Hill-Climbing, Simulated Annealing and Genetic Algorithms: A Comparative Study

Sbârcea Ștefan-Vladimir

November 23, 2021

1 Abstract

Hill-climbing, simulated annealing and genetic algorithms are search techniques that can be applied to most combinatorial optimization problems. The three algorithms are used to find benchmark functions global minimum. The genetic algorithm is independently evaluated and optimized according to its parameters.

A comparative study of the algorithms is then carried out. The performance criteria considered are the quality of the solutions obtained and the search time used for several benchmarks.

2 Introduction

Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. [8]

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. [3]

Genetic Algorithms (GAs) are a class of probabilistic algorithms that are loosely based on biological evolution. [2]

In this paper we examine 4 functions in order to test specific parameters of GAs and to evaluate the differences between GAs, BIHC and SA.

The structure of this paper is as follows: Section 3 describes the algorithms used for the experiment. Section 4 describes the functions used in this paper. In Section 5 we present the experimental setup, we examine the average, best and worst values for different GA parameters and we compare the robustness, average and best values for GA, SA and BIHC. Finally we draw some general conclusions.

3 Methods

We implemented each algorithm in C++ according to the pseudocode and description presented in the following subsections.

3.1 Simulated Annealing

Overview

In the early 1980s three IBM researchers, Kirkpatrick, Gelatt and Vecchi, introduced the concepts of annealing in combinatorial optimization. These concepts are based on a strong analogy with the physical annealing of materials. This process involves bringing a solid to a low energy state after raising its temperature. It can be summarized by the following two steps:

- Bring the solid to a very high temperature until " melting" of the structure;
- Cooling the solid according to a very particular temperature decreasing scheme in order to reach a solid state of minimum energy.

In the liquid phase, the particles are distributed randomly. It is shown that the minimum-energy state is reached provided that the initial temperature is sufficiently high and the cooling time is sufficiently long. If this is not the case, the solid will be found in a metastable state with non-minimal energy; this is referred to as hardening, which consists in the sudden cooling of a solid. [1]

Pseudocode

Input: A function $f : [a, b]^n \to \mathbb{R}$ and the values of n, a and b where $n \geq 1, a \leq b$
Output: The minimum value found for the function f when the termination criteria is met.

```
function SIMULATED ANNEALING
    Initialize the temperature T;
    Select a candidate solution v_c at random;
    t := 0;
    repeat
        repeat
            Select at random v_n: a neighbor of v_c;
            if f(v_n) < f(v_c) then
                v_c := v_n;
            else if random[0,1) < exp(-|f(v_n) - f(v_c)|/T then
                v_c := v_n;
            end if
            t++;
        until t ≥ MAX
        T := T * coolingRate;
        t := t + 1;
    until T ≤ minT
    return f(v_c);
end function
```

Algorithm description

The candidate solution v_c and the neighbors v_n are represented as vectors of bits. The vector size is determined with the following formula: $\lceil \log_2 (10^p * (b - a)) \rceil * n$ where p = precision, b = upper boundary, a = lower boundary, n = function dimensions. The selection of v_n goes as follows: We negate s bits at random from v_c,

$$s = 1 + 2 \cdot \frac{vector.size()}{dimensions \cdot precision}$$

.

3.2 Iterated Hill Climbing

Overview

In numerical analysis, hill climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found. [7]

Pseudocode

Input: A function $f : [a, b]^n \to \mathbb{R}$ and the values of n, a and b where $n \geq 1, a \leq b$
Output: The minimum value found for the function f when the termination criteria is met.

```
function ITERATED HILL CLIMBING
    Initialize best;
    t := 0;
    repeat
        locat := FALSE;
        Select a candidate solution v_c at random;
        repeat
            v_n := SelectBestNeighbor(v_c);
            if f(v_n) < f(v_c) then
                v_c := v_n;
            else
                local := TRUE;
            end if
        until local
        t++;
        if f(v_c) < f(best) then
            best := v_c;
        end if
    until t = MAX
    return best;
end function
```

Algorithm description

The variable *best*, the candidate solution v_c and the neighbors v_n are represented as vectors of bits. The vector size is determined with the following formula: $\lceil \log_2 (10^p * (b - a)) \rceil * n$ where p = precision, b = upper boundary, a = lower boundary, n = function dimensions. The function $SelectBestNeighbor(v_c)$ selects the best neighbor it finds after searching v_c neighbors at hamming distance one. To decrease the running time of the algorithm at the cost of slightly worst solutions the $SelectBestNeighbor(v_c)$ function was designed to randomly select a number of neighbors (the neighborhood iterator is increased with a random value $s \in [0, 2 \cdot precision \cdot dimensions)$).

3.3 Genetic Algorithm

Overview

Genetic Algorithm raises a couple of important features. First it is a stochastic algorithm; randomness as an essential role in genetic algorithms. Both selection and reproduction needs random procedures. A second very important point is that genetic algorithms always consider a population of solutions. Keeping in memory more than a single solution at each iteration offers a lot of advantages. The algorithm can recombine different solutions to get better ones and so, it can use the benefits of assortment. It is also important to mention that GAs are not guaranteed to find the global optimum solution to a problem, they are satisfied with finding " acceptably good" solutions to the problem. [5]

Pseudocode

Input: A function $f : [a, b]^n \to \mathbb{R}$ and the values of n, a and b where $n \geq 1, a \leq b$
Output: The minimum value found for the function f when the termination criteria is met.

```
function GENETIC ALGORITHM
    Initialize population;
    Calculate population fitness;
    generation := 0; it := 0;
    best := f(population[0]);
    while generation ≤ MAX && it < MAX/10 do
        Select population;
        if best > f(population[0]) then
            best := f(population[0]);
            it := 0;
        end if
        Mutate population;
        Crossover population;
        Calculate population fitness;
        generation++; it++;
    end while
    return best;
end function
```

Algorithm description

The *population* is represented as vector of individuals and each individual is a vector of bits and has a fitness value. The individual size is determined with the following formula: $\lceil \log_2 (10^p * (b - a)) \rceil * n$ where p = precision, b = upper boundary, a = lower boundary, n = function dimensions. We initialize the population randomly and the fitness of the population will always be 1, only the individuals fitnesses are modified by the best and the worst individual value in the population:

$$\frac{f(best) - f(individual[i])}{f(best) - f(worst) + 0.000001},\ 0 \leq i < populationSize$$

The selection method we used is roulette wheel selection. The mutation method used is bit flipping mutation. To reduce the genetic drift we applied elitism strategy in mutation and selection. The crossover method used is single point crossover and the candidates were chosen in ascending order of their fitness (lower fitness meaning better individual) and crossed with one selected with roulette wheel selection. The variable *it* is used to stop the algorithm if the best individual doesn't change in a number of generations.

4 Set of benchmarking functions

For the study of the algorithms we used one unimodal function: De Jong's Function 1 and three multimodal functions: Rastrigin's Function, Michalewicz's Function and Schwefel's Function. The next subsections present a short description of the functions.

4.1 De Jong's Function 1

$$f(x) = \sum_{i=1}^{d} x_i^2, x_i \in [-5.12, 5.12]$$

Global minimum [4]: $f(x) = 0;\ x_i = 0,\ i = 1 : d.$

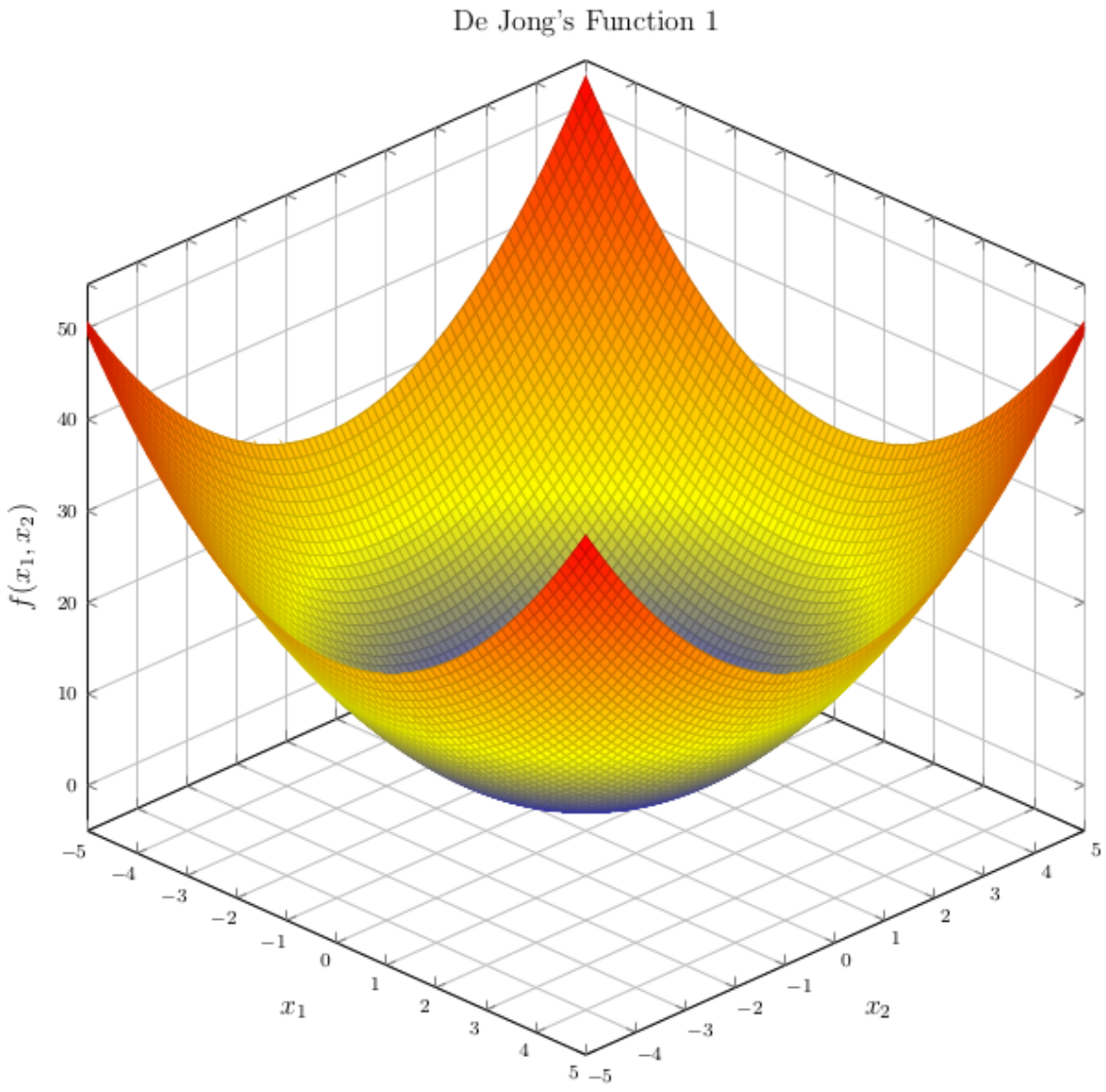

Figure 1: De Jong's Function 1. 2D representation.

4.2 Rastrigin's Function

$$f(x) = A \cdot d + \sum_{i=1}^{d} \left[x_i^2 - A \cdot cos(2\pi x_i) \right], A = 10, x_i \in [-5.12, 5.12]$$

Global minimum [4]: $f(x) = 0$; $x_i = 0$, $i = 1 : d$.

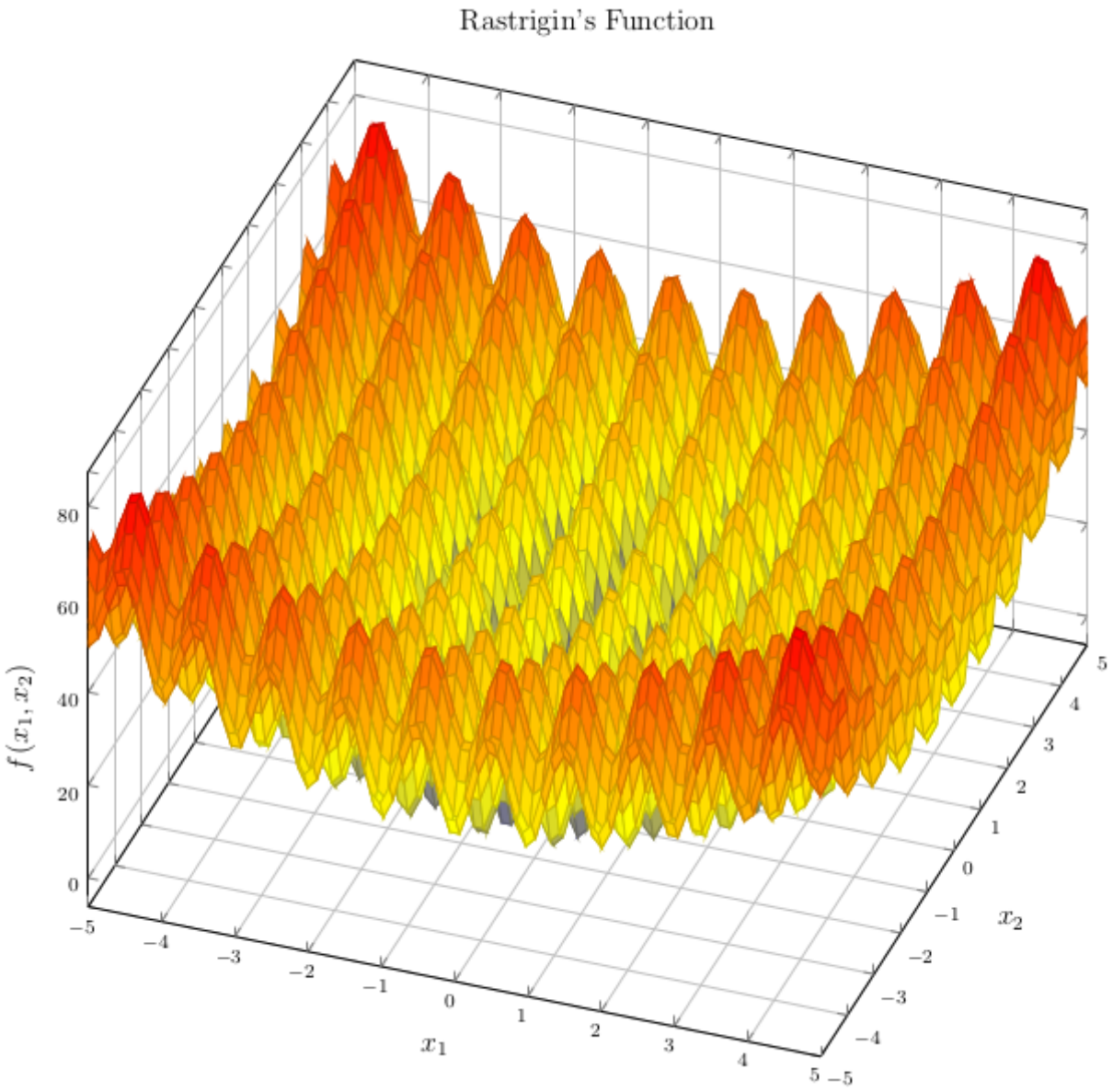

Figure 2: Rastrigin's Function. 2D representation.

4.3 Michalewicz's Function

$$f(x) = -\sum_{i=1}^{d} sin(x_i) \cdot sin^{2m}(\frac{ix_i^2}{\pi}), m = 10, x_i \in [0, \pi]$$

Global minimum [6]: $f(x) = -1.8013;\ (d = 2)\ x = (2.2, 1.57)$
$f(x) = -9.66;\ (d = 10)\ x_i = ???,\ i = 1 : d.$
$f(x) = -29.63;\ (d = 30)\ x_i = ???,\ i = 1 : d.$

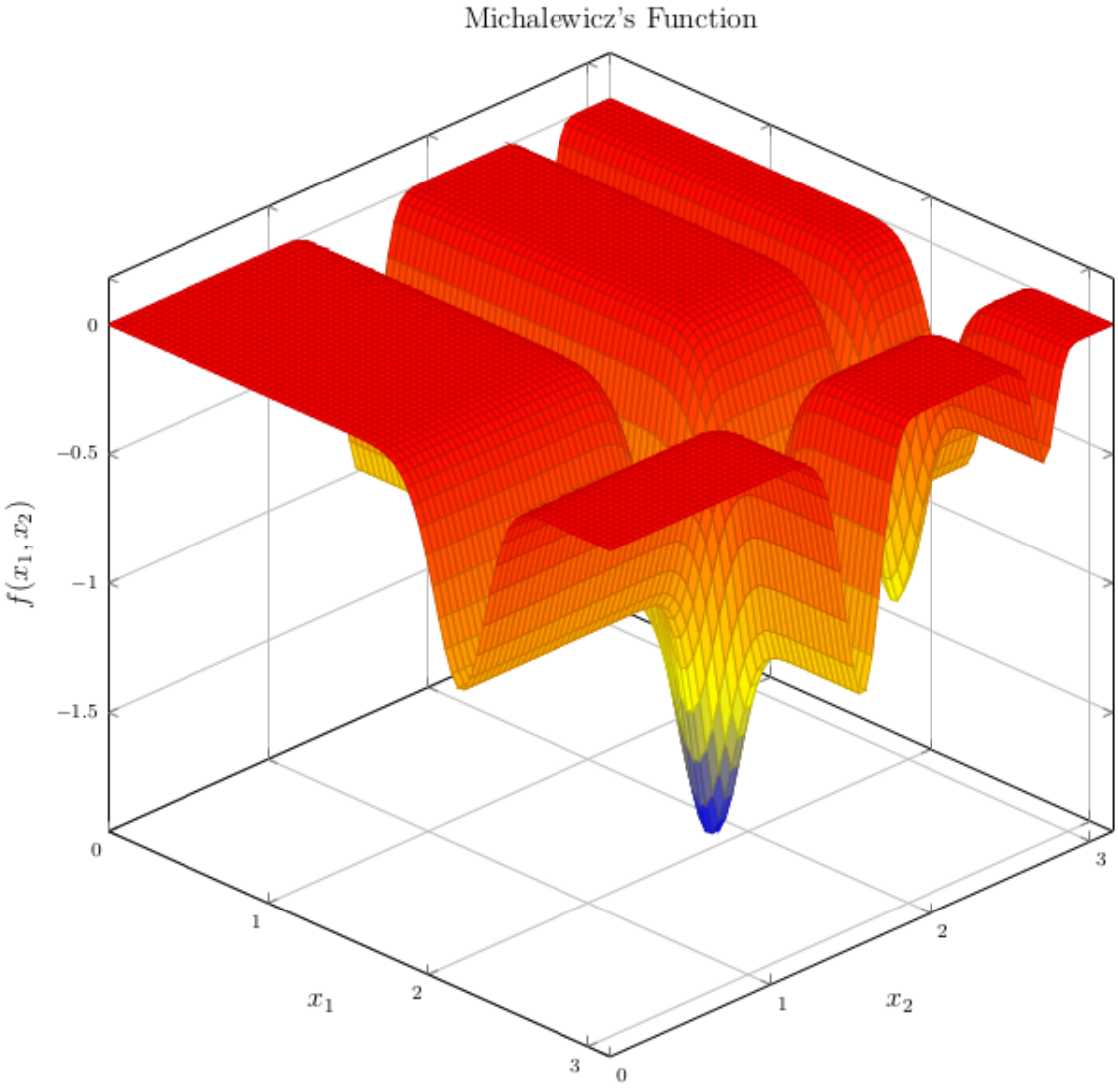

Figure 3: Michalewicz's Function. 2D representation.

4.4 Schwefel's Function

$$f(x) = \sum_{i=1}^{d} -x_i \cdot sin(\sqrt{|x_i|}), x_i \in [-500, 500]$$

Global minimum [4]: $f(x) = -d \cdot 418.9829$; $x_i = 420.9687$, $i = 1 : d$.

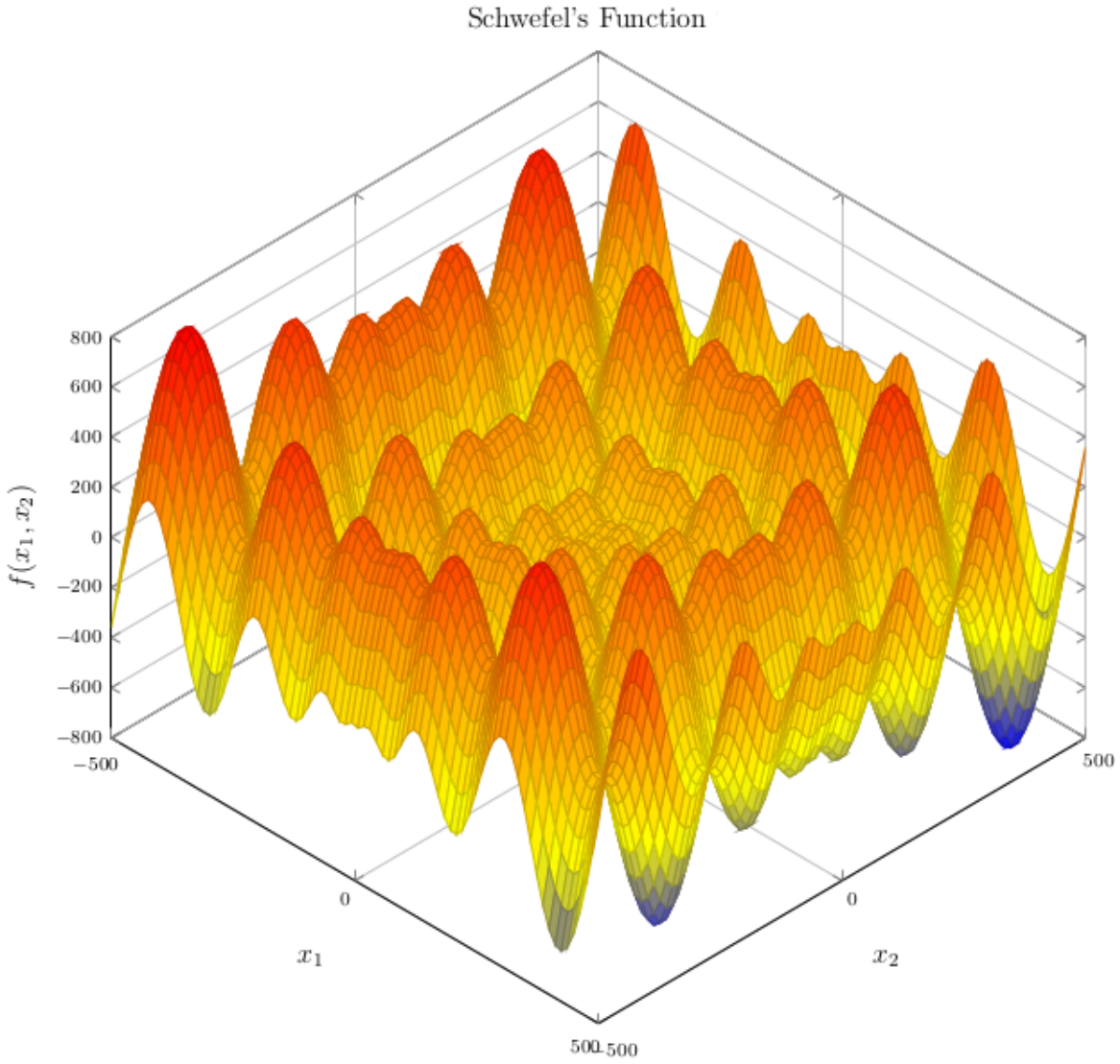

Figure 4: Schwefel's Function. 2D representation.

5 Experiment

All experiments were conducted on a 64-bit Ubuntu virtual machine with kernel version **Linux 5.11.0-40-generic**, running on a **2.6-GHz core i7-10750H CPU, 8 GB of RAM** and the source files were compiled using **g++ (Ubuntu 9.3.0-17ubuntu1 20.04) 9.3.0**.

5.1 Genetic Algorithm parameters optimization

The tests were run with $MAX = 10000$ number of generations and as we presented in pseudocode the algorithm can stop if the best individual is not substantially mutated after a number of generations (in this paper $\frac{MAX}{10}$ generations).
We ran 30 tests for each function on different dimensions 2,10 and 30 with precision 5 and different sets of parameters. We decided not to change the population size, the mutation percentage in population and the elites percentage. The population size is 100, the mutation percentage is 100 and the elites percentage is 10 for each test. First we tested the algorithm with different crossover rates 0.9, 0.95 and 1, with the fixed mutation rate 0.001 to maintain some diversity in the population. (tables 1-3).

5.1.1 Results 2 dimensions with different crossover rates (cRate)

cRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.9	De Jong's 1	2.13s	3.18s	6.86s	0	0	0
	Rastrigin's	2.28s	3.75s	5.53s	0	0	0
	Michalewicz's	2.07s	3.03s	5.53s	-1.8013	-1.8006	-1.80124
	Schwefel's	4.42s	8.7s	16.57s	-837.96577	-837.86211	-837.96223
0.95	De Jong's 1	2.01s	3.23s	7.17s	0	0	0
	Rastrigin's	2.28s	3.66s	7.13s	0	0.00002	0
	Michalewicz's	2.08s	3.39s	6.22s	-1.8013	-1.80037	-1.80119
	Schwefel's	2.99s	9.29s	16.22s	-837.96577	-837.86148	-837.94833
1	De Jong's 1	2.23s	3.44s	5.75s	0	0	0
	Rastrigin's	2.26s	3.42s	6.64s	0	1.00001	0.03333
	Michalewicz's	2.03s	3.11s	5.09s	-1.8013	-1.80071	-1.80121
	Schwefel's	3.53s	8.96s	15.42s	-837.96577	-837.86211	-837.95866

Table 1: Results cRate 2-dimensional version

5.1.2 Results 10 dimensions with different crossover rates (cRate)

cRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.9	De Jong's 1	14.02s	19.41s	26.07s	0	0.00003	0
	Rastrigin's	13.49s	22.58s	27.19s	0	1.98992	0.27187
	Michalewicz's	12.44s	36.02s	48.99s	-9.66013	-9.2045	-9.54356
	Schwefel's	32.36s	35.1s	40.26s	-4189.8277	-4152.19336	-4188.44495
0.95	De Jong's 1	13.71s	20.09s	26.79s	0	0	0
	Rastrigin's	13.41s	22.94s	28.17s	0	2.23204	0.28322
	Michalewicz's	19.6s	39.07s	45.54s	-9.66015	-9.20726	-9.55606
	Schwefel's	34s	35.26s	37.09s	-4189.82761	-4189.38074	-4189.65897
1	De Jong's 1	14s	20.65s	27.94s	0	0.00010	0
	Rastrigin's	13.5s	21.94s	27.10s	0	3.97984	0.56747
	Michalewicz's	20.37s	36.16s	45.49s	-9.66015	-9.32936	-9.56250
	Schwefel's	23.08s	33.44s	36.74s	-4189.82851	-4155.28043	-4188.54416

Table 2: Results cRate 10-dimensional version

5.1.3 Results 30 dimensions with different crossover rates (cRate)

cRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.9	De Jong's 1	68.96s	76.29s	82.7s	0	0.00176	0.00019
	Rastrigin's	67.98s	78.26s	85.71s	4.03355	18.05214	8.00606
	Michalewicz's	74.73s	90.47s	101.31s	-29.05968	-27.42699	-28.52293
	Schwefel's	85.43s	101.34s	115.66s	-12568.32264	-12144.37107	-12459.84400
0.95	De Jong's 1	67.37s	70.85s	75.57s	0	0.00640	0.00041
	Rastrigin's	60.48s	72.3s	79.63s	3.03084	13.68101	8.01771
	Michalewicz's	48.93s	83.85s	94.85s	-29.13489	-27.63084	-28.53505
	Schwefel's	85.69s	95.58s	107.08s	-12568.62735	-12172.71368	-12495.36944
1	De Jong's 1	66.11s	68.34s	74.56s	0	0.00642	0.00048
	Rastrigin's	51.01s	69.41s	75.77s	3.71037	16.90496	8.82972
	Michalewicz's	37.46s	81.19s	89.24s	-29.03575	-25.73873	-28.40302
	Schwefel's	86.61s	93.08s	101.61s	-12567.41088	-12337.54685	-12483.69780

Table 3: Results cRate 30-dimensional version

By analyzing the data from the tables 1-3 we can see that for the multimodal functions (Rastrigin's function, Michalewicz's function and Schwefel's function) there are significant differences for the 10 and 30 dimensions tests.

If we examine the 10 dimensional tests (table 2), with crossover rate 1 the solutions are worst on average for Rastrigin's function (figure 5) than the other two probabilities and for Schwefel's function (figure 7) we can see that 0.95 crossover rate makes the algorithm more robust.

For 30 dimensions we can see that with the crossover rate 0.95 the algorithm works better, it finds similar results or better result on average and the best solution found is better than the ones

found with crossover rate 0.9 or 1 (figures 8-10 and table 3).

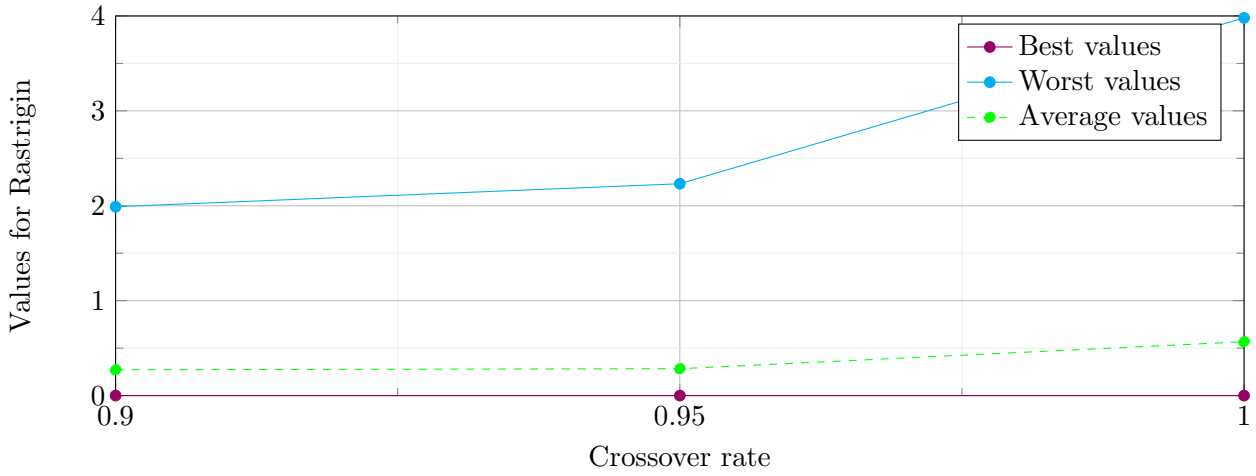

Figure 5: cRate study 10 dimensions Rastrigin's function

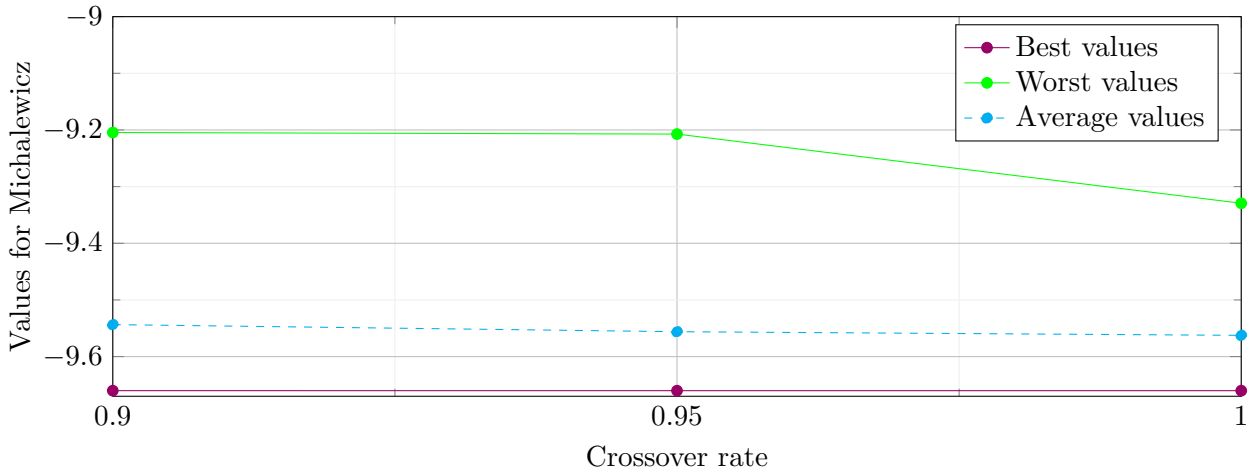

Figure 6: cRate study 10 dimensions Michalewicz's function

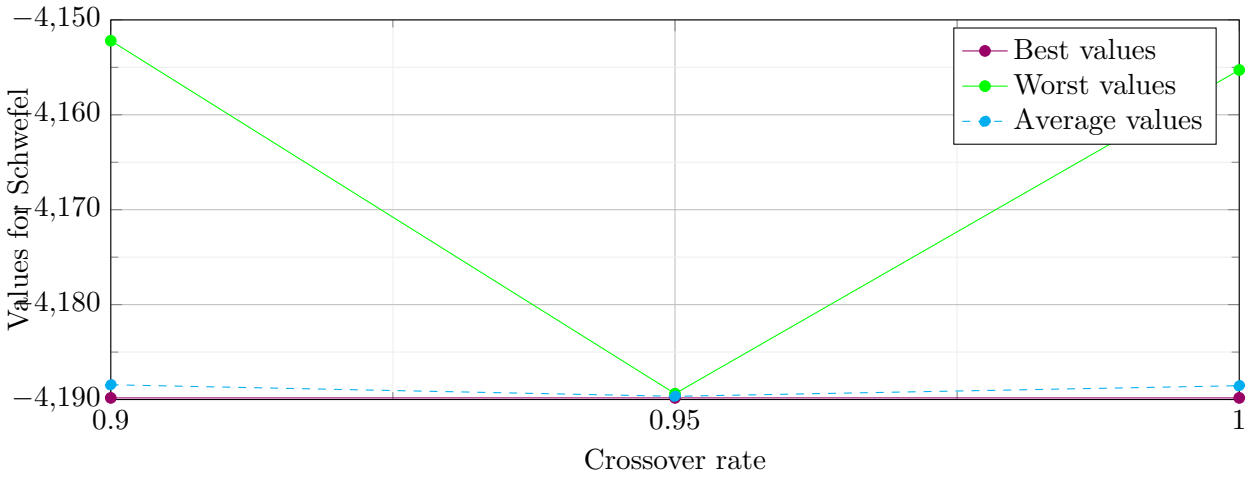

Figure 7: cRate study 10 dimensions Schwefel's function

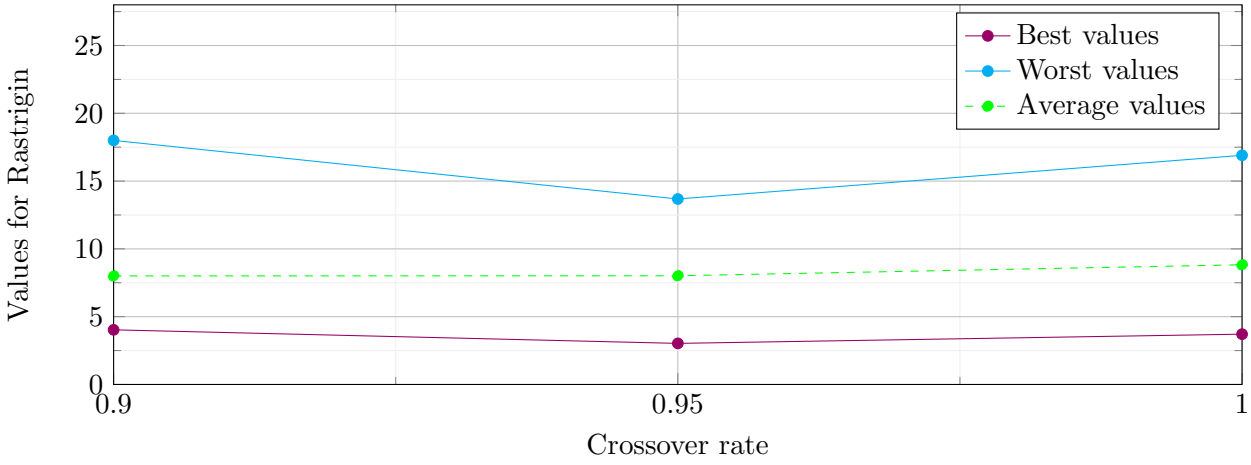

Figure 8: cRate study 30 dimensions Rastrigin's function

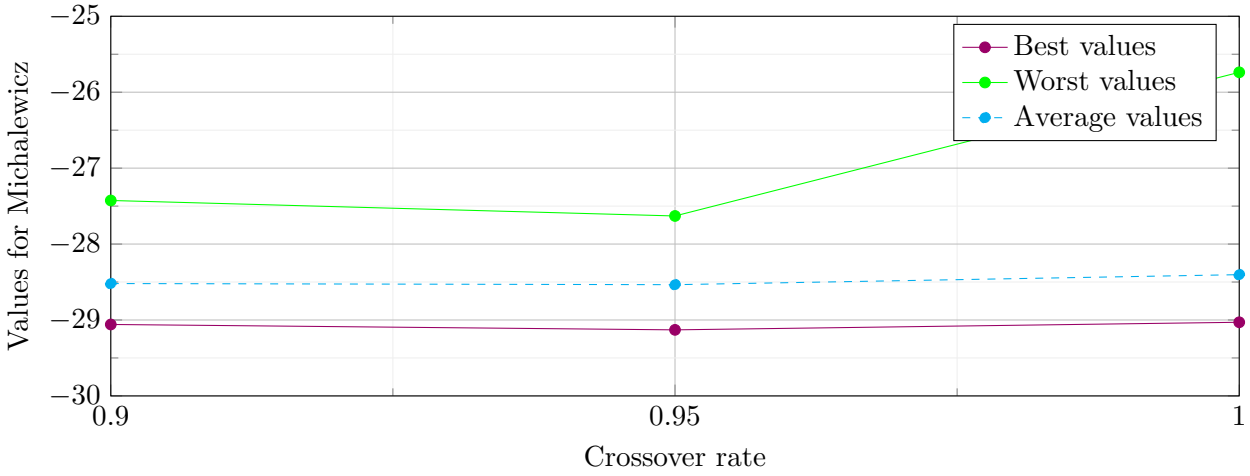

Figure 9: cRate study 30 dimensions Michalewicz's function

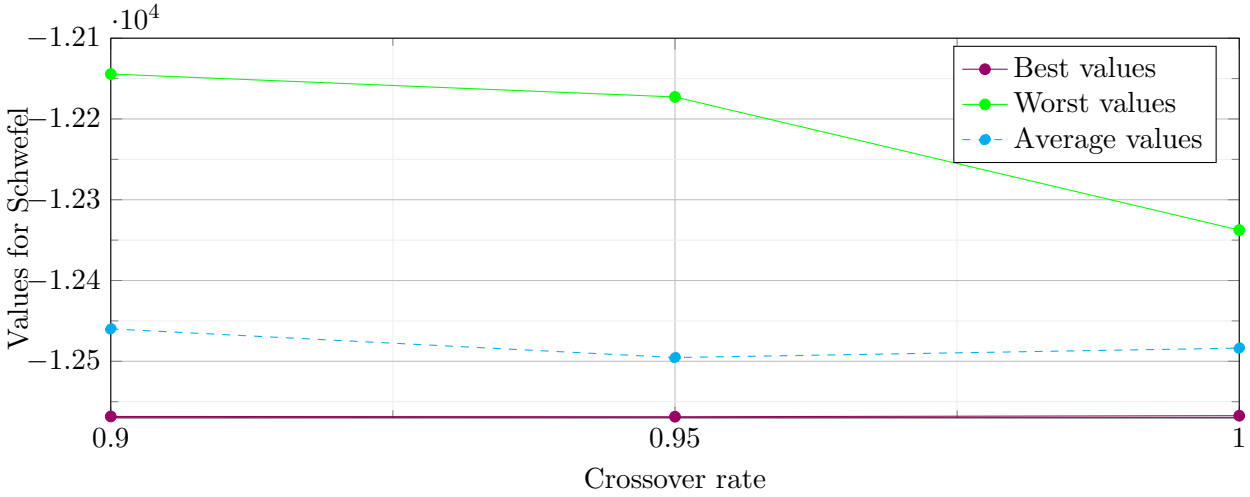

Figure 10: cRate study 30 dimensions Schwefel's function

Judging by what we said prior we can conclude that crossover rate 0.95 works better than the other two. To optimize the parameters even more we tried to find the optimal mutation rate for this crossover probability. We tested 3 different mutation rates 0.001, 0.005 and 0.01 (tables 4-6).

5.1.4 Results 2 dimensions with different mutation rates (mRate)

mRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.001	De Jong's 1	2.01s	3.23s	7.17s	0	0	0
	Rastrigin's	2.28s	3.66s	7.13s	0	0.00002	0
	Michalewicz's	2.08s	3.39s	6.22s	-1.8013	-1.80037	-1.80119
	Schwefel's	2.99s	9.29s	16.22s	-837.96577	-837.86148	-837.94833
0.005	De Jong's 1	2.2s	2.91s	4.98s	0	0	0
	Rastrigin's	2.11s	2.77s	3.54s	0	0	0
	Michalewicz's	2.13s	2.68s	4.22s	-1.8013	-1.8013	-1.8013
	Schwefel's	3.81s	6.68s	10.68s	-837.96577	-837.96577	-837.96577
0.01	De Jong's 1	2.26s	2.91s	4.61s	0	0	0
	Rastrigin's	2.31s	3.1s	5.4s	0	0	0
	Michalewicz's	2.24s	2.83s	4s	-1.8013	-1.8013	-1.8013
	Schwefel's	4.34s	6.67s	9.89s	-837.96577	-837.96515	-837.96575

Table 4: Results mRate 2-dimensional version

5.1.5 Results 10 dimensions with different mutation rates (mRate)

mRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.001	De Jong's 1	13.71s	20.09s	26.79s	0	0	0
	Rastrigin's	13.41s	22.94s	28.17s	0	2.23204	0.28322
	Michalewicz's	19.6s	39.07s	45.54s	-9.66015	-9.20726	-9.55606
	Schwefel's	34s	35.26s	37.09s	-4189.82761	-4189.38074	-4189.65897
0.005	De Jong's 1	11.15s	14.62s	19.39s	0	0	0
	Rastrigin's	11.48s	17.53s	27.21s	0	0.99496	0.03317
	Michalewicz's	20.25s	33.4s	43.91s	-9.66015	-9.51006	-9.64173
	Schwefel's	26.66s	34.2s	35.75s	-4189.82886	-4189.41296	-4189.74401
0.01	De Jong's 1	8.86s	14s	20.12s	0	0	0
	Rastrigin's	11.84s	16.29s	24.09s	0	0.99496	0.13334
	Michalewicz's	19.18s	34.44s	44.5s	-9.66015	-9.51013	-9.65056
	Schwefel's	21.76s	34.29s	35.97s	-4189.82886	-4189.41385	-4189.73592

Table 5: Results mRate 10-dimensional version

5.1.6 Results 30 dimensions with different mutation rates (mRate)

mRate	Functions	Time			Solution		
		Min	Avg	Max	Best	Worst	Avg
0.001	De Jong's 1	67.37s	70.85s	75.57s	0	0.00640	0.00041
	Rastrigin's	60.48s	72.3s	79.63s	3.03084	13.68101	8.01771
	Michalewicz's	48.93s	83.85s	94.85s	-29.13489	-27.63084	-28.53505
	Schwefel's	85.69s	95.58s	107.08s	-12568.62735	-12172.71368	-12495.36944
0.005	De Jong's 1	66.6s	68.93s	74.66s	0.00001	0.00188	0.00052
	Rastrigin's	69.01s	71.31s	76.72s	3.66093	13.84741	8.49787
	Michalewicz's	78.18s	84.14s	91.28s	-29.13037	-28.20371	-28.67536
	Schwefel's	90.62s	95.79s	104.27s	-12568.369	-12533.66924	-12559.81811
0.01	De Jong's 1	66.78s	68.07s	71.24s	0.00002	0.00683	0.00088
	Rastrigin's	66.24s	70.3s	75.92s	1.27902	13.51294	7.29616
	Michalewicz's	75.04s	82.33s	89.27s	-29.22989	-28.14024	-28.74358
	Schwefel's	83.76s	92.49s	98.6s	-12568.85057	-12437.40602	-12548.88311

Table 6: Results mRate 30-dimensional version

We can see from the tables 4-6 that a bigger mutation rate leads to better solutions for the multimodal functions. For De Jong's function 1 a bigger mutation rate leads to worst solutions on average. If we compare the best solutions for the multimodal functions on 30 dimensions we can see that mutation rate 0.01 manages to find a better global minimum. (figures 11-13)

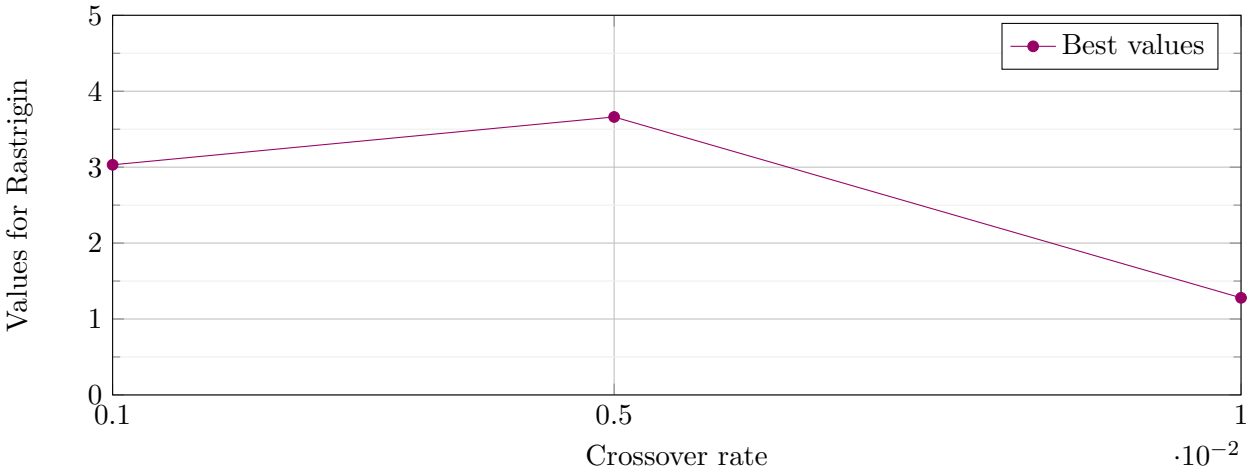

Figure 11: mRate study 30 dimensions Rastrigin's function

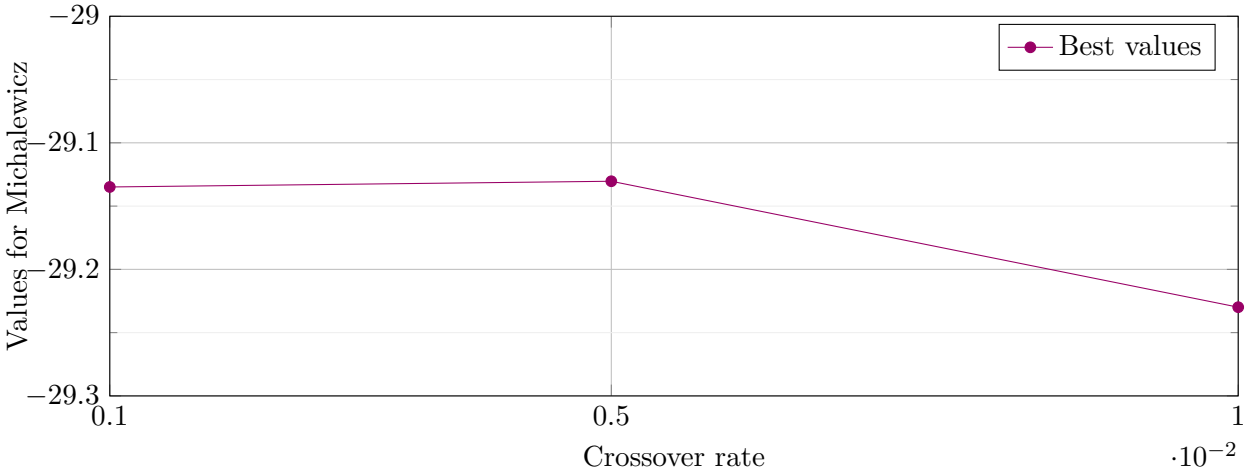

Figure 12: mRate study 30 dimensions Michalewicz's function

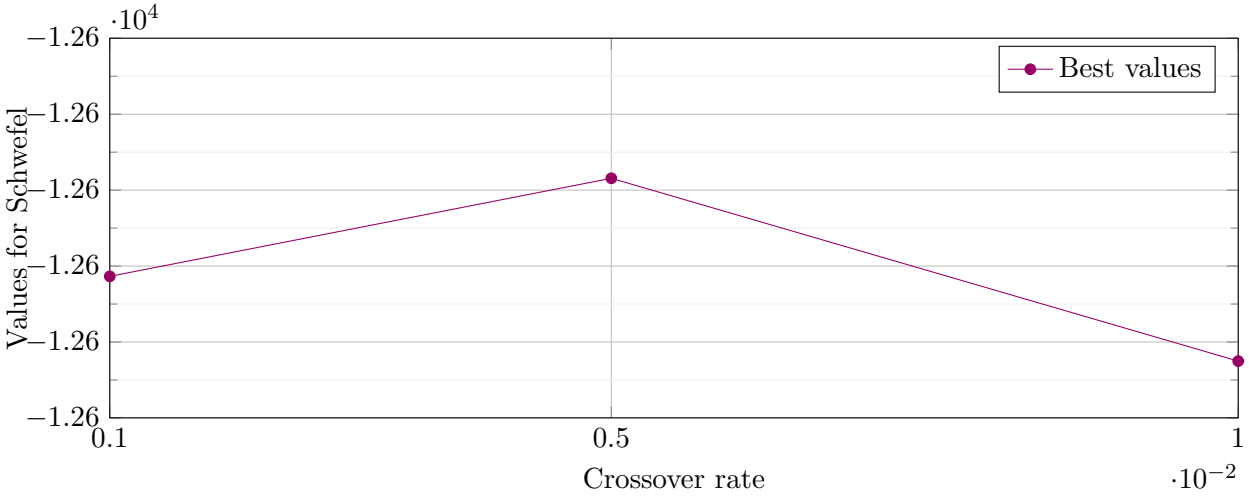

Figure 13: mRate study 30 dimensions Schwefel's function

In conclusion, we can say that, in our case, the best crossover rate is 0.95 with 0.01 mutation rate.

5.2 Comparison of GA with BIHC and SA

The experiments were made for 2, 10 and 30 dimensions 30 tests each with precision 5 for every algorithm.

We tested simulated annealing algorithm with $T = t = 1000$ (t iterations for every $T \in \{10^3, 10^2 ... 10^{-7}\}$), iterated hill climbing best-improvement algorithm with 1000 iterations and for the genetic algorithm we used the best results we found by optimizing the parameters (MAX = 10000, populationSize = 100, mutation percentage in population = 100, elites percentage = 10, cRate = 0.95 and mRate = 0.01).

5.2.1 Results 2 dimensions

Alg	Functions	Time			Solution			σ
		Min	Avg	Max	Best	Worst	Avg	
SA	De Jong's 1	0.24s	0.24s	0.26s	0	0.00001	0	0
	Rastrigin's	0.24s	0.24s	0.25s	0	0.00037	0.00008	0.00007
	Michalewicz's	0.22s	0.23s	0.24s	-1.8013	-1.8007	-1.80128	0.00011
	Schwefel's	0.3s	0.32s	0.34s	-837.96577	-837.75309	-837.89489	0.06956
BIHC	De Jong's 1	0.88s	0.94s	1.01s	0	0	0	0
	Rastrigin's	0.78s	0.84s	0.88s	0	0.00152	0.00008	0.00027
	Michalewicz's	0.69s	0.73s	0.78s	-1.8013	-1.8013	-1.8013	0
	Schwefel's	2.22	2.32s	2.47s	-837.96577	-837.96412	-837.9655	0.00039
GA	De Jong's 1	2.26s	2.91s	4.61s	0	0	0	0
	Rastrigin's	2.31s	3.1s	5.4s	0	0	0	0
	Michalewicz's	2.24s	2.83s	4s	-1.8013	-1.8013	-1.8013	0
	Schwefel's	4.34s	6.67s	9.89s	-837.96577	-837.96515	-837.96575	0.00011

Table 7: Results 2-dimensional version

5.2.2 Results 10 dimensions

Alg	Functions	Time			Solution			σ
		Min	Avg	Max	Best	Worst	Avg	
SA	De Jong's 1	0.66s	0.69s	0.77s	0.00035	0.00189	0.0008	0.00037
	Rastrigin's	0.69s	0.71s	0.75s	1.12037	18.91075	8.76412	4.67667
	Michalewicz's	0.75s	0.77s	0.83s	-9.38364	-8.20529	-8.9943	0.23792
	Schwefel's	0.93s	0.94s	0.98s	-4189.21289	-3952.42035	-4107.11749	71.4331
BIHC	De Jong's 1	28.24s	29.01s	29.65s	0	0	0	0
	Rastrigin's	24.5s	25.24s	25.82s	2.47194	8.20618	5.68509	1.15565
	Michalewicz's	26.8s	27.52s	28.1s	-9.46303	-8.89546	-9.23763	0.11327
	Schwefel's	68.63s	70.66s	72.96s	-4128.29369	-3880.1473	-3974.99471	70.1451
GA	De Jong's 1	8.86s	14s	20.12s	0	0	0	0
	Rastrigin's	11.84s	16.29s	24.09s	0	0.99496	0.13334	0.34375
	Michalewicz's	19.18s	34.44s	44.5s	-9.66015	-9.51013	-9.65056	0.02851
	Schwefel's	21.76s	34.29s	35.97s	-4189.82886	-4189.41385	-4189.73592	0.10322

Table 8: Results 10-dimensional version

5.2.3 Results 30 dimensions

Alg	Functions	Time			Solution			σ
		Min	Avg	Max	Best	Worst	Avg	
SA	De Jong's 1	1.9s	1.97s	2.13s	0.01111	0.0407	0.02139	0.00864
	Rastrigin's	1.99s	2.03s	2.12s	35.603	77.18658	56.22618	9.8651
	Michalewicz's	1.89s	1.96s	2.07s	-26.86676	-22.62479	-25.24687	0.97333
	Schwefel's	2.66s	2.72s	2.88s	-11546.48915	-10465.19389	-11087.01334	301.7024
BIHC	De Jong's 1	343.45s	355.84s	366.26s	0	0	0	0
	Rastrigin's	293.44s	302.88s	311.45s	26.61194	42.80525	37.14005	3.69216
	Michalewicz's	242.05s	250.9s	257.82s	-26.74449	-25.67914	-26.16094	0.291
	Schwefel's	815.13s	879.85s	959.95s	-11367.11059	-10736.08596	-10995.81213	146.0378
GA	De Jong's 1	66.78s	68.07s	71.24s	0.00002	0.00683	0.00088	0.00167
	Rastrigin's	66.24s	70.3s	75.92s	1.27902	13.51294	7.29616	2.90844
	Michalewicz's	75.04s	82.33s	89.27s	-29.22989	-28.14024	-28.74358	0.26531
	Schwefel's	83.76s	92.49s	98.6s	-12568.85057	-12437.40602	-12548.88311	33.37485

Table 9: Results 30-dimensional version

Comparison

We can see that hill climbing best-improvement finds better solutions than simulated annealing, but it takes more time for the algorithm to execute. As for the genetic algorithm we can see how robust it is in comparison with SA and BIHC and it finds the best solutions for each dimension tested. Moving forward, BIHC seems to work best at finding the global minimum for the unimodal function (De Jong's function 1) showing that the search is more precise if the function has only one global minimum.

6 Conclusions

Three algorithms: Iterated Hill Climbing, Simulated Annealing and a Genetic Algorithm are described and compared in this paper. The simulations have been carried out using benchmarking functions, such as De Jong's function 1, Rastrigin's function, Michalewicz's function and Schwefel's function. The genetic algorithm mutation and crossover rates have been studied and improved according to the experiments. A comparison between the algorithms was presented. The computational results have demonstrated that simulated annealing algorithm performs faster than iterated hill climbing algorithm, but it gives worst solutions and we've seen that the genetic algorithm outperforms both simulated annealing and iterated hill climbing in terms of solutions found and robustness of the algorithm.

References

[1] Daniel Delahaye, Supatcha Chaimatanan, and Marcel Mongeau. Simulated annealing: From basics to applications. In *Handbook of metaheuristics*, pages 1–35. Springer, 2019.

[2] Jason G Digalakis and Konstantinos G Margaritis. On benchmarking functions for genetic algorithms. *International journal of computer mathematics*, 77(4):481–506, 2001.

[3] Javatpoint contributors. Hill climbing algorithm in artificial intelligence, 2021. [Online; accessed 21-November-2021].

[4] Hartmut Pohlheim. Examples of objective functions. *Retrieved*, 4(10):2012, 2007.

[5] SN Sivanandam and SN Deepa. Genetic algorithm optimization problems. In *Introduction to genetic algorithms*, pages 20–21. Springer, 2008.

[6] Charlie Vanaret, Jean-Baptiste Gotteland, Nicolas Durand, and Jean-Marc Alliot. Certified global minima for a benchmark of difficult optimization problems. *arXiv preprint arXiv:2003.09867*, 2020.

[7] Wikipedia contributors. Hill climbing — Wikipedia, the free encyclopedia, 2021. [Online; accessed 21-November-2021].

[8] Wikipedia contributors. Simulated annealing — Wikipedia, the free encyclopedia, 2021. [Online; accessed 21-November-2021].