Optimizations of the genetic algorithm for finding the minimum of benchmark functions

Aciocanesa Bianca-Ioana

December 13, 2023

1 Abstract

The following document contains results obtained by using a genetic algorithm designed to find the global minimum of four different benchmark functions: De Jong 1, Schwefel's, Rastrigin's, Michalewicz's. Using a sample size of 10, the tests were run for 3 different dimensions of each function (5,10 and 30) and the results were compared with those obtained by using the non-optimized version of the algorithm.

2 Introduction

A genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, it selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution - in this case, the global minimum of different benchmark functions.

This report delves into the realm of optimizing genetic algorithms for the specific task of finding the global minimum of benchmark functions. The benchmark functions chosen for this study include De Jong 1, Schwefel's, Rastrigin's, and Michalewicz's, which are commonly used to evaluate the performance of optimization algorithms. The primary objective of this research is to explore and implement optimizations to enhance the efficiency of the genetic algorithm. By comparing the results obtained from the optimized version with those from the non-optimized version, we aim to assess the impact of these optimizations on the algorithm's performance.

3 Methods

3.1 Genetic Algorithm

3.1.1 Inspiration

Genetic algorithms were proposed by John Holland in 1973 after many years of studying the idea of simulating evolution. These algorithms model genetic inheritance and the Darwinian struggle for survival. Alongside two other directions, evolutionary strategies, and evolutionary programming, they form the class of evolutionary algorithms.

3.1.2 Key components

Evolutionary algorithms use vocabulary borrowed from genetics:

- **Gene**: the atomic information in a candidate solution
- **Locus**: the position a gene occupies
- **Allele**: all possible values for a gene
- **Chromosome**: a candidate solution represented as a string of genes
- **Population**: a set of chromosomes that evolves through the application of genetic operators : **mutation** and **crossover**
- **Evolution**: simulated through a sequence of generations of a population of candidate solutions

3.1.3 Representation

The algorithm starts from a given number of candidate solutions randomly generated as bitstrings - this is the **starting population**. Each chromosome from the population is then **evaluated**, meaning that it is converted to the decimal representation and applied to the given function, and it is considered "better" if it returns a smaller value (since the purpose of the algorithm is to find the global minimum of that function). A new **generation** of a population is obtained by applying the genetic operators over the previous population, in order to find progressively better solutions as generations advance until reaching the maximum number of generations set.

For this algorithm, the following set of parameters has been chosen:

- population size = 200
- number of generations = 2000

3.1.4 Pseudocode

```
1. $t := 0$
2. generate the starting population $P(t)$
3. evaluate $P(t)$
4. while not stopping condition
    • $t := t + 1$
    • select $P(t)$ from $P(t-1)$
    • mutate $P(t)$
    • crossover $P(t)$
    • evaluate $P(t)$
```

3.1.5 Genetic operators

1. **MUTATION**
   Mutation serves as an **exploration mechanism**, allowing the algorithm to explore neighboring regions of the solution space. This randomness helps the genetic algorithm avoid getting stuck in local optima and facilitates the discovery of novel, potentially better solutions. For each individual from a given population, mutation is applied probabilistically to each gene in the following way: for each gene, a random number from 0 to 1 is generated and if it is smaller than the chosen **mutation probability/rate**, that gene is modified . Since a gene has 2 possible values (0 or 1), the mutation represents changing that value with the other possible one.

2. **CROSSOVER**
   Crossover, also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring and it serves as an **exploitation mechanism**. It is one way to stochastically generate new solutions from an existing population. We traverse all individuals in a generation, and with a predefined **probability of crossover**, we select them. Subsequently, two consecutive individuals from this selection undergo the crossover process as follows: a cutting point is randomly chosen, and from that point, the exchange of genes takes place, as illustrated in the figure below, resulting in two new individuals.

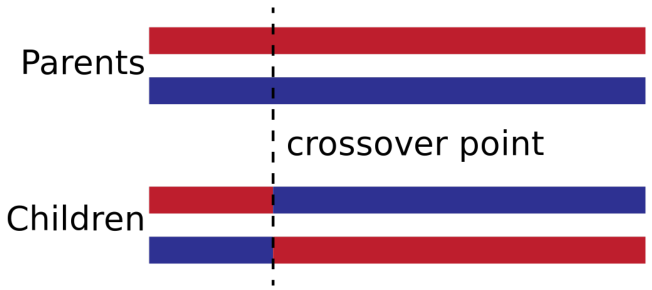


Figure 1: Crossover with a single cutting point [4]

3. **SELECTION**
   The purpose of selection is to choose the most adapted individuals for survival in the population, with the hope that their descendants will have higher fitness. In combination with variations of crossover and mutation operators, selection must maintain a balance between exploration and exploitation. High selection pressure leads to the creation of low diversity in the population composed of well-adapted but suboptimal individuals, resulting in a limitation of changes and, consequently, progress. On the other hand, low selection pressure slows down evolution. The algorithm used for this experiment contains the roulette wheel as selection method, that works in the following way:

   - Every individual from a generation is evaluated (the bitstring is converted to the decimal representation and applied to the given function) : $eval[i]$
   - The fitness function is calculated for every individual from a generation using the following formula:
     $fitness[i] = ((max - eval[i])/(max - min + 0.0001) + 1)^{10}$
     where max is the maximum evaluation from that population and min is the minimum evaluation
   - Then, the fitness proportion of each individual is calculated by dividing its fitness by the sum of the fitness values of all individuals in the population - this creates a probability distribution where higher fitness individuals have larger proportions
   - The roulette wheel is created - it is divided into segments, each corresponding to an individual in the population. The size of each segment is proportional to the individual's fitness proportion : the better the solution, the bigger the size of the segment, in order to create a larger probability for that element to be selected
   - A "random spin" of the roulette is performed: a random number is generated and the individual on the segment where the probability lands is selected for the next generation

3.2 Optimizations

There are various optimizations and strategies that can be applied to improve the performance of a genetic algorithm. Among the optimization techniques employed, the integration of **elitism** and **gray decoding** played a pivotal role, unleashing the full potential of the genetic algorithm to navigate complex solution spaces with increased precision and efficiency. In this regard, the improvements made to the algorithm are as described below:

- **GRAY DECODING**
  Gray decoding is vital in a genetic algorithm as it effectively addresses the Hamming cliff problem and ensures a seamless exploration of the solution space. The Hamming cliff problem arises when a slight alteration in the binary representation of a solution leads to a profound change in the corresponding phenotype. By guaranteeing that the decoded values bear similarity to their binary counterparts, Gray decoding plays a key role in fostering a diverse and representative gene pool within the population. Its ability to mitigate the Hamming cliff problem, preventing abrupt shifts in the phenotype due to minor changes in the genotype, results in a more fluid exploration of the solution space. This aligns well with genetic algorithm principles, emphasizing gradual transitions, diversity maintenance, and the reduction of premature convergence risks. .

- **ELITISM**
  Elitism in a genetic algorithm is a strategy that involves preserving a certain percentage of the best individuals (chromosomes or solutions) from one generation to the next without applying genetic operators like crossover and mutation to them. This ensures that the best-performing individuals in the population are directly carried over to the next generation, preventing the loss of their beneficial traits.

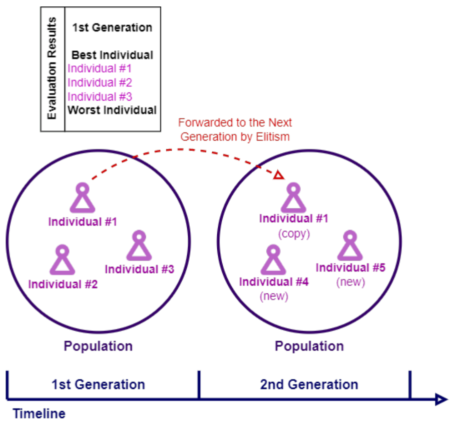


Figure 2: How elitism works [5]

In this regard, a reversed version of elitism has been implemented too, where, similar to elitism, where the top $x\%$ solutions directly advance to the next generation, the bottom $x\%$ solutions have a 50% chance of proceeding to the next generation. Otherwise, they are replaced in the new generation by other randomly generated bit strings. For this experiment, the elitism was 5, meaning that out of 200 individuals, only the best 5 of them were directly sent to the next generation without suffering crossover and mutation, and the worst 5 of them had 50% chance of survival.

4 Functions used for the experiment

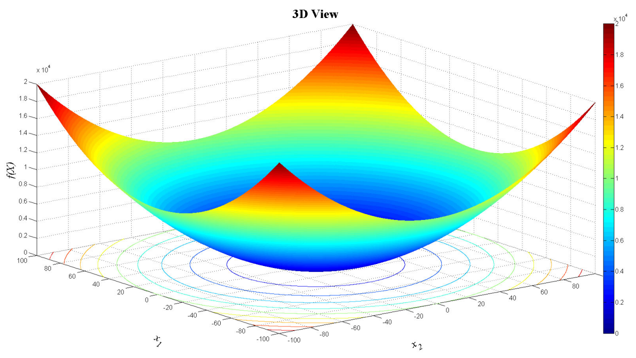
The algorithms were tested on four different functions, trying to find the best aproximation of their global minimum. The functions are presented below.

4.1 De Jong 1 function[6]

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

$$-5.12 \leq x_i \leq 5.12$$

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

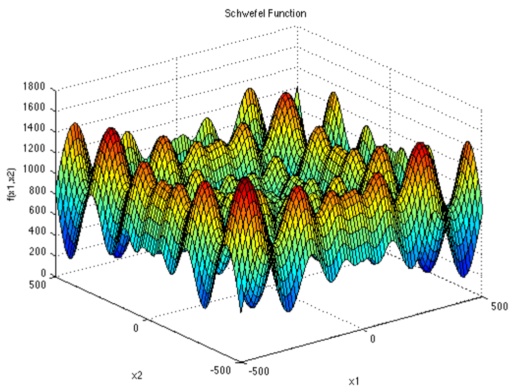


4.2 Schwefel's function[7]

$$f(x) = \sum_{i=1}^{n} -x_i \cdot sin(\sqrt{|x_i|})$$

$$-500 \leq x_i \leq 500$$

$$Global\ minimum : f(x) = -n \cdot 418.9829, x_i = 420.9687, i = 1 : n$$

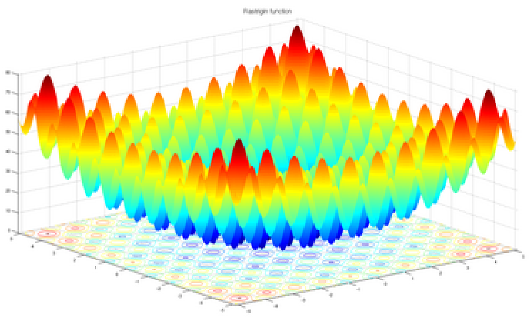


4.3 Rastrigin's function[8]

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

$$-5.12 \leq x_i \leq 5.12$$

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



4.4 Michalewicz's function[9]

$$f(x) = -\sum_{i=1}^{n} sin(x_i) \cdot (sin))^{2 \cdot m}$$

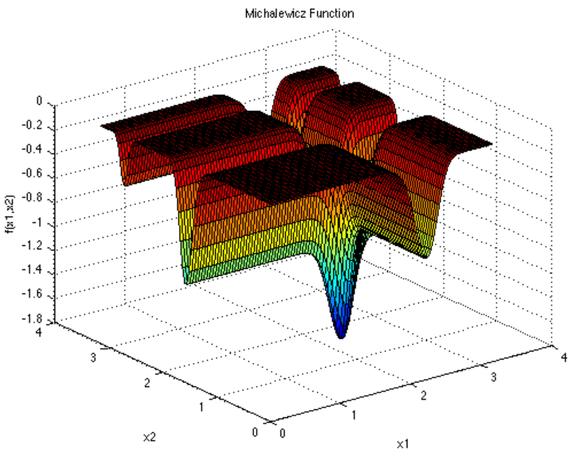
$$0 \leq x_i \leq \pi$$

$$Global\ minimum:$$

$$f(x) = -4.687(n = 5)$$

$$f(x) = -9.66(n = 10)$$

$$f(x) = -29.6308839(n = 30)$$



5 Experimental results

We will now compare the results obtained with the improved genetic algorithm versus the unimproved one, initially used, to see if it eventually reached the real minimum of the function or at least came closer to it. The mutation and crossover rates remained the same as in the previously used algorithm, with the only enhancements being the introduction of crossover with 4 cutting points instead of only one, gray encoding and elitism.

5.0.1 De Jong 1 function

Crossover probability = 0.5
Mutation probabilty = 0.001

	best result	average	worst result
Genetic Algorithm	5.97844e-05	0.000544151	0.00190047
Improved Genetic Algorithm	1.1921e-10	1.1921e-10	1.1921e-10

Table 1.1: De Jong 1 function with 5-dimensions

	best result	average	worst result
Genetic Algorithm	0.0068875	0.0234538	0.0124763
Improved Genetic Algorithm	9.77518e-09	3.803168e-08	1.23072e-07

Table 1.2: De Jong 1 function with 10-dimensions

	best result	average	worst result
Genetic Algorithm	0.175375	0.232216	0.30217
Improved Genetic Algorithm	0.00128778	0.001911884	0.00295812

Table 1.3: De Jong 1 function with 30-dimensions

5.0.2 Schwefel's function

Crossover probability = 0.6
Mutation probabilty = 0.01

	best result	average	worst result
Genetic Algorithm	-2094.88	-2094.78	-2094.69
Improved Genetic Algorithm	-2094.91	-2094.91	-2094.91

Table 2.1: Schwefel function with 5-dimensions

	best result	average	worst result
Genetic Algorithm	-4188.25	-4186.11	-4184.42
Improved Genetic Algorithm	-4189.82	-4189.724	-4189.52

Table 2.2: Schwefel function with 10-dimensions

	best result	average	worst result
Genetic Algorithm	-12416.7	-12377.36	-12285.1
Improved Genetic Algorithm	-12567.5	-12566.4	-12564.4

Table 2.3: Schwefel function with 30-dimensions

5.0.3 Rastrigin's function

Crossover probability = 0.9
Mutation probabilty = 0.01

	best result	average	worst result
Genetic Algorithm	0.00121817	0.135304	1.31444
Improved Genetic Algorithm	2.36502e-08	2.36502e-08	2.36502e-08

Table 3.1: Rastrigin function with 5-dimensions

	best result	average	worst result
Genetic Algorithm	0.143358	1.05831	2.42877
Improved Genetic Algorithm	8.51408e-08	5.204808e-08	4.73004e-08

Table 3.2: Rastrigin function with 10-dimensions

	best result	average	worst result
Genetic Algorithm	17.0829	21.2733	24.749
Improved Genetic Algorithm	2.71438	6.7017124	11.2127

Table 3.3: Rastrigin function with 30-dimensions

5.1 Michalewicz's function

Crossover probability = 0.8
Mutation probabilty = 0.005

	best result	average	worst result
Genetic Algorithm	-3.79457	-3.71642	-3.69165
Improved Genetic Algorithm	-3.70465	-3.70465	-3.70465

Table 4.1: Michalewicz function with 5-dimensions

	best result	average	worst result
Genetic Algorithm	-8.58271	-8.51287	-8.28613
Improved Genetic Algorithm	-8.66598	-8.647562	-8.58369

Table 4.2: Michalewicz function with 10-dimensions

	best result	average	worst result
Genetic Algorithm	-27.7996	-27.32755	-27.3735
Improved Genetic Algorithm	-28.0119	-27.66556	-27.3353

Table 4.3: Michalewicz function with 30-dimensions

6 Comparison and analysis of the results

To make a comparison between the non-optimized and the optimized algorithm used for calculating the minimum, we need to take into consideration the accuracy of the results, depending on the complexity of the functions mentioned above and the real minimum of the given functions. As a first conclusion, it can be stated that in all cases, the improved version of the algorithm provided superior results, with significant improvements observed in some cases. This indicates that the techniques employed indeed had a tangible enhancing effect on the algorithm's performance.

The De Jong 1 is a simpler function; hence, both the optimized and unoptimized versions provided satisfactory results. However, the improvement is significant in the case of the enhanced algorithm, particularly for smaller dimensions of the function (for 5 dimensions, from a minimum of 5.97844e-05 to a minimum of 1.1921e-10 - a result more than 10^5 times smaller). Additionally, for the 30-dimensional variant, the results were approximately 100 times smaller - still a considerable improvement.

The algorithm applied to the **Schwefel function** yielded excellent results even from the unoptimized version - the surprise lies in the fact that, with the improved algorithm, for the 5-dimensional version, the real minimum of the function (-2094.91) was found, and for the 10-dimensional version, the best result obtained is also the actual minimum of the function (-4189.82). For the 30-dimensional function, despite a much larger search space, the difference between the result and the real minimum is only 3 units.

The Rastrigin function is a much more challenging function with numerous local minima that can trap the algorithm. However, with the improvements made, the results were over 8 times better, approaching nearly zero for both the 5 and 10-dimensional versions. For the 30-dimensional variant, further enhancements could be introduced, considering that the average of the obtained results is only 6.7017124 (still a considerable difference compared to the unimproved algorithm, where the average was approximately 21).

Although there has been a significant improvement in results so far, for **the Michalewicz function**, the enhanced algorithm, while producing slightly better results, did not manage to significantly approach the local minimum of the function.

7 Conclusion

Since it combines exploration and exploitation through mechanisms like crossover and mutation, the genetic algorithm is an efficient technique for finding the minimum of a function. This balance allows the genetic algorithm to explore a diverse solution space (exploration) while exploiting promising regions to refine solutions (exploitation). This study compared the standard genetic algorithm with its improved version, demonstrating the versatility of such an algorithm. In summary, the optimized genetic algorithm, incorporating enhancements such as gray decoding, elitism, and multi-point crossover, demonstrated notable improvements across benchmark functions. From efficiently handling simpler functions like De Jong 1 to surprising success in discovering real minima for Schwefel's function, the enhancements showcased adaptability and efficiency. The Rastrigin function, known for its complexity, witnessed substantial gains, with results approaching zero in lower dimensions. These results highlight the effectiveness of the optimizations, underscoring their role in improving the genetic algorithm's performance across diverse optimization tasks.

References

[1] Course's site (2023):"Concepts - Genetic Algorithms" https://profs.info.uaic.ro/~eugennc/teaching/ga/

[2] Wikipedia contributors, "Genetic algorithm," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1186064335(accessed December 5, 2023).

[3] ”An analysis of Gray versus binary encoding in genetic search”, Information Sciences, volume 156, pages 253-269, `https://www.sciencedirect.com/science/article/pii/S0020025503001786`

[4] Wikipedia contributors, ”Selection (genetic algorithm),” Wikipedia, The Free Encyclopedia, `https://en.wikipedia.org/w/index.php?title=Selection_(genetic_algorithm)&oldid=1178932579` (accessed December 5, 2023).

[5] ”Elitism in evolutionary algorithms”, Vinicius Fulber-Garcia on `https://www.baeldung.com/cs/elitism-in-evolutionary-algorithms`

[6] De Jong’s function 1 informations `http://www.geatbx.com/docu/fcnindex-01.html`

[7] Schwefel’s function informations `https://www.sfu.ca/~ssurjano/schwef.html`

[8] Rastrigin’s function informations `https://www.sfu.ca/~ssurjano/rastr.html`

[9] Michalewicz’s function informations `https://www.sfu.ca/~ssurjano/michal.html`