The conclusions are based on empirical tests which point out strong and weak points of the method used.

1 Introduction:

Genetic Algorithms are stochastic search algorithms that are based on the principles of natural selection and genetics, simulating "survival of the fittest" concept among individuals. They are commonly used to generate high-quality solutions for optimization problems and search problems.

In this paper we are describing a basic Genetic Algorithm, investigating how it behaves in the process of minimizing a function and what impact brings each operator to the results. More detailed studies were conducted on the influence of the population size and number of generations for functions with an increased size.

2 Methods:

2.1 Algorithm

Since genetic algorithms simulate the process of natural selection the solutions that can adapt to the desired environment are able to survive and reproduce to the next generation. Each generation is a population on individuals (chromosomes) and each individual represents a starting point in the search space of possible solutions. Therefore, the genetic algorithm initiates its search from a population of points, not a single one. Similar to natural genetics, these solutions must face improvements over time so these modifications consists of mutations and crossovers. In order to keep the "survival of the fittest" concept, to each individual we apply a fitness function to rate their quality. This fitness function must be chosen such that the value of this function is higher for the best solutions. This way, the fitter individuals are given a higher chance of selection over generations evolving better solutions until reaching a stopping criterion.

Algorithm 1 Pseudocode of genetic algorithms

```
t=0
generate the starting population P(t)
evaluate P(t)
while not StoppingCondition do
    t=t+1
    select P(t) from P(t-1)
    mutate P(t)
    cross-over P(t)
    evaluate P(t)
end while
```

2.2 Implementation

The implementation of this method follows the guidelines from 2.1 section with the following additions

- The population is a set of fixed-length bitstrings. This type of binary encoding facilitates the cross-over and mutation of atomic information.
- Whenever we evaluate a solution we are converting it to their decimal value.
- The probability of mutation is position-independent, without taking into account the bit-string length or the bit position into the bitstring.
- Single point cross-over with the probability position-independent, without taking into account the quality of the bitstrings.
- Resulting chromosomes replace their parents.
- Selection proportional with the fitness value (Roulette-wheel[9])

3 Experiments:

The experiments were conducted on the following 4 functions:

1. DeJong's function
2. Schwefel's function
3. Rastrigin's function
4. Michalewicz's function

For all the functions the algorithms were executed 30 times with a precision of 5 decimals. The search spaces chosen were 5, 10 and 30. In the first part of the experiment for all search spaces (5,10,30) the population size was 200 and the number of generations was 1000. Later we are only focussing on the search space of 30 where there were used populations of 100,200 and 1000 and the number of generations were 1000,5000 respectively 10000. The mutation probability (0.01) and cross-over probability (0.2) are fixed. At each run we are randomly initialising a population of individuals. The random number generator is the standard implementation of a Mersenne Twister[13].

3.1 DeJong Function

$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 \leq x_i \leq 5.12$$

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

DeJong's function is also known as a sphere model. It is continuos, convex and unimodal.

Fitness function chosen: $f(x)^{-3}$

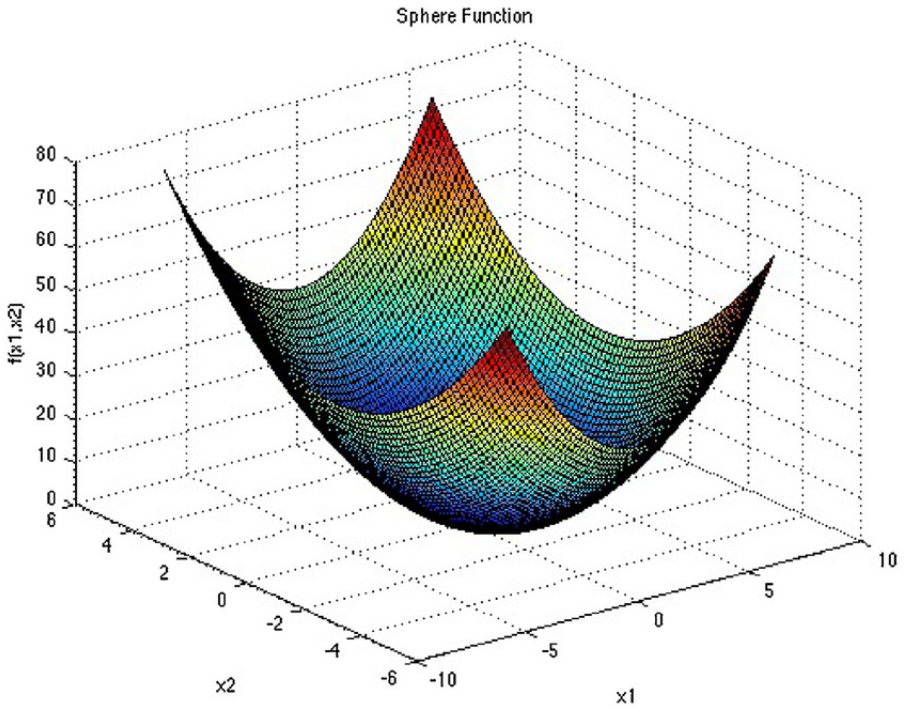


Figure 1: DeJong function 1 graph for n=2

Size	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0	0	0	0	1.61	1.467	2.067
10	0	0	0	0	2.89	2.73	3.52
30	0	0	0	0	8.467	7.737	10.5

*Actual values were smaller than 10^{-3} so they were approximated to 0

Fitness function: $f(x)^{-3}$ | Nr. Generations: 1000 | Population size: 200

Table 1: DeJong function 1 result on Genetic Algorithm

3.2 Schewefel's function

$$f(x) = \sum_{i=1}^{n} -x_i * sin(\sqrt{|x|}) \qquad -500 \leq x_i \leq 500$$

Global minimum: $f(x) = -n * 418.9829, x(i) = 420.9686, i = 1 : n$

Schwefel's function global minimum is geometrically distant over the parameter space, from the next best local minima. Because of this the search algorithms are potentially prone to converge in the wrong direction.

Fitness function chosen: $(\frac{(500*n*2)}{(f(x)+500*n)})^{10}$

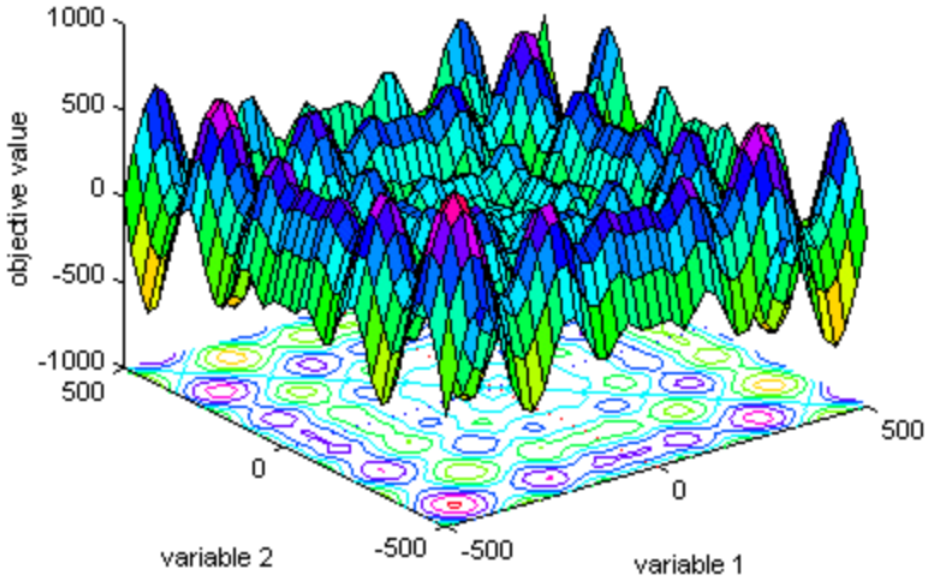


Figure 2: Schwefel's graph for n=2

Size	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-2094.323	0.288	-2094.78	-2093.67	2.168	2.113	2.615
10	-4179.144	13.845	-4188.13	-4143.43	4.019	3.97	4.307
30	-12176.34	96.6387	-12350.5	-11925.8	11.527	11.416	12.128

Fitness function: $(\frac{(500*n*2)}{(f(x)+500*n)})^{10}$ | Nr. Generations: 1000 | Population size: 200

Table 2: Schwefel function 1 result on Genetic Algorithm

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	-11891.04	191.214	-12246.8	-11506.5	2.778	2.751	2.914
5000	-12283.73	87.9274	-12414	-12110.3	13.824	13.777	14.163
10000	-12271.67	84.36319	-12438.1	-12090.7	27.68	27.579	28.524

Fitness function: $(\frac{(500*n*2)}{(f(x)+500*n)})^{10}$ | **Population size: 100**

Table 3: Schwefel function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	-12176.34	96.638	-12350.5	-11925.8	11.527	11.416	12.128
5000	-12379.85	39.734	-12436.23	-12275.9	27.63	27.566	27.998
10000	-12394.57	41.337	-12461.8	-12306.4	55.254	55.094	55.678

Fitness function: $(\frac{(500*n*2)}{(f(x)+500*n)})^{10}$ | **Population size: 200**

Table 4: Schwefel function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s
1000	-12457.391	37.821	-12497.12	-12330.3	27.908	27.698	28.351
5000	-12460.32	16.779	-12489.32	-12415.286	139.2633	138.781	140.472
10000	-12475.63	20.855	-12519.2	-12425.89	278.499	277.795	281.233

Fitness function: $(\frac{(500*n*2)}{(f(x)+500*n)})^{10}$ | **Population size: 1000**

Table 5: Schwefel function result on Genetic Algorithm for size 30

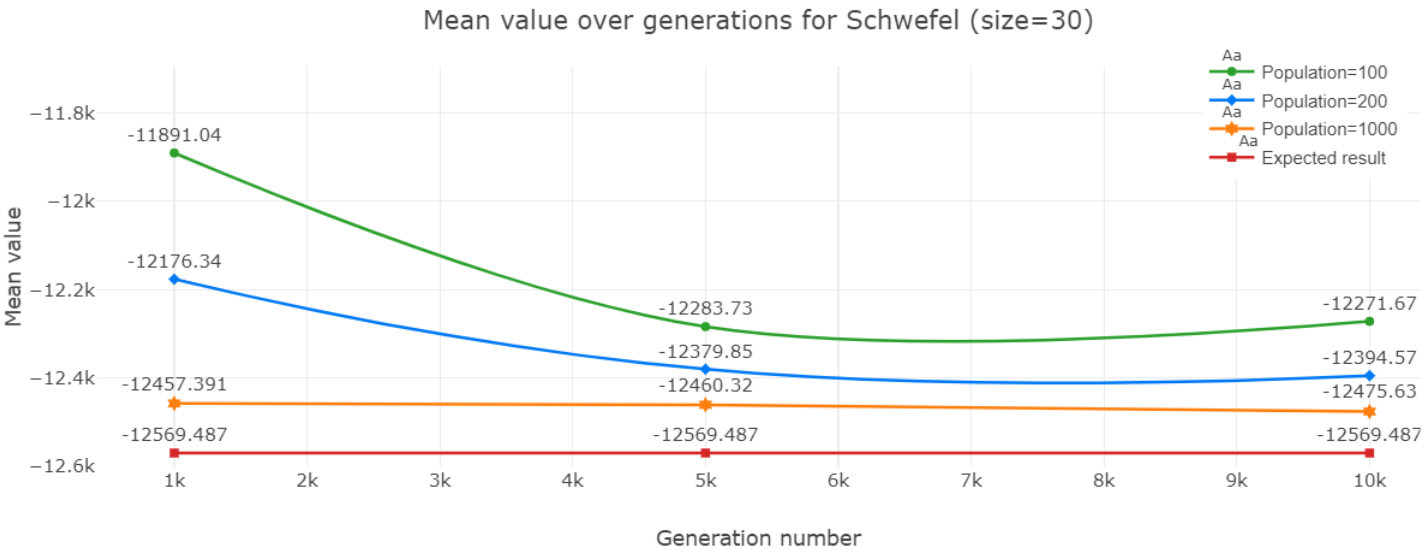


Figure 3: Mean value over generations for Schwefel(size=30)

3.3 Rastrigin's function

$$f(x) = 10 * n + \sum_{i=1}^{n} (x_i^2 - 10 * cos(2 * \pi * x_i)) \qquad -5.12 \leq x_i \leq 5.12$$

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

Rastrigin's function is multimodal and the location of the minima are distributed. The value of the function increases with the distance from the origin so the global minimum can only be obtained in the origin.

Fitness function chosen: $f(x)^{-10}$

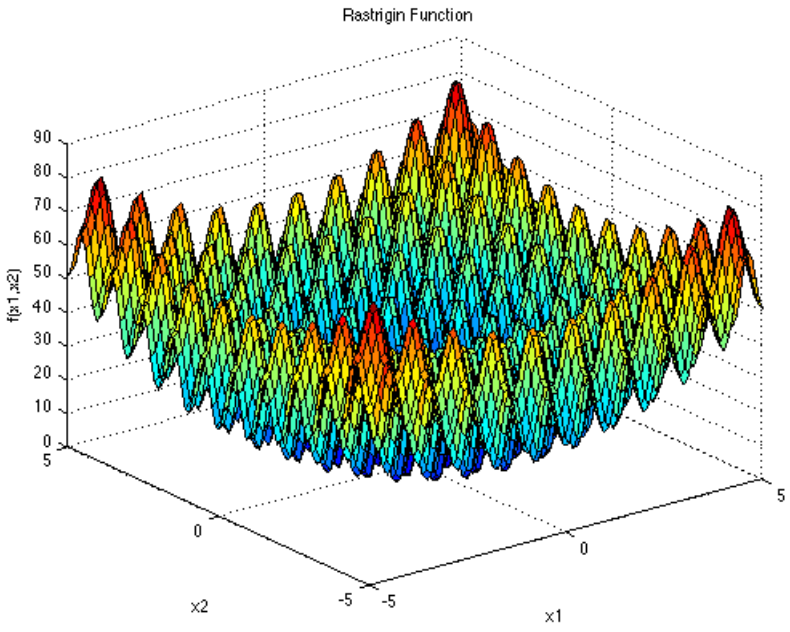


Figure 4: Rastrigin graph for n=2

Size	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	0.055	0	0	1.646	5.402	5.3	6.132
10	0.638	0.803	0	1.653	5.185	4.93	6.168
30	27.615	6.905	14.791	42.716	9.56	9.339	10.36

*Actual values were smaller than 10^{-3} so they were approximated to 0

Fitness function: $f(x)^{-10}$ | Nr. Generations: 1000 | Population size: 200

Table 6: Rastrigin function result on Genetic Algorithm

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	38.614	8.644	24.877	53.8912	2.448	2.358	2.764
5000	16.301	5.874	4.039	32.726	12.313	12.102	12.616
10000	9.45	5.016	1.583	21.402	25.542	23.7	30.195

Fitness function: $f(x)^{-10}$ | **Population size: 100**

Table 7: Rastrigin function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	27.615	6.905	14.791	42.716	9.56	9.339	10.36
5000	10.643	5.621	2.021	24.941	26.4259	25.455	29.9
10000	5.677	3.475	0.009	16.4131	55.953	50.885	59.952

Fitness function: $f(x)^{-10}$ | **Population size: 200**

Table 8: Rastrigin function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	12.75	6.17	3.508	28.908	30.538	28.106	42.11
5000	3.563	1.863	0	6.996	157.119	131.713	223.834
10000	0.907	1.154	**0**	3.578	206.047	200.495	229.112

*Actual values were smaller than 10^{-3} so they were approximated to 0

Fitness function: $f(x)^{-10}$ | **Population size: 1000**

Table 9: Rastrigin function result on Genetic Algorithm for size 30

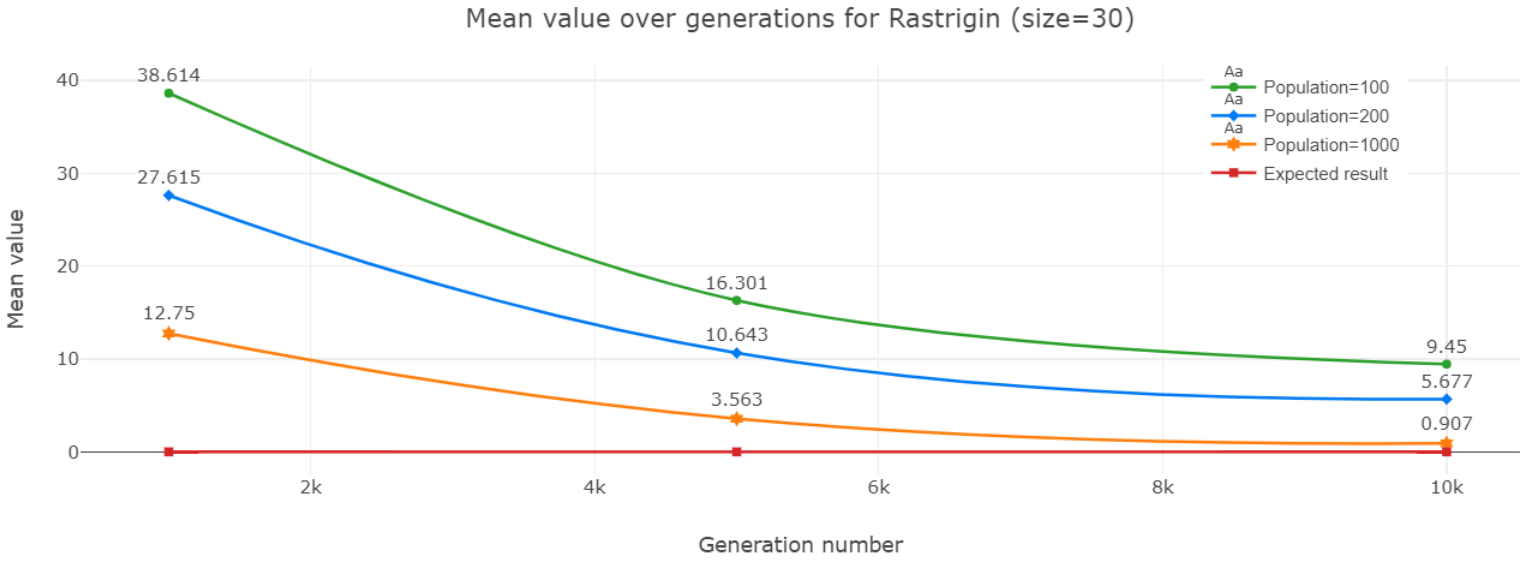


Figure 5: Mean value over generations for Rastrigin (size=30)

3.4 Michalewicz's function

$f(x) = -\sum_{i=1}^{n} sin(x_i) * (sin(\frac{i*x_i^2}{\pi}))^{2*m}$, $i = 1 : n, m = 10, 0 \leq x_i \leq \pi$

Michalewicz's function is a multimodal function with n! local minima where m defines the steepness of the valleys and ridges. Since the global minima becomes difficult to search when m reaches a larger value its recommended value is 10.

Fitness function chosen: $e^{f(x)^{-10}}$

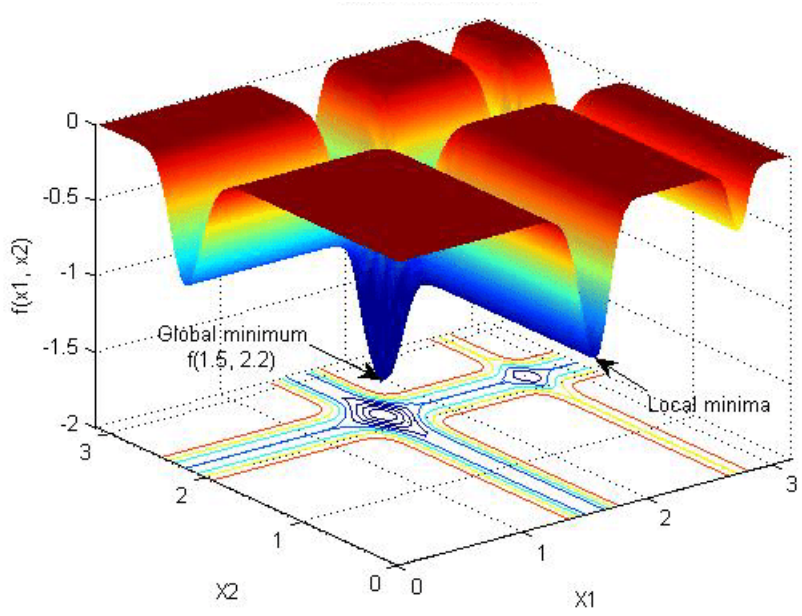


Figure 6: Michalewicz's function for n=2

Size	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
5	-4.570	0.141	-4.687	-4.039	1.011	0.962	1.319
10	-9.032	0.342	-9.62	-7.9	1.796	1.73	2.05
30	-27.25	0.418	-27.989	-26.4571	5.353	5.022	6.51

Fitness function: $e^{f(x)^{-10}}$ | Nr. Generations: 1000 | Population size: 200

Table 10: Michalewicz's function result on Genetic Algorithm

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	-26.511	0.553	-27.466	-25.051	1.935	1.911	2.104
5000	-27.321	0.486	-27.96	-25.859	9.683	9.619	9.947
10000	-27.724	0.422	-28.316	-26.301	19.23	19.173	19.596

Fitness function: $e^{f(x)^{-10}}$ | **Population size: 100**

Table 11: Michalewicz's function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	-27.25	0.418	-27.989	-26.4571	5.353	5.022	6.51
5000	-27.789	0.498	-28.743	-26.783	19.213	19.18	19.374
10000	-28.039	0.35	-28.95	-27.468	40.125	38.384	45.979

Fitness function: $e^{f(x)^{-10}}$ | **Population size: 200**

Table 12: Michalewicz's function result on Genetic Algorithm for size 30

Gen. Nr.	Mean	σ	Min	Max	Avg Time(s)	Min Time(s)	Max Time(s)
1000	-27.848	0.577	-28.717	-26.464	19.62	19.573	19.829
5000	-28.401	0.392	**-29.174**	-27.556	97.893	97.736	98.005
10000	-28.601	0.29	-29.103	-28.031	198.058	195.453	217.579

Fitness function: $e^{f(x)^{-10}}$ | **Population size: 1000**

Table 13: Michalewicz's function result on Genetic Algorithm for size 30

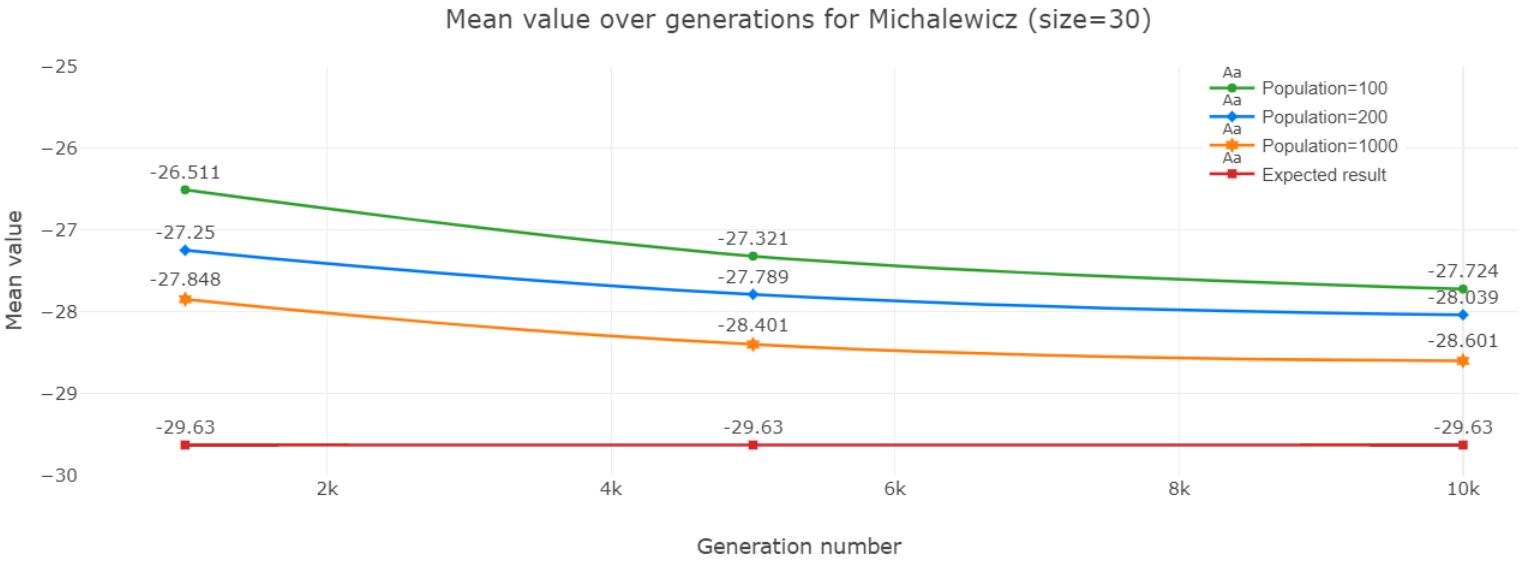


Figure 7: Mean value over generations for Michalewicz (size=30)

4 Observations

From these experiments we can clearly see that there is no "Jack of all trades" genetic algorithm for the minimization problem. Even if for the simplest test function (DeJong) the expected result was obtained for all sizes, on others with a numerous local minima the algorithm fails to give an exact answer even for a smaller search space (f.i. Schwefel run on 5 dimension returned a minima 0.015 bigger). One of the reasons is the fact that a genetic algorithm can not recognize an optimal solution when it is found, therefore we might replace that solution in the evolution process.

4.1 Roulette-wheel selection

This method guarantees a fitness proportionate selection by assigning to each individual a probability in regards with its quality. However, this method can lead to populations overflown with a single chromosome in the next generation, thus converging to a possibly local minima instead of a global one. On the other hand this method guarantees a chance of selection for all of the solutions so we are able to evaluate worse solutions that might improve better over time.

4.2 Selection pressure

Selection pressure dictates if our algorithm will focuss upon either exploration or exploitation. If the selection pressure is high, fitter solutions are more likely to survive or be chosen as parents, favouring the exploitation of better solution.

Hence DeJong function has only one global maximum and no local one, a fitness that has a high selection pressure is most suitable for this problem, because it restricts our search space towards the best chromosomes, therefore exploiting the best solutions. An increased selection pressure however also brings premature convergence, meaning that the population will mostly search towards the same direction. For functions where the minima are distributed over a large search space this approach is not useful and might end in a local minima rather than the global one.

4.3 Computing a fitness function

The fitness function is the function that rates the quality of our chromosomes. It states the quality of a chromosome and dictates its chances of being selected in the next generation. A drawback of this type of fitness-proportionate selection is the fact that in the latter generations most of the fitness values are similar, therefore the selection pressure decreases.

Schwefel function is one of the hardest to optimize, mainly because of its minima distribution and increased search space available. Since Schwefel function can take negative values a constant needs to be added to have a positive result. We know that the fitness function is a sum of terms $(-x_i * sin(\sqrt{|x|})$ and since the function is defined on $-500 < x_i < 500$ we can deduct that the minimum point might occur when either:

1. $(sin(\sqrt{|x|}) = 1$ and $x_i = 500$ in every term
2. $(sin(\sqrt{|x|}) = -1$ and $x_i = -500$ in every term

Therefore, the constant added can be $500 * n$. Now the results are strictly positive, and we can use a fitness function of $\frac{1}{f(x)}$ where the codomain of the new function is $[0, (500 * n + 500 * n)]$. In early tests this type of search was proven to be rather chaotic so we increased the selection pressure by raising it to a negative power of -10 (this number was chosen because in the initial tests in had the best average results). In order to avoid overflowing the results we also previously reduced the interval to $[0, 1]$ by dividing the function with it's upper bound codomain value.

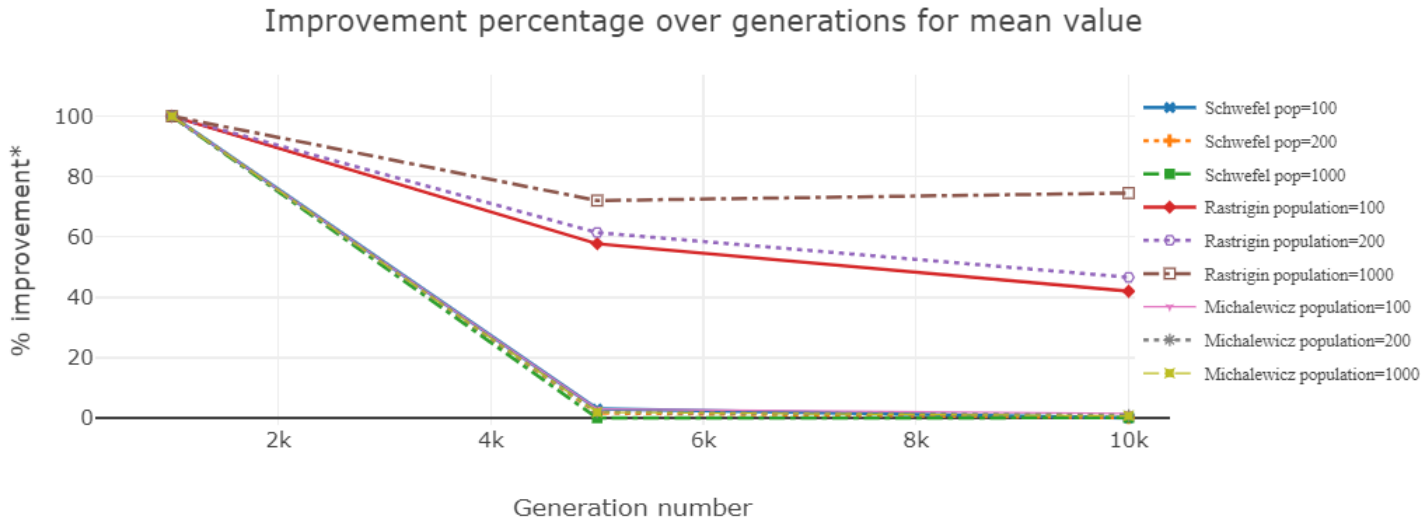
Other functions might be simpler to find as we can see in the Rastrigin's case where the function is from the start strictly positive over a small inteval and it's enough to raise the pressure of selection by raising the function to a negative power to get favourable results.

4.4 Increasing the parameters

Even if the results proved to be better than what we might find with other heuristic search algorithms (f.i. Hillclimber, Simulated Annealing) in the environment with a population of 200 and 1000 generations I decided to explore how these would change by increasing the population respectively generation size for the functions where the best result was not already found.

From figures 3,5 and 7 we can clearly see that every function benefited from this "extension" returning always a better mean but not necesarily a better minima. The main aspects that have been concluded by this empirical research were:

- The improvement of a population over generations is logarithmic. Therefore, after a certain iteration we are wasting computer power that could be used for a new run.



* Improvement percentage is calculated by comparating the current mean with the mean from the previous generation set

Figure 8: Improvement percentage over generations for mean value

- An increased generation count allows for a selection of more fit chromosomes, increasing the quality of the population.
- The computational time is linear with the population size and the number of generations.
- For the Rastrigin function we obtained the global minima because the algorithm had enough time to search over an decreasing search space.
- This type of "extension" might prove that the fitness function chosen is innefective. As we can see in the figure 3, with a population of 1000 the improvements over generations are so low that the line is almost parralel with the expected result. One of the reasons of this paralelism is the fact that the constant added was too big and it decreased the pressure of selection up to a point that there were very few improvements available. *Other tests were conducted with a smaller constant (13000) but the results didn't improve.

4.5 Cross-over and mutation

Cross-over and mutation are essentially the main operators in genetic algorithms. At the start of the experiment smaller scale runs were tested with different probabilities of mutation(0.001, 0.01, 0.1) and crossover (0.2, 0.5, 0.8) but the results either had a huge standard deviation (chaotic results far from expected results) or the evolution process was too slow. The main drawback of these operators in our experiment is the posibility of replacing a better solution with a worse one after applying them, even if in theory this solution can be refound in the evolution process.

4.6 Genetic Algorithm vs other non-deterministic approaches

The following section contains informations that can be studied in depth in **this[5]** report. In the previous report we have talked about some non-deterministic approaches of minimizing a function. The one studied were: HillClimbing (First and Best Improvement) and Simulated Annealing. Along with the Genetic Algorithm discussed in this report all are part of the class of **probabilistic algorithms**.

The main difference is the fact that when we are using a Genetic Algorithm we are evaluating a set of candidates at every run in comparison with the other approaches studied before that where we are evaluating a single candidate. Thus, the Genetic Algorithm has the capacity of searching a bigger space and also allows a more diverse exploration.

An unexpected result observed from the experiments its the fact that on a smaller sample size HillClimbing and Simulated Annealing tend to outmatch the Genetic Algorithm. One of the reasons is the distructive manner of our Genetic Algorithm that can not recognize an optimum solution, thus replacing it with a worse one in the evolving process while the HillClimbing will always move downwards.

The following table shows for every function the best result obtained with our Genetic Algorithm followed by the best result obtained from any non-deterministic algorithm studied in the previous report. The results in bold represent the best result obtained at the certain size of the function:

Function	Algorithm*	Size	Expected Result	Actual Result	Error(Act-Exp)
Schwefel	GA	5	-2094.914	-2094.78	0.083
	HCB			**-2094.91**	0.004
	GA	10	-4189.829	-4188.13	1.699
	HCB			**-4189.52**	0.309
	GA	30	-12569.487	**-12519.2**	50.528
	SA			-11822.4	747.087
Rastrigin	**GA**	5	0	**0**	0
	HCB/HCF/SA			**0**	0
	GA	10	0	**0**	0
	HCB			2.230	2.230
	GA	30	0	**0**	0
	HCB			23.364	23.364
Michalewicz	**GA**	5	-4.68765	**-4.687**	0
	HCB			**-4.687**	0
	GA	10	-9.66	**-9.62**	0.04
	HCF			-9.492	0.167
	GA	30	-29.63	**-29.174**	0.456
	HCB			-27.479	2.151

*Notations: GA= Genetic Algorithm | HCB= Hillclimbing Best Improvement | HCF= Hillclimbing First Improvement | SA=Simulated Annealing

Table 14: Best result from Genetic Algorithm compared with the best result from another non-deterministic method

We can see that even if we had decent results with all methods at a smaller sample size, for all functions tested the Genetic Algorithm scales better with the dimension of the function. Hence,

for functions that have an increased number of local minima the ability of searching in multiple directions and evolving multiple solutions at a time of the Genetic Algorithm outperforms the others. The time required for returning an aproximation with a Genetic Algorithm is also smaller than a HillClimber because we are not evaluating all the possible solution but rather the best ones based on the fitness function.

5 Conclusion

Looking at the results presented we can conclude that Genetic Algorithms can be used in the process of minimizing a function because they can compute favourable estimations in a decent time. A good selection of the parameters improve both computation time and the accuracy of the solution. This process however requires time and detailed analysis is order to find the cause of errors and best solving approach.

The results are overall better than other heuristic algorithms used in function minimization when applied to functions with several number of minima because of the capacity of searching a bigger space. The quality of the results can be improved by increasing the population size and maximum number of generations but it is worth to stop the algorithm in a state where the improvements over generations no longer be argued by the computing power wasted.

Actual problems that still remain unanswered when studying genetic algorithms remain the process of choosing a fitness function, and a selection method such that the algorithm favours exploration of the search space while also exploiting the best ones further in the process.

References

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

[2] Genetic Algorithms Theoretical concepts
https://profs.info.uaic.ro/~eugennc/teaching/ga/res/gaSlidesOld.pdf

[3] Functions used and images
http://www.geatbx.com/docu/fcnindex-01.html#P89_3085

[4] Genetic Algorithms general informations
https://www.geeksforgeeks.org/genetic-algorithms/

[5] HillClimbing and Simulated Annealing report
https://drive.google.com/file/d/1zjJ7JFevY8Yxao5lTzfkHLIEWXPug7JD/view?usp=sharing

[6] Fitness function computing methods
http://students.info.uaic.ro/~vladut.ungureanu/Algoritmi-genetici-ID.pdf

[7] Report structure
https://gitlab.com/eugennc/research/-/blob/master/2019SYNASC/synasc2015-paper18.pdf

[8] Influence of Population on the Genetic Algorithm
https://annals-csis.org/Volume_1/pliks/167.pdf

[9] Roulette-Wheel Selection
https://medium.com/datadriveninvestor/genetic-algorithms-selection-5634cfc45d78

[10] Graph editor
http://mathcha.io

[11] Simulated Annealing vs Genetic Algorithms
https://www.seas.upenn.edu/~cis391/Lectures/informed-search-II.pdf

[12] Sorting based on probability
https://stackoverflow.com/questions/37368787/c-sort-one-vector-based-on-another-one/46370189

[13] Mersene twister random number generator
https://www.guyrutenberg.com/2014/05/03/c-mt19937-example/

[14] When a Genetic Algorithm outperforms Simulated Annealing
https://bit.ly/3lkcj8b

[15] Good Genetic Algorithm practices
http://gpbib.cs.ucl.ac.uk/gecco2000/GA110.pdf

[16] Genetic Algorithms in optimization
https://www.researchgate.net/publication/283361244_Genetic_Algorithm_-_an_Approach_to_Solve_Global_Optimization_Problems