# How a genetic algorithm change when using gray coding

Stamate Valentin 2B4

December 9, 2020

## Abstract

In this raport we will see how the results change when using gray coding for gene instead of binary.

## 1 Introduction

We already saw how good a genetic algorithm is when it comes on solving an optimization problem that cannot be solved using a deterministic algorithm because of the complexity. This time, we will try to see if we can make it better by introducing adaptive parameters and changing the codification of the gene from binary to gray.

### 1.1 Motivation

If the new algorithm gives better results this can have a huge inpact in any different application when genetic algorithm are used, because this adds more flexibility and every algorithm can be optimized to fit better in different enviroments.

## 2 Methods

The algorithm used is a genetic algorithm that has adaptive parameters. These are the crossover probability that increses linear with the number of generations 20% in the beginning and 100% at the end. Same for the mutation probability but decreases from 10% to 0.1%. Also the number of generation differs together with the input size. This will be shown in the result tables.

For every generation the candidates are picked in such a way that the best of them are selected for the next generation keeping the population number constant.

The genome is reprezented as a bitstring that contains all the components for a point, because the mutations can be made very easy and we work with data. Now, instead of using binary codification for gene, we will use gray coding to see what differs. For having

a slightly time improvement, every codification of a point is kept into an *int* with the use of binary operators(xor, or, etc).

The fitness for every candidate is calculated this way $P(i).fit = pow(10, 1/(-min(P) * cd + F(P(i).gene) + 100) \cdot 2000 + 1)$ where, $cd$ is 1 if $min(P)$ is negative, 0 otherwise $min(P)$ is the minimum value found within members, $P$ is the population and $F(gene)$ is the value of the function with the point as a bitstring. This way we keep a positive fitness even if the function takes negative numbers and we don't need to know the global minimum.

The stop condition is when the maximum number of generations is reached.

Another change is the introduction of a new mutation method described as follows: for 10 members of the population when the mutation happens, we will change the bit only if the fitness is better. This way, those members will only improve in the current generation.

# 3 Experimental Setup

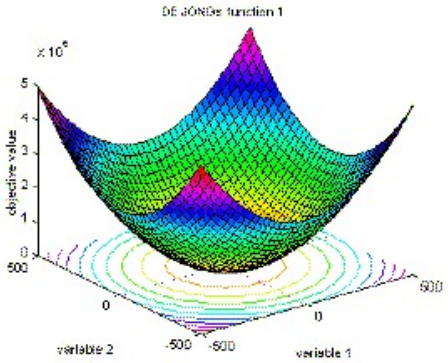
To have a better idea of how the algorithm is behaving we use different inputs and functions: as function dimention 2, 5, 10, 30 and functions: **DeJong's function**, **Schwefel's function**, **Rastrigin's function** and **Michalewicz's function**. Each function have a different number of global minimums. The precision for the experiment is 3 so $\varepsilon = 0.001$. The number of iterations is the same number as the stop condition witch is 500, 1000, 5000 and 7000 iterations/generations and the sample size is 30.

CPU : Intel i5 - 8265U with 4 phisical and 8 virtual cores.

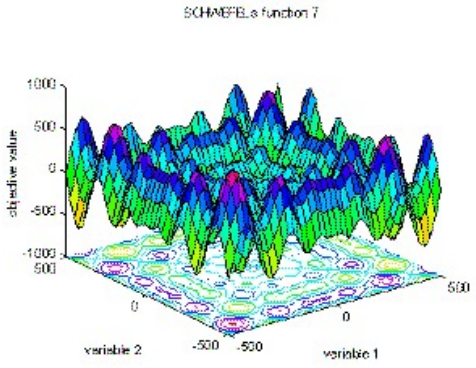
Software: Visual Studio Code

Programming Language: C++ 11

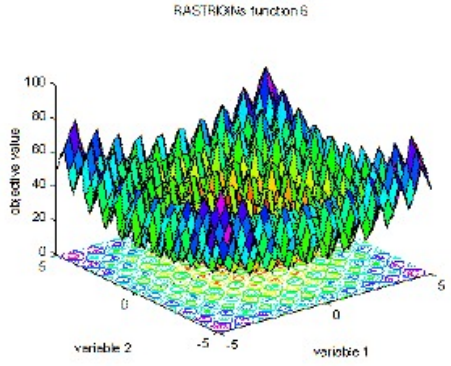
The reprezentation of the functions have two dimensions and below evey image is the funtion definition, the interval and the global minimum.



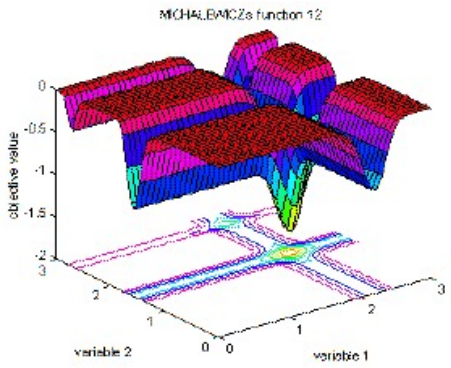

De Jong's function: $f_n(x) = \sum_{i=1}^{n} ix_i^2 \quad -5.12 \leq x_i \leq -5.12, \quad f(x) \geq 0$




Schwefel's function: $f_n(x) = \sum_{i=1}^{n} -x_i sin(\sqrt{|x_i|}), -500 \leq x_i \leq 500, f_n(x) \geq -n*418.9829$




Rastrigin's function: $f_n(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)), -5.12 \leq x_i \leq 5.12, f(x) \geq 0$




Michalewicz's function: $f_n(x) = -\sum_{i=1}^{n} sin(x_i)(sin(ix_i^2/\pi))^2 0, f_5(x) \geq -4.687, f_{10}(x) \geq -9.66$

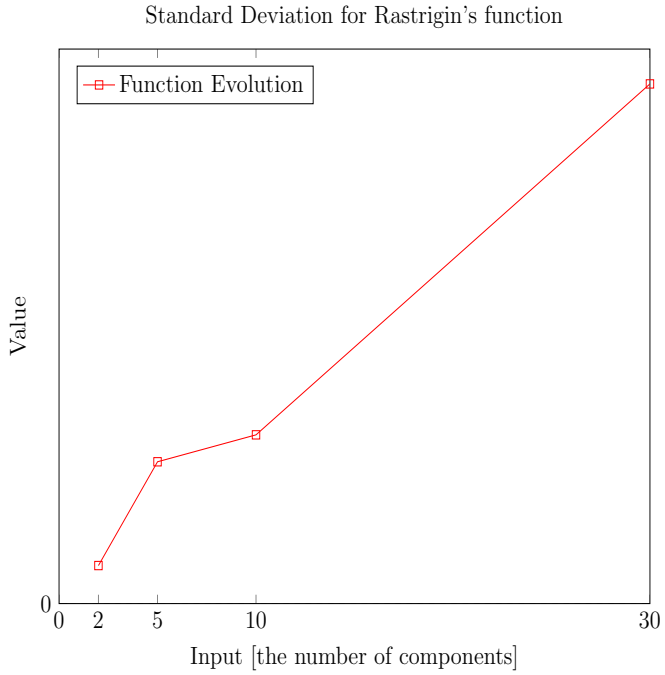
# 4 Results and Comparisons

| 2 dimensions — 500 generations | | | | |
|---|---|---|---|---|
| Function | Best Value | Mean | StDev | Duration |
| De Jong | 0 | 0 | 0 | 1m 39s |
| Schwefel | −837.966 | −837.9658 | 0.00042 | 3m 36s |
| Rastrigin | 0 | 0.29848 | 0.48061 | 2m 48s |
| Michalewicz | −0.80132 | −0.80132 | 0 | 2min 37s |

| 5 dimensions — 1000 generations | | | | |
|---|---|---|---|---|
| Function | Best Value | Mean | StDev | Duration |
| De Jong | 0 | 0 | 0 | 15m 33s |
| Schwefel | −2094.91 | −2094.91 | 0 | 21m 25s |
| Rastrigin | 0 | 2.17882 | 1.79351 | 15m 38s |
| Michalewicz | −3.69885 | −3.67422 | 0.05153 | 15m 18s |

| 10 dimensions — 5000 generations | | | | |
|---|---|---|---|---|
| Function | Best Value | Mean | StDev | Duration |
| De Jong | 0 | 0 | 0 | 2h 12m |
| Schwefel | −4189.83 | −4166.142 | 49.93869 | 3h 3m |
| Rastrigin | 0 | 3.78084 | 2.13909 | 2h 15m |
| Michalewicz | −8.53593 | −6.63260 | 5.376 | 2h 10m |

| 30 dimensions — 7000 generations | | | | |
|---|---|---|---|---|
| Function | Best Value | Mean | StDev | Duration |
| De Jong | 0 | 0 | 0 | 9h 31m |
| Schwefel | −12332.6 | −12010.84 | 195.4222 | 14h 6m |
| Rastrigin | 14.9246 | 25.27198 | 6.5697 | 10h 39m |
| Michalewicz | −26.8394 | −26.22901 | 0.49174 | 9h 33min |




Let's compare this algorithm with the standard one, and also without adaptive parameters.

| Rastrigin's function with 5 dimensions — 1000 generations | | | |
|---|---|---|---|
| Type | Best Value | Mean | StDev |
| *current* | 0 | 2.17882 | 1.79351 |
| *without adaptive par.* | 0.0077 | 0.05781 | 0.0337 |
| *second mutation* | 0 | 7.16436 | 4.93696 |
| *old approach* | 0.81863 | 7.84385 | 3.28231 |

The third result is obtain with the second mutation applied to all members. More than that, the number of dimensions is 30 for that row.

# 5 Conclusions

We see that the genetic algorithm with adaptive parameters and gray coding gives better results than the standard algorithm with binary coding. The mutation probability and crossover variation gives more flexibility so every generic algorithm can be optimized to fit better in problems.

# References


[1] Return vector of pointers from function
https://stackoverflow.com/questions/39102028/how-to-return-vector-of-pointers-and-ow

[2] Constant size vector
https://stackoverflow.com/questions/11134497/constant-sized-vector

[3] Sort a vector of objects
https://stackoverflow.com/questions/1380463/sorting-a-vector-of-custom-objects

[4] Genetic algorithm site
https://profs.info.uaic.ro/~eugennc/teaching/ga/

[5] Coding Train GA playlist
https://www.youtube.com/watch?v=9zfeTw-uFCw&list=
PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV&ab_channel=TheCodingTrain

[6] Seminar 5 notes

[7] De Jong's function
http://www.geatbx.com/docu/fcnindex-01.html#P89_3085

[8] Schwefel's function
http://www.geatbx.com/docu/fcnindex-01.html#P150_6749

[9] Rastrigin's function
http://www.geatbx.com/docu/fcnindex-01.html#P140_6155

[10] Michalewicz's function
http://www.geatbx.com/docu/fcnindex-01.html#P204_10395

[11] Benefits Of Coding Gray
https://stackoverflow.com/questions/41245917/what-are-the-benefits-of-gray-code-in-e

[12] Converting Binary To Gray
https://www.youtube.com/watch?v=cbmh1DPPQyI

[13] Gray To Binary
https://www.geeksforgeeks.org/gray-to-binary-and-binary-to-gray-conversion/

[14] Second Homework Results