A Genetic Algorithm in Real Function Optimization. Performance Comparison between G.A, Hill Climbing & Simulated Annealing

Florin Eugen Rotaru, Ștefan Nastasiu

24 November 2022

Abstract

This paper starts with the presentation of a Genetic Algorithm variation used for numerical optimization. For this we provide the designs of all the common modules: Representation, Selection Strategy, Fitness Function, Mutation & Crossover methods, followed by all the necessary parameters regarding probabilities, sizes and termination conditions. Our algorithm variation relies on nonuniform mutation and crossover operators. We tested our algorithm on 4 test functions: Rastrigin's, Schwefel's, Michalewicz's and DeJong's 1, on dimensions $n = 5$, $n = 10$ and $n = 30$. We compared the results with the ones provided by Hill Climbing & Simulated Annealing, studied previously. The results show that the G.A is a more powerful optimization method that manages to perform well both in global search and in local search, leading to very good results (some of them better than H.C on best improvement), in a runtime far shorter.

1 Introduction

A Genetic Algorithm (G.A) is an *iterative global search* procedure that aims to find the optimum of the function used for evaluation. The algorithm works with an entire population of solutions in parallel, which are called ***chromosomes***, and are distributed over the entire search space. The dynamics of the search process is based on combinining ($crossing - over$) and modifying $mutating$) the chromosomes, with the goal of finding the optimum combination, (i.e., the one that presents the best performance). It is generally admitted that a new generation consists of better and more fit individuals. As the generations succeed each other, the individuals tend to evolve towards the *global optimum* of the function.

A new Generation from an old one:

- ***Evaluation***

  The G.A analyzes the performance of the population, by computing the $fitness\ function$ value for each chromosome.

- ***Selection***

  The G.A selects the individuals of the population $P(t)$ based on their performance. The selected individuals will form an intermediate population P'. Chromosomes of P' will be the parents of the new generation $P(t+1)$.

- ***Recombination and modification***

  The G.A recombines and modifies the individuals of P'. For this purpose, several genetic operators may be used, such as *crossover* ($gene\ transfer\ between\ chromosomes$), *mutation*, *inversion*, *etc.*

The parameteres of a G.A are the operators and the probabilities associated to them(the probability that an operator will be applied to an individual). The probabilities can be constant, or dynamic (adaptive).

2 The Problem

We consider the problem of optimizing real functions, with one ore more real variables. In our case, optimizing the functions means finding the global minimum.

3 Our Approach

3.1 Representation

All the chromosomes are internally represented as bitstrings that store the function arguments.[1] Deciding which chromosome is more fit implies converting the bits to Real $\mathcal{R}$ numbers and evaluating the fitness function. A population is a set of chromosomes.

3.2 Fitness Evaluation

As we are facing a classical Numerical Optimization problem, the fitness can be evaluated in the following way: let $c1$ and $c2$ be two individuals and let J be the function we want to minimize. The chromosome $c1$ is considered to be more fit than $c2$ if and only if $J(c1) < J(c2)$.

3.3 Parent Selection

Here we will use a selection method called *Ranking Selection* [2]. It preserves a constant selection pressure by sorting the population on the basis of fitness, and

then allocating selection probabilities to individuals according to their rank, rather than according to their actual fitness values. Conventionally, we set the $Rank = 0$ for the most fit individual, and $Rank = N$ for the last individual in the sorted list, where N is the number of chromosomes in a population.

The probability p_i of selecting the individual i is computed with the following formula:[3]

$$p_i = \frac{1}{N}\left(q - \frac{2(r_i - 1)(q - 1)}{N - 1}\right)$$

With the selection pressure $q \in [1; 2]$ defined as the average number of descendants of the most fit individual and r_i - the rank. In order to implement the Selection, we have simulated a Roulette-Wheel [2] in which the probabilities are computed using the formula above.

Obs : in order the set the ranks, we will sort the Population in ascending order with respect to the evaluated function value.

3.4 Crossover

Not every selected chromosome will produce offspring, only some of them. The probability P_c of a chromosome to crossover with another is constant.

We use the 70 *to* 30 *crossover* method, which works in the following way: In order to perform the crossover, we randomly take two chromosomes which have been previously selected. Let $c1$ be the better one, and c2 the other one. The offspring $c3$ and $c4$ will replace them in the next generation. They are both obtained similarly:

for $i = 1$ *to chromosome_length* **do**
  /* c1 is better than c2*/
  generate random $r = [0; 1]$
  if $r \leq 0.7$ **then**
    $offspring[i] \leftarrow c1[i]$
  else
    $offspring[i] \leftarrow c2[i]$
  end if
end for
return $offspring$

This crossover method gives priority to the more fit individuals, aiming to "save good genes" by sending them to the next generation.

3.5 Mutation

Mutation adds to the diversity of a Population. It is a probabilistic operator, that takes place with the *mutation probability* P_m, where P_m is of order 10^{-3}. The mutation operator for a bitstring is quite obvious: a bit suffers a mutation means a bit is negated. Mutations on a population means mutating any bit, on any of its chromosomes. [3].

Similarly to the way Selection is done, we implement a *Rank Based Adaptive Mutation*, which uses the Rank of each chromosome to compute the mutation probability of any of its bits (genes): [4]

$$p_i = P_m \left(1 - \frac{r_i - 1}{N - 1}\right)$$

Where P_m is the initial and maximum probability of mutation, r_i- the Rank and $N-$ the population size.

Note that P_m will be the mutation probability of the chromosome of the highest rank.

Also, for any population $P(t)$, the probabilities will be the same with respect to the rank, regardless of the value of t.

4 Algorithm Parameters & Results

Representation	Binary Bitstrings
Solution Precision	$\epsilon = 10^{-5}$
Recombination	70 to 30 crossover
Recombination Probability	0.6
Mutation	Adaptive Mutation by Rank
Worst Rank Mutation Probability P_m	0.01
Parent Selection	Rank Based Selection
Parent Selection Pressure q	2
Population Size N	100
Number of Offspring	2
Initialisation	Random
Termination Condition	1000 Generations Processed
Return Value	The optimum solution throughout all of the Populations

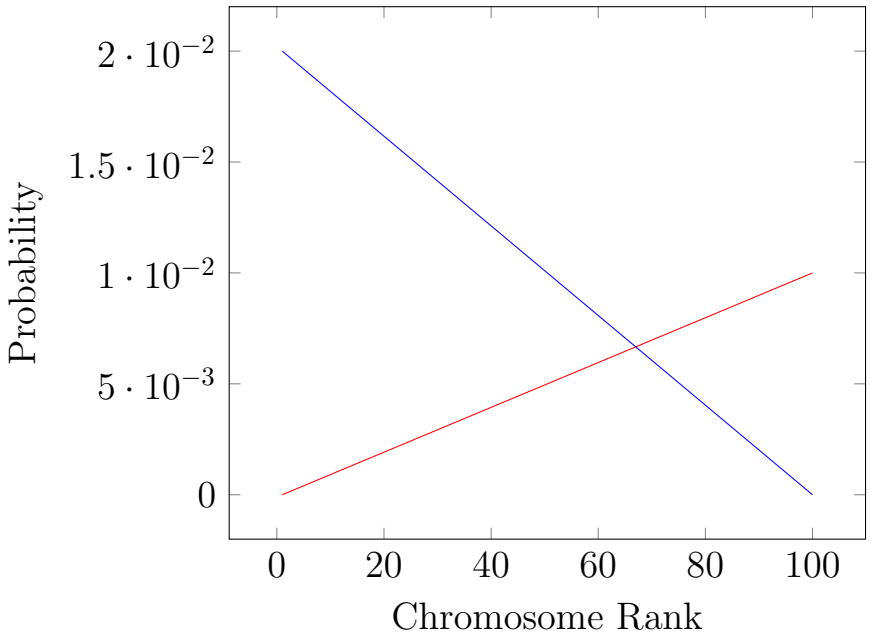


Fig. 1

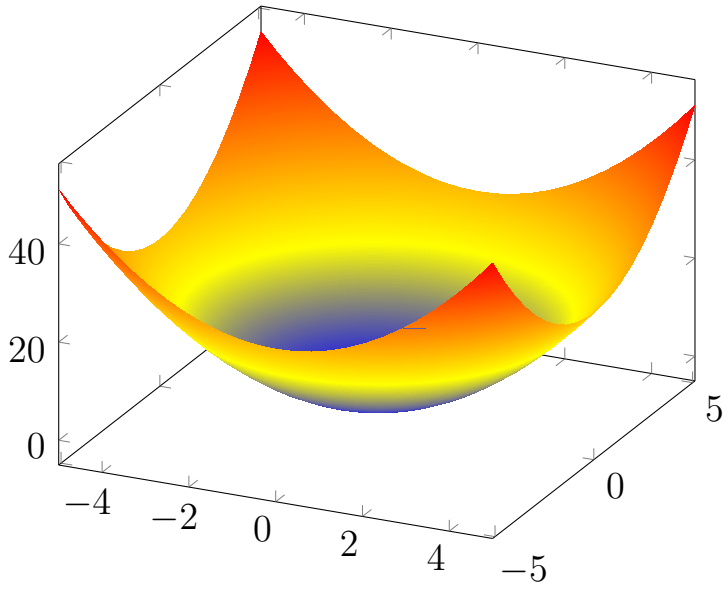
Mutation Probability by Rank Selection Probability by Rank

4.1 Sample Size

For each function, we executed the G.A 30 times, and registered the runtime. The results are presented below.

4.2 De Jong's 1

$f : [-5.12; 5.12] \to \mathbb{R}, f(\overline{x}) = \sum_{i=1}^{n} x_i^2$
global minimum $f(\overline{x}) = 0, \overline{x} = 0$

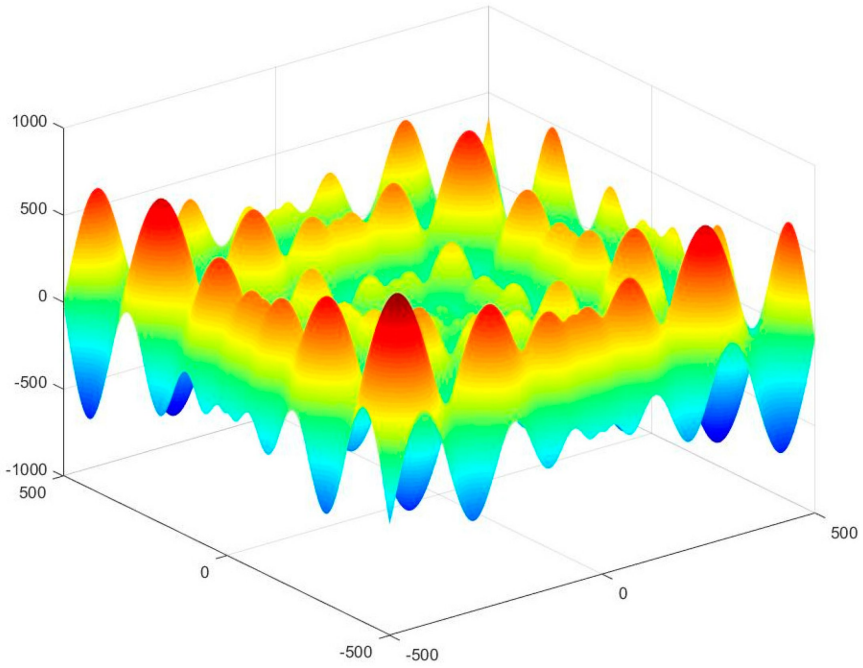


$\mu(min), \sigma(sd), \mu(t)$	n=5	n=10	n=30
Hill Climbing (BI)	$\mu = 1.08295 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 2.641min$	$\mu = 2.166 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 14.197min$	$\mu = 6.606 \cdot 10^{-10}$ $\sigma = 0$ $\mu(t) = 164.45min$
Simulated Annealing	$\mu = 0.001695$ $\sigma = 0.000836$ $\mu(t) = 0.11min$	$\mu = 0.00275697$ $\sigma = 0.0013362$ $\mu(t) = 0.305min$	$\mu = 0.0071$ $\sigma = 0.00202$ $\mu(t) = 2.12min$
Genetic Algorithm	$\mu = 0$ $\sigma = 0$ $\mu(t) = 1.2163sec$	$\mu = 0$ $\sigma = 0$ $\mu(t) = 2.5778sec$	$\mu = 0$ $\sigma = 0$ $\mu(t) = 8.7891sec$

4.3 Schwefel's

$f : [-500; 500] \to \mathbb{R}, f(\overline{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$

global minimum $f(\overline{x}) = -n \cdot 418.9829, \overline{x} = 418.9829$
$\{-2094.9145; -4189.829; -12569\}$

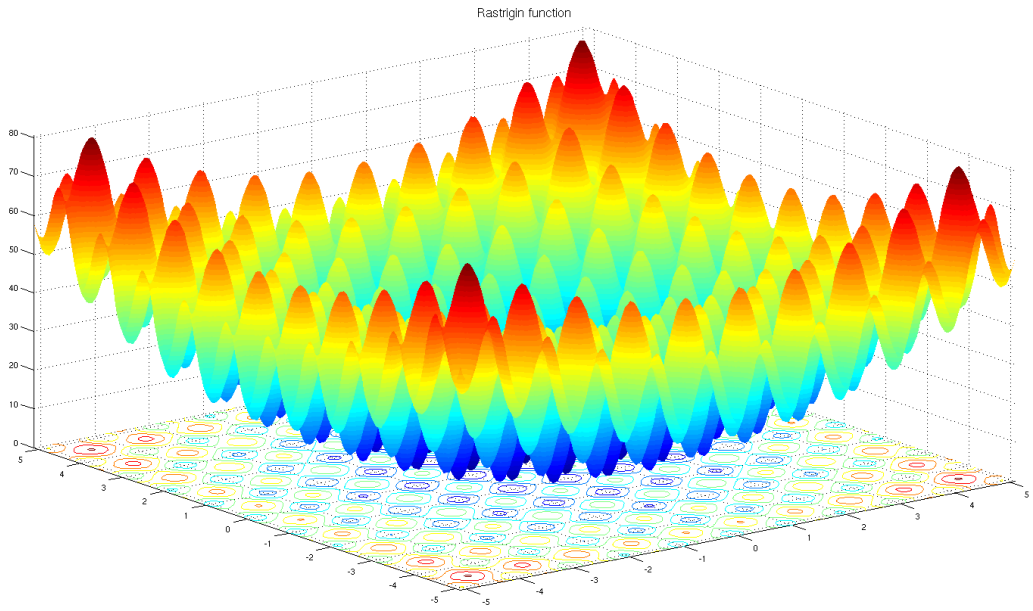


$\mu(min), \sigma(sd), \mu(t)$	n=5	n=10	n=30
Hill Climbing (BI)	$\mu = -2094.91$ $\sigma = 0$ $\mu(t) = 2.223min$	$\mu = -4161.98$ $\sigma = 39.877$ $\mu(t) = 21.94min$	$\mu = -11476.8$ $\sigma = 329.73$ $\mu(t) = 183.7min$
Simulated Annealing	$\mu = -1971.602$ $\sigma = 137$ $\mu(t) = 0.32min$	$\mu = -3775.443$ $\sigma = 166.22$ $\mu(t) = 1min$	$\mu = -11103.9$ $\sigma = 290.73$ $\mu(t) = 3.1366min$
Genetic Algorithm	$\mu = -2077.6372$ $\sigma = 40.0100$ $\mu(t) = 1.7454sec$	$\mu = -4101.7447$ $\sigma = 85.6797$ $\mu(t) = 3.6485sec$	$\mu = -11476.6134$ $\sigma = 256.3448$ $\mu(t) = 13.9571sec$

4.4 Rastrigin's

$f : [-5.12; 5.12] \to \mathbb{R}, f(\overline{x}) = 10 \cdot n + \sum_{i=1}^{n} \left[x_i^2 - 10 \cdot cos(2\pi x_i)\right]$

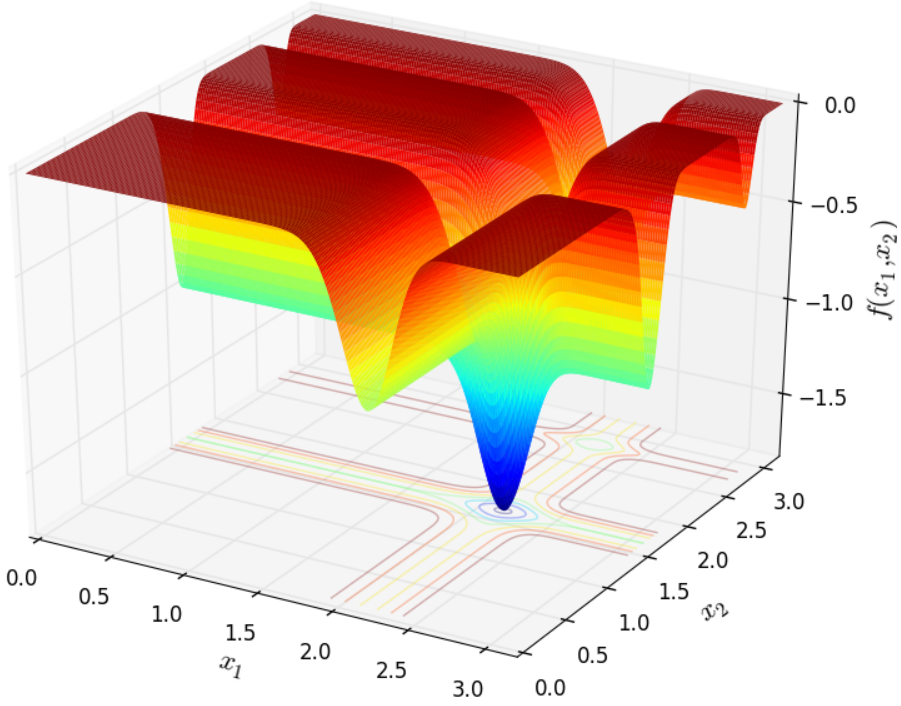
global minimum $f(\overline{x}) = 0, \overline{x} = 0$



$\mu(min), \sigma(sd), \mu(t)$	n=5	n=10	n=30
Hill Climbing (BI)	$\mu = 3.8147 \cdot 10^{-7}$ $\sigma = 0.114441 \cdot 10^{-5}$ $\mu(t) = 1.875min$	$\mu = 2.25937$ $\sigma = 0.66687$ $\mu(t) = 8.227min$	$\mu = 21.0924$ $\sigma = 10.28$ $\mu(t) = 228.3min$
Simulated Annealing	$\mu = 5.8838$ $\sigma = 3.9918$ $\mu(t) = 0.2min$	$\mu = 13.02$ $\sigma = 5.0639$ $\mu(t) = 0.45min$	$\mu = 39.6733$ $\sigma = 4.871$ $\mu(t) = 3.03min$
Genetic Algorithm	$\mu = 2.9111$ $\sigma = 2.44104$ $\mu(t) = 1.3271sec$	$\mu = 6.5613$ $\sigma = 3.5045$ $\mu(t) = 2.5968sec$	$\mu = 35.4190$ $\sigma = 8.4294$ $\mu(t) = 9.6633sec$

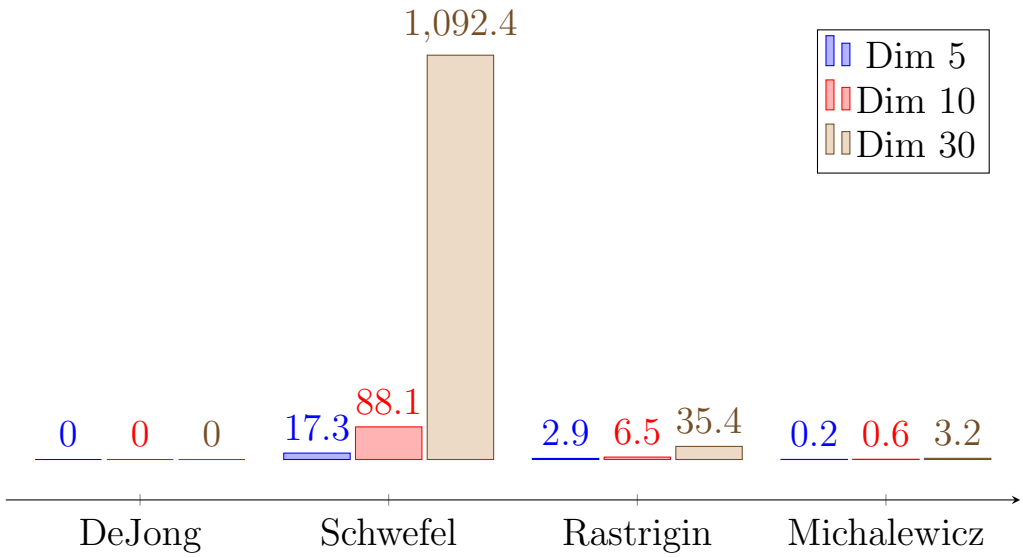
4.5 Michalewicz's

$f : [0; \pi] \to \mathbb{R}$, $f(\overline{x}) = -\sum_{i=1}^{n} \sin(x_i) \cdot (\sin(\frac{ix^2}{\pi}))^{2 \cdot 10}$
global minimum: $\{-4.687; -9.66; -29.630\}$



$\mu(min), \sigma(sd), \mu(t)$	n=5	n=10	n=30
Hill Climbing (BI)	$\mu = -3.69886$ $\sigma = 0$ $\mu(t) = 1.0722min$	$\mu = -8.567$ $\sigma = 0.0331$ $\mu(t) = 6.75min$	$\mu = -26.4234$ $\sigma = 1.44$ $\mu(t) = 134.97min$
Simulated Annealing	$\mu = -3.62532$ $\sigma = 0.0886$ $\mu(t) = 0.09min$	$\mu = -8.052692$ $\sigma = 0.2323$ $\mu(t) = 0.2min$	$\mu = -25.788$ $\sigma = 0.3277$ $\mu(t) = 1.26min$
Genetic Algorithm	$\mu = -4.4886$ $\sigma = 0.1334$ $\mu(t) = 1.1002sec$	$\mu = -9.0734$ $\sigma = 0.2398$ $\mu(t) = 2.3051sec$	$\mu = -26.3693$ $\sigma = 0.7296$ $\mu(t) = 8.7885sec$

GA: Distance from the global minimum



5 Conclusion

The enormous difference between the Genetic Algorithm and other Metaheuristic methods (such as Hill Climbing and Simulated Annealing) is the **runtime**. For example, on 30 dimensions, it takes the G.A around 10 *seconds* to do what Simulated Annealing does in several minutes and Hill Climbing in several hours.

Taking a closer look at the results themselves, we observe that G.A manages to find the solutions even better, in some places, than H.C on Best Improvement(which surpasses other variations of H.C and is definitely more accurate than S.A).

5.1 Focus on De Jong's 1

De Jong's 1 is a unimodal function, so all of the methods manage to find the single local minimum easily, with 0 standard deviation . This is due to the fact that both H.C and S.A belong to the family of local search algorithms, and our G.A also converges to local optimum at the end-phase of the execution.

5.2 Focus on Schwefel's

On dimensions $n = 5$ and $n = 10$, H.C performs slightly better than G.A. It also provides solutions that are considerably more uniform. The standard deviations increase as the dimension increases, S.A still having the highest std. dev, with the worst results.

5.3 Focus on Rastrigin's

On all dimensions, H.C beats G.A. The biggest difference is seen on dimension $n = 5$. The reason for G.A's poor performance might be the fact that it is designed for global search, and the algorithm did not manage to converge enough throughout the 1000 generations processed, that is, as our parameters do not depend on the value of generation index, the last generations are not more inclined to converge than the first generations. Note that the std. dev. of S.A on dimension $n = 30$ is better than on G.A. Even so, it still has the worst results. On top of all of this is the radical change in runtime.

5.4 Focus on Michalewicz's

This and De Jong's 1 are the cases where G.A outperforms the other two algorithms, with the exception of dimension $n = 30$ where G.A and H.C are similar. Note a better std. dev. for G.A this time, similar to the one of S.A on dimension $n = 10$.

In conclusion, it is clear that the use of a G.A is desirable, because they enforce a reasonable compromise between exploring new solutions and exploiting of the solution space. This compromise is almost optimal [3]. Even if the presented results are not satisfactory, there are multiple ways it can be further optimized and

improved. For example: by increasing the population size, increasing the generations number (see termination condition), changing the parameters, strategies, probabilites, adding new (adaptive) operators, Gray coding, etc.

References

[1] Eugen Croitoru UAIC-FII
`https://profs.info.uaic.ro/~eugennc/teaching/ga/`

[2] A.E. Eiben and J.E. Smith. Introduction to Evolutionary Computing Second Edition

[3] D. Dumitrescu. Algoritmi Genetici și Strategii Evolutive - Aplicații în Inteligența Artificială și în domenii conexe - 2000

[4] Avijit Basak. A Rank based Adaptive Mutation in Genetic Algorithms - August 2020
IBM India Pvt. Ltd

[5] Global minimum of Michalewicz function
https://www.researchgate.net/publication/278769271