Analysis of Genetic Algorithms on a set of functions

Zaborilă Andrei

December the 2nd, 2020

1 Introduction

"A **genetic algorithm** is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection"[3]

This report is meant to present the behaviour of a genetic algorithm on a set of four functions. Our goal is to determine an approximation of the global minima of said functions using a genetic algorithm and to analyze its efficiency and accuracy.

2 Method

In a genetic algorithm a starting population goes through evolution for a certain number of generations. A population is a group of chromosomes, and a chromosome represents a possible solution. The whole process is composed of a few important steps:

- **Selection** is the process of choosing which chromosomes will survive into the next generation, more fit individuals having a higher chance of survival.
- **Mutation** is the process of mutating a chromosome.
- **Cross-Over** is the process in which two or more chromosomes interchange genetic information.
- **Evaluation** is the process in which every chromosome is evaluated and is provided with a certain fitness, according to a set criterion.

In the used algorithm we initialize a starting population with random values. Then, for each generation, we first put them through the process of **Selection**, in which we create a new, more fit, population from the previous one. We then iterate through all the chromosomes and try to mutate them, each bit of each chromosome having a chance to mutate. After that they go through the process of **Cross-Over** and then we evaluate each of them, by applying the current fitness function and storing its result for each chromosome, it representing that chromosome's fitness. After going through all the generations, we select the best individual from the last population and provide it as the result of the algorithm.

This whole process is very similar to the real-life concept of "Survival of the fittest". From a given population, only the fittest individuals tend to survive to the next one. Each individual also has a chance to mutate, thus avoiding getting stuck in a certain local minima. A huge advantage that the genetic algorithms have when compared to other non-deterministic algorithms, such as Hill Climbing, is the time efficiency; in general, a genetic algorithm is very fast, and it also provides better results if well-configured.

A sample size of **36** was chosen for each size (**5, 10, 30**) of each function and **12** threads will be used to run the experiments, on an AMD Ryzen 3700X.

3 Experiments

A precision of five decimals has been chosen for all the experiments, so all the results have been truncated where needed.

3.1 De Jong's function

3.1.1 The function

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

Global minimum: $DeJong(x) = 0, \quad x_i = 0, \quad i = 1 : n$

De Jong's function is also known as "The Spehere function". It has a number of local minima equal to the number of dimensions, except for the global one. It is continuous, convex and unimodal. **Figure 1** shows its two-dimensional form.[8]

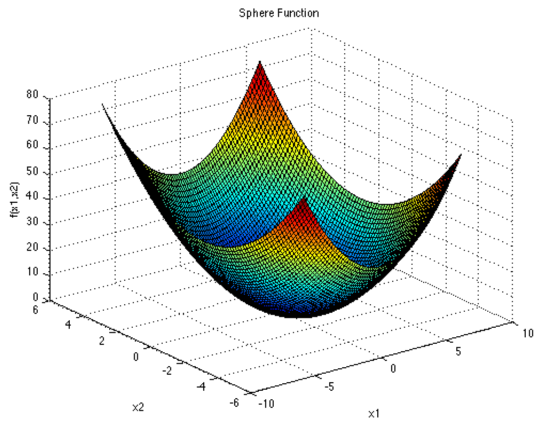

Figure 1: De Jong's function

3.1.2 The fitness function and the setup

Setup:

Population Size = 200, Generations = 2500, Mutation Chance = 1%, Crossover Chance = 20%

$$Fitness_{DeJong}(x) = \frac{1}{DeJong(x)^{10}} \qquad x \in [-5.12, 5.12]^n,\ n \in \{5, 10, 30\}$$

Knowing that DeJong's function's results are always positive, and the minimum is 0, the fitness function should return a value inversely proportional to that of DeJong's.

3.1.3 Results

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	0	0	0	0	2.412	2.647	2.568
10	0	0	0	0	4.618	4.827	4.716
30	0	0	0	0	12.98	13.978	13.144

Table 1: Results on De Jong's function

3.2 Schwefel's function

3.2.1 The function

$$Schwefel(x) = \sum_{i=1}^{n} -x_i \cdot \sin(\sqrt{|x_i|}) \qquad -500 \le x_i \le 500$$

$$\text{Global minimum: } Schwefel(x) = n \cdot 418.9829, \quad x_i = 420.9687, \quad i = 1:n$$

The Schwefel function is complex, with many local minima. **Figure 2** shows the two-dimensional form of the function.[9]

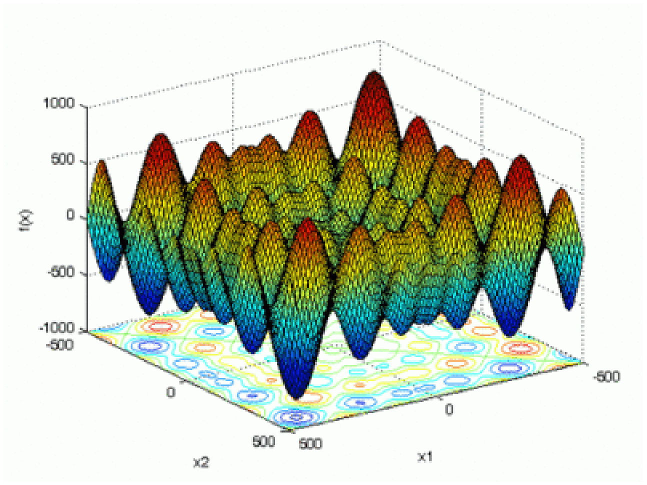

Figure 2: Schwefel's function

3.2.2 The fitness function and the setup

Setup:

Population Size = 250/500/1000, Generations = 2500, Mutation Chance = 2%, Crossover Chance = 20%

$$Fitness_{Schwefel}(x) = \begin{cases} 1.01^{arctan(\frac{1e-4}{Schwefel(x)} \cdot |Schwefel(x)|)} & Schwefel(x) > 0 \\ 1.01^{arctan((\frac{1e-4}{Schwefel(x)} + \pi) \cdot |Schwefel(x)|)} & Schwefel(x) < 0 \end{cases} \qquad x \in [-500, 500]^n, \; n \in \{5, 10, 30\}$$

Schwefel's function poses multiple problems when trying to find a proper fitness function. First of all, the global minima is negative, while the values returned by Schwefel's function can be both positive and negative. In order to (almost) ignore positive values we use a variation of the *arctan* function, *atan*2[7]. It takes two arguments, y and x, and has two cases. If $x > 0$ then it will return $arctan(\frac{y}{x})$; otherwise it will return $arctan(\frac{y}{x} + \pi)$. Since we use $y = 1e-4$, *atan*2 will return a value close to 0 for positive values of Schwefel's. We then multiply this with the absolute value of Schwefel's and then raise 1.01 to the power of our previous result. This will result in a more strict fitness function, it being exponential.

3.2.3 Results for population 250

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-2094.83	-2093.55	-2094.485	0.30806	4.122	4.342	4.247
10	-4188.99	-4179.71	-4185.319	1.95201	7.726	8.135	7.89
30	-12499.5	-12366.3	-12454.69	33.75011	21.95	22.565	22.216

Table 2: Results on Schwefel's function, population 250

3.2.4 Results for population 500

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-2094.87	-2094.32	-2094.69	0.12937	8.095	9.793	8.366
10	-4189.32	-4183.56	-4187.227	1.29472	14.98	15.536	15.279
30	-12527.2	-12468.9	-12494.17	12.84797	42.461	43.798	43.064

Table 3: Results on Schwefel's function, population 500

3.2.5 Results for population 1000

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-2094.88	-2094.55	-2094.786	0.06539	17.456	17.751	17.599
10	-4189.48	-4186.2	-4188.299	0.66057	32.808	33.09	32.944
30	-12530.2	-12480.5	-12512.8	12.53855	90.297	91.182	90.752

Table 4: Results on Schwefel's function, population 1000

3.2.6 Comparison

In **Figure 3** we can see a comparison between the results obtained with different population sizes, regarding the difference between the expected result (actual global minima) and the result the algorithm returned.

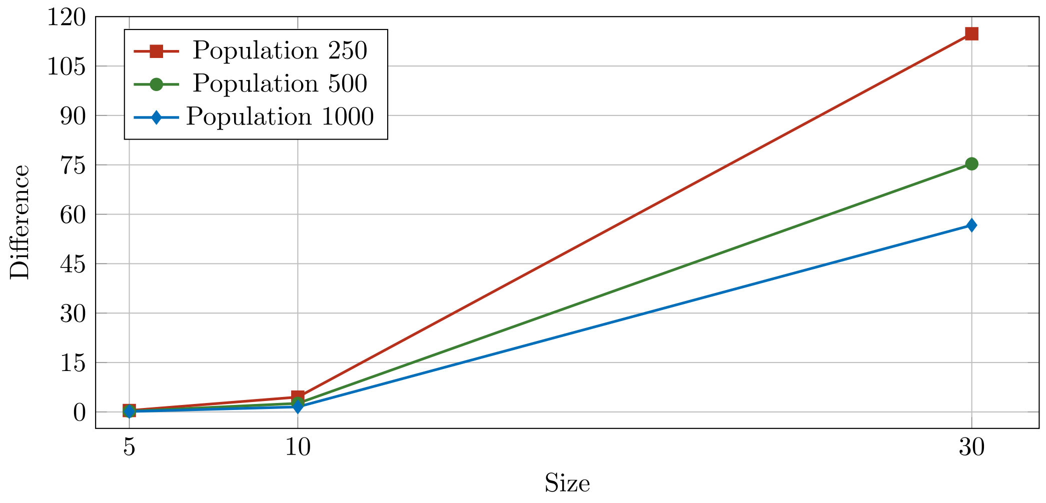

Figure 3: Difference between expected result and actual result

It's easy to see that the higher the population, the lower the difference gets, so we can draw the conclusion that if we want better results we need to increase the population. We can also see that the difference between **Population 250** and **Population 500** is almost double the difference between **Population 500** and **Population 1000**. We can conclude that the more we raise the population size, the lower the impact on the difference will be. Therefore we will eventually reach a point where increasing the population size will become futile, the improvement being negligible.

3.3 Michalewicz's function

3.3.1 The function

$$Michalewicz(x) = -\sum_{i=1}^{n} \sin(x_i) \cdot \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{2 \cdot m} \qquad i = 1 : n,\ m = 10,\ 0 \leq x_i \leq \pi$$

Global minimum: $Michalewicz(x) = -4.687$, for n = 5
$Michalewicz(x) = -9.66$, for n = 10
$Michalewicz(x) = -29.63$, for n = 30

The Michalewicz function has d! local minima, where d is the number of dimensions, and it is multimodal. The parameter m defines the steepness of the valleys and ridges; a larger m leads to a more difficult search. The recommended value of m is m = 10. The function's two-dimensional form is shown in **Figure 4**.[11]

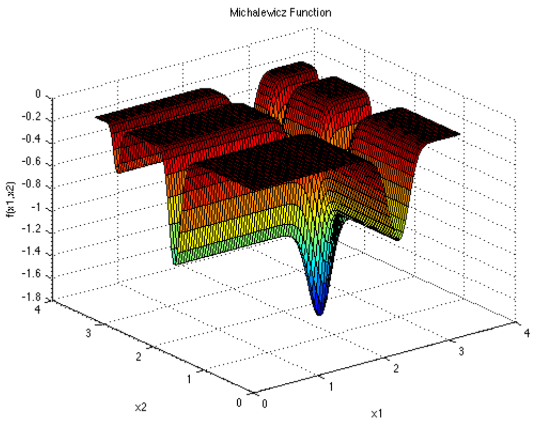

Figure 4: Michalewicz's function

3.3.2 The fitness function and the setup

Setup:

Population Size = 250/500/1000, Generations = 10000, Mutation Chance = 1%, Crossover Chance = 25%

$$Fitness_{Michalewicz}(x) = 1.05^{Michalewicz(x)^2}$$

The main problem that arises when trying to find the global minima for Michalewicz's function is the fact that it has d! local minima, d being the number of dimensions. Since we are studying the behaviour of the genetic algorithm on sizes 5, 10 and 30, the latter has an absurd amount of local minima. To solve this problem solely with the fitness function is a tough task, and so changes to the number of generations and population size, combined with a more strict fitness function (an exponential one), are required to accomplish it.

3.3.3 Results for population 250

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-4.68008	-4.59996	-4.63854	0.02153	13.524	16.476	13.729
10	-9.60935	-9.3157	-9.50287	0.06761	24.352	27.865	24.656
30	-28.6498	-27.8168	-28.20142	0.24865	67.56	72.279	68.281

Table 5: Results on Michalewicz's function, population 250

3.3.4 Results for population 500

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-4.68504	-4.61109	-4.65557	0.02191	25.84	26.843	26.567
10	-9.6252	-9.35155	-9.54046	0.06548	46.839	48.079	47.497
30	-28.9874	-27.6806	-28.43006	0.30506	128.783	139.076	130.87

Table 6: Results on Michalewicz's function, population 500

3.3.5 Results for population 1000

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	-4.68298	-4.6317	-4.65775	0.01997	57.149	69.869	58.929
10	-9.63886	-9.51425	-9.5786	0.034043	102.395	116.96	103.951
30	-29.0489	-28.0609	-28.67256	0.23154	278.287	282.358	280.217

Table 7: Results on Michalewicz's function, population 1000

3.3.6 Comparison

In **Figure 5** we can see a comparison between the results obtained with different population sizes, regarding the difference between the expected result (actual global minima) and the result the algorithm returned.

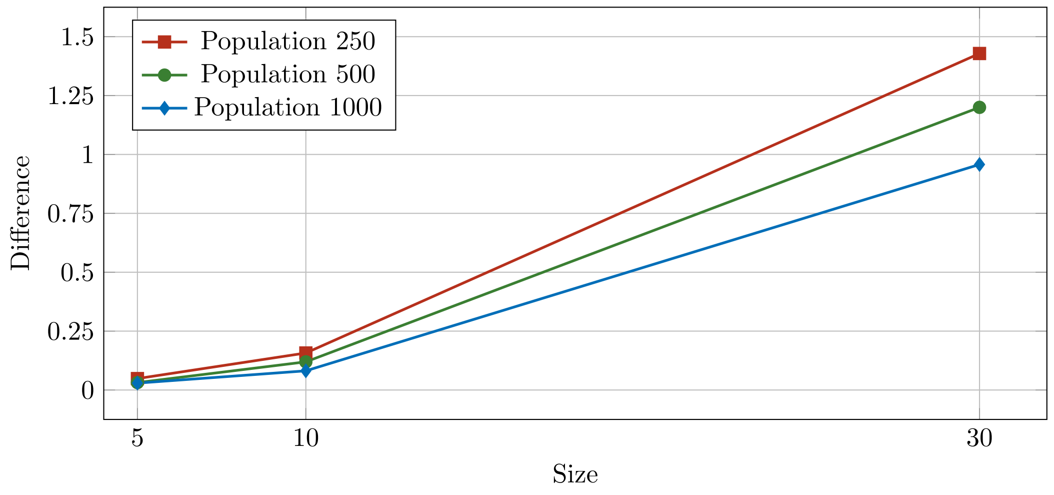

Figure 5: Difference between expected result and actual result

Similar to the behaviour on Schwefel's function, the algorithm returns better results the higher the population. We can see that the difference between **Population 250** and **Population 500** is similar to the one between **Population 500** and **Population 1000**, unlike what we saw in **Figure 3**, where the impact of the population got hastily more insignificant the higher we raised it. Therefore increasing the population size is not as taxing as it was on Schwefel's function, and we can keep improving the accuracy of the algorithm by increasing it (we will eventually reach a population size for which the time cost would not justify the increase).

3.4 Rastrigin's function

3.4.1 The function

$$Rastrigin(x) = 10 \cdot n + \sum_{i=1}^{n}(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)) \qquad -5.12 \le x_i \le 5.12$$

Global minimum: $Rastrigin(x) = 0, \quad x_i = 0, \quad i = 1:n$

The Rastrigin function has several local minima. It is highly multimodal, but locations of the minima are regularly distributed. **Figure 6** shows the two-dimensional form of the function.[10]

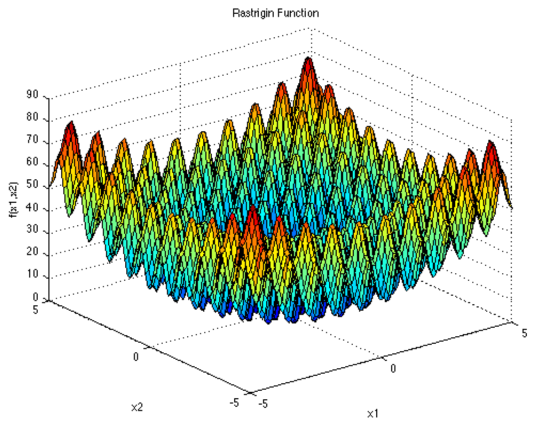

Figure 6: Rastrigin's function

3.4.2 The fitness function and the setup

Setup:

Population Size = 250/500/1000, Generations = 10000, Mutation Chance = 1.25%, Crossover Chance = 30%

$$Fitness_{Rastrigin}(x) = \frac{1}{Rastrigin(x)^{10}}$$

Rastrigin's function has many local minima, a problem that is hard to overcome simply through the fitness function. Similar to Michalewicz's, a good combination of population size, generations and fitness function is needed to provide good results. The fitness function itself is identical to the one used on DeJong's because, as we can see in **Figure 7**, the functions themselves are quite similar, Rastrigin's simply having way more local minima. Since said fitness function worked well with DeJong's, we only need to increase the searching space to accommodate to the high number of local minima, which is accomplished by increasing the population size, and the number of generations.

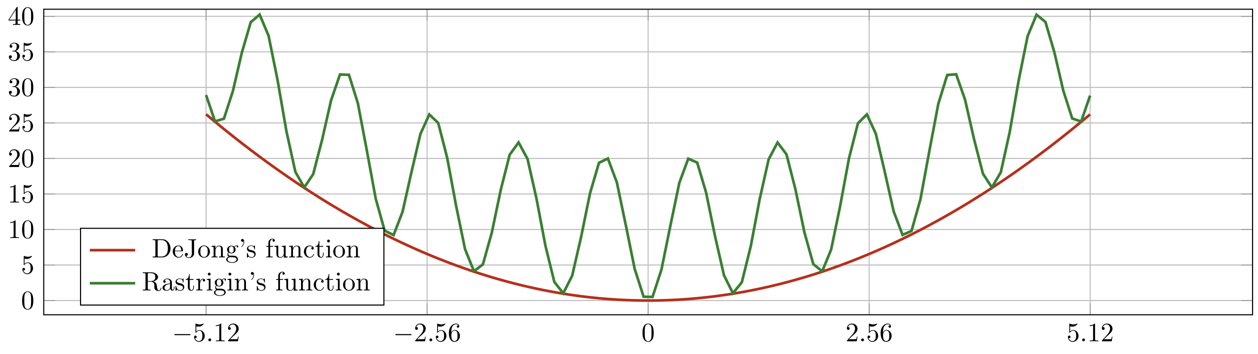

Figure 7: DeJong's function and Rastrigin's function with one dimension

3.4.3 Results for population 250

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	0	0	0	0	13.34	13.52	13.449
10	0	0	0	0	24.243	28.597	26.083
30	0.01521	13.331	4.33176	3.11942	72.503	73.689	73.05

Table 8: Results on Rastrigin's function, population 250

3.4.4 Results for population 500

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	0	0	0	0	26.181	26.9	26.548
10	0	0	0	0	46.423	47.766	47.254
30	0.00095	6.51695	2.38078	1.79257	127.183	130.113	128.832

Table 9: Results on Rastrigin's function, population 500

3.4.5 Results for population 1000

Dimension	Min	Max	Mean	σ	Min Time	Max Time	Mean Time
5	0	0	0	0	56.318	58.002	57.37
10	0	0	0	0	100.68	101.546	101.141
30	0	3.78674	0.76919	1.02847	262.057	289.119	270.67

Table 10: Results on Rastrigin's function, population 1000

3.4.6 Comparison

In **Figure 8** we can see a comparison between the results obtained with different population sizes, regarding the difference between the expected result (actual global minima) and the result the algorithm returned.

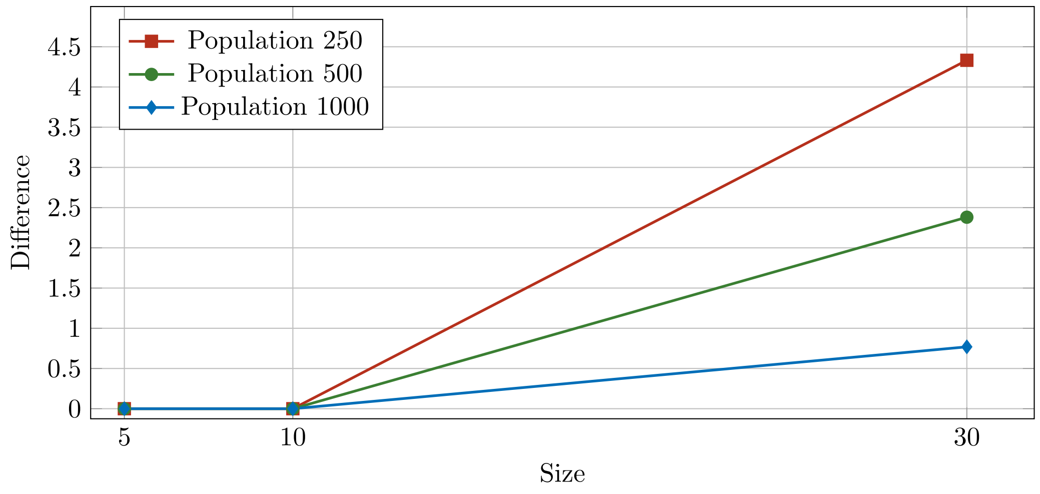

Figure 8: Difference between expected result and actual result

As we have seen before, increasing the population improves the results, especially on functions with a high amount of local minima, since it enlarges the search area. The mean value for **Population 1000** is great, the difference between it and the actual global minima being less than one, but what is far greater is the best result obtained. **Table 11** best illustrates how close we came to the global minima with at least one value.

Population Size	Mean Time
250	0.01521
500	0.00095
1000	0

Table 11: Best results on Rastrigin's with size 30

Using a population size of 250 already brings us close to the global minima, 0. The population size of 500 is even closer and with population size 1000 we actually get the global minima of Rastrigin's at least once.

Population Size	Best result
250	73.05
500	128.832
1000	270.67

Table 12: Mean time on Rastrigin's with size 30

Looking at the time it took to obtain these results we can see that it's almost insignificant compared to other non-deterministic algorithms, being a matter of minutes.

4 Observations and Conclusions

4.1 Choosing a fitness function

From the information presented we can clearly conclude that there is no perfect fitness function and that some problems can't be wholly solved with just a fitness function, slight modifications of population size, generations, mutation chance or crossover chance being needed. The minimum we want to determine also affects how difficult the fitness function is to choose. For functions such as DeJong's or Rastrigin's we have chosen simple fitness functions because the global minima is 0. Therefore a good result is a result closest to 0, and simply raising the value to a negative power (in our case, -10) should provide a good enough basis of comparison between different results.

There are also functions like Schwefel's, for which the global minima is dependent on the number of dimensions and is not such a common value (such as 0). Another problem with Schwefel's is the fact that its values are not always negative or positive, they alternate, this being a challenge in itself. A more complex function is needed when we deal with such a minima.

When choosing a fitness function we must also take into consideration how strict we want it to be. If the function is too strict then we risk getting stuck in a local minima. If it isn't strict enough then we might allow worse chromosome to survive into future generations, worsening the final result. A good example of a function for which a very strict fitness function is the best one is DeJong's. The fact that it has only one minima guarantees that no matter how strict a fitness function we choose, the algorithm will never get stuck in a local minima. Unlike DeJong's, Rastrigin's function has many local minima, making a strict fitness function less useful. In the algorithm we compensated for the strictness of Rastrigin's fitness function through an increase in population.

4.2 Population size and Generations

Figure 3, **Figure 5** and **Figure 8** best illustrate that an increase in population size result in a better final solution. This is because, by increasing the size of the population, we increase the search area, thus increasing the chance of finding the global minima or at least a better local minima. The main draw of this practice is that the higher the population size the more time it takes for the algorithm to finish, and we eventually get to a point where it is no longer justifiable to increase the population.

Increasing the number of generations can be a double edged sword. On one hand this raises the number of times we evaluate and change the population towards a better one, which in the end is what this algorithm is meant to do. On the other hand with every generation we give the chromosomes an extra chance to mutate, which is not always good for the final result because we could mutate a good result, the global minima even, and lose it.

4.3 Mutation and Cross-Over

Mutation Chance and **Cross-Over Chance** are two constants that can greatly alter the behaviour of the algorithm. As stated before, mutation can be both good and bad for the final result. When a chromosome gets a mutation it could either not affect it that much, make it get out of a local minima and thus improve the chance of finding a better result or ruin a good minima. Therefore it's hard to find a good mutation chance, especially when combined with all the other factors. For the algorithm, where the mutation chance is different from 1% more tests have been made until we arrived at a good one. A slight change in the mutation chance can modify the results entirely, for better or for worse, and much testing is needed to choose the perfect mutation chance for a certain algorithm.

The cross-over chance is similar in many ways when it comes to its ability to alter the results. Changing it slightly doesn't usually shift the results completely, unlike the mutation chance, but it will alter them a bit. Finding a good cross-over chance is just as hard as finding a good mutation chance, and choosing both well is many times tougher.

4.4 Comparison with Hill Climbing and Simulated Annealing

Function	Algorithm	Expected Result	Actual Result	Average Time
Schwefel's	HCF	-12569.487	-10992.85	3142.548
	HCB		-11487.99	5708.184
	SA		-10904.62	96.407
	GA		**-12512.8**	**90.752**
Michalewicz's	HCF	-29.63	-26.84332	1887.912
	HCB		-27.38289	3288.474
	SA		-25.52226	**145.406**
	GA		**-28.67256**	280.217
Rastrigin's	HCF	0	33.92544	1160.72
	HCB		26.52137	2078.675
	SA		36.24808	**75.859**
	GA		**0.76919**	270.67

Table 13: Comparison between Hill Climbing, Simulated Annealing and Genetic Algorithms on size 30

In **Table 13** is presented a comparison between the results and the time elapsed of **Hill Climbing First-Improvement (HCF)**, **Hill Climbing Best-Improvement (HBF)**, **Simulated Annealing (SA)** and **Genetic Algorithms (GA)**. The first column represents the function, the second one the algorithm used, the third one is the actual global minima, the fourth one is the mean result on size 30 for the corresponding algorithm and the last column represents the time elapsed for that result.

We didn't compare the results on DeJong's function because they were almost always 0, and the only difference is the time difference, but it doesn't justify a comparison between the four algorithms on that function.

In every case the Genetic Algorithm provides way better results, and in Schwefel's case it even takes less time than any of the other algorithms. Simulated Annealing is faster for Michalewicz's and Rastrigin's functions, but it provides the worst mean results. We can see that for Schwefel's and Michalewicz's the difference between the genetic algorithm and the next best algorithm is pretty big, but for Rastrigin's it is immense, the other algorithms not being even close to the global minima while the genetic algorithm provides a very close result.

4.5 Conclusion

In conclusion, for a genetic algorithm to function properly and provide the best possible result, it is required to find a good combination of **population size**, **generations**, **mutation chance** and **cross-over chance**, with a well-suited **fitness function**.

Our results show that a genetic algorithm can provide values very close to the actual global minima of the four functions selected, whilst also providing a great time advantage. Compared to results of other non-deterministic algorithms, such as Hill Climbing or Simulated Annealing, they excel both at accuracy and efficiency, providing better results in less time.

References

[1] Information about genetic algorithms
Information and definitions on the subject of genetic algorithms. https://profs.info.uaic.ro/~eugennc/teaching/ga/#Notions04

[2] Information about genetic algorithms
Further information about genetic algorithms. https://profs.info.uaic.ro/~eugennc/teaching/ga/res/gaSlidesOld.pdf

[3] Genetic Algorithms
Information abour genetic algorithms. https://en.wikipedia.org/wiki/Genetic_algorithm

[4] Thread information
How to get thread id. https://en.cppreference.com/w/cpp/thread/get_id

[5] Using C++ vectors
How to create, manipulate and use vectors. https://en.cppreference.com/w/cpp/container/vector

[6] Inverse Trigonometric Functions
Information about inverse trigonometric functions. https://en.wikipedia.org/wiki/Inverse_trigonometric_functions

[7] Atan2 function
Information abour atan2 function and how to use it. https://www.medcalc.org/manual/atan2_function.php

[8] De Jong's function
Information about De Jong's function. http://www.geatbx.com/docu/fcnindex-01.html#P89_3085 De Jong's function image
Image for De Jong's function and further information. https://www.sfu.ca/~ssurjano/spheref.html

[9] Schwefel's function
Information about Schwefel's function. http://www.geatbx.com/docu/fcnindex-01.html#P150_6749 Schwefel's function image
Image for Schwefel's function and further information. https://esa.github.io/pagmo2/docs/cpp/problems/schwefel.html

[10] Rastrigin's function
Information about Rastrigin's function. http://www.geatbx.com/docu/fcnindex-01.html#P140_6155 Rastrigin's function image
Image for Rastrigin's function and further information. https://www.sfu.ca/~ssurjano/rastr.html

[11] Michalewicz's function
Information about Michalewicz's function. http://www.geatbx.com/docu/fcnindex-01.html#P204_10395 Michalewicz's function image
Image for Michalewicz's function and further information. https://www.sfu.ca/~ssurjano/michal.html

[12] Merging text files
Site that merges text files, used for easier processing of the results. https://www.filesmerge.com/merge-text-files

[13] Information aboud PgfPlots
Documentation for pgfplots, how to use them and examples. https://www.iro.umontreal.ca/~simardr/pgfplots.pdf

[14] LaTeXinformation
Mathematical symbols in LaTeX. https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols
Floats in LaTeX. https://www.overleaf.com/learn/latex/Positioning_of_Figures
Text formatting in LaTeX. https://www.overleaf.com/learn/latex/bold,_italics_and_underlining
Example of LaTeXcode. https://gitlab.com/eugennc/teaching/-/blob/master/GA/texample.tex
Example of LaTeXexported pdf. https://gitlab.com/eugennc/teaching/-/blob/master/GA/texample.pdf