Optimization using genetic algorithms

Mazilu Cosmin-Alexandru

UAIC
Computer Science

December 1, 2024

Abstract

Finding the optimal solutions, in Computer Science, it's a challenging problem, as it requires a lot of optimizations and much computational resources to find optimal solutions. A Genetic Algorithm is a population-based optimization technique inspired by the principles of natural selection and genetics. It evolves solutions to a problem by iteratively applying operators such as selection, crossover, and mutation. By maintaining a diverse pool of candidate solutions, GAs efficiently explore the search space and adapt over generations, making them particularly effective in solving complex multimodal optimization problems. The results are compared with those obtained using hill climbing and simulated annealing. Key differences in performance, precision, and computational complexity are analyzed.

Introduction

The purpose of this report is to explore the effectiveness of the Genetic Algorithm in finding the minima of several well-known functions, on different dimensions, in comparison to the trajectory algorithms, Hill Climbing and Simulated Annealing. A good understanding of these methods, their strengths and weaknesses, and their results is essential for applying optimization techniques in real-world problems.

Methods

Genetic Algorithm

In a genetic algorithm, a population of candidate solutions is developed, termed individuals or phenotypes, to improve their fitness for a given optimization problem. Each candidate solution is characterized by a set of properties, known as chromosomes, which can undergo alterations such as mutation or crossover. Traditionally, these solutions are encoded as binary strings of 0s and 1s.
The process begins with a randomly generated population and proceeds iteratively, with each iteration referred to as a generation. During each generation, the fitness of each individual is assessed, typically by evaluating the objective function of the optimization problem. Based on their fitness, individuals are stochastically selected to contribute to the next generation. Their genetic information may be recombined and occasionally mutated to produce a new population. This process repeats until a stopping criterion is met, such as reaching a predetermined number of generations or achieving a satisfactory fitness level.

Initialization of the Genetic Algorithm

An initial population of candidate solutions is typically created by generating many random solutions. The size of this population depends on the specific problem but often consists of several hundred to thousands of potential solutions. The population is traditionally generated randomly to ensure it spans the entire search space, providing diverse starting points for the optimization process. [3]

Selection of the candidates

Roulette wheel selection is a widely used method in genetic algorithms for randomly selecting parents for reproduction, with the probability of selection based on their fitness scores. The process resembles spinning a roulette wheel, where each individual in the population is assigned a slice proportional to its fitness. The wheel is then spun, and the parent is chosen based on where the wheel stops, with individuals having higher fitness scores occupying larger slices. This approach increases the likelihood of selecting individuals with better fitness, promoting the generation of superior offspring [1].

Crossover operator

This operator selects genes from two parent chromosomes to create a new offspring. The simplest approach involves randomly selecting a crossover point. Genes before this point are copied from the first parent, while genes after the crossover point are copied from the second parent. [3]

Mutation operator

To prevent the entire population from converging to a local optimum in the problem, mutation introduces random changes in the offspring. In the case of binary encoding, mutation involves flipping a few randomly selected bits, changing them from 1 to 0 or from 0 to 1. [3]

Elitism

Elitism in genetic algorithms refers to a strategy where the best individuals (solutions) from the current population are guaranteed to survive to the next generation without any modification. This ensures that the highest-quality solutions are not lost during the process of selection, crossover, and mutation.

Representation of solutions

The solutions will be represented as bit strings [4]. The search space will be discretized up to a certain precision 10^{-d}, where d is the precision, the number of decimals of the solutions. For the function $f : [a, b]^n \to R$, the domain $[a, b]$ will be divided into a number of equal subintervals $(b - a) \times 10^d$, noted as N. For representing all the N values, a number of ceil($log_2 N$) bits, noted as x. The length of the bit string representing a candidate solution is the sum of the lengths of all the parameters of the function to optimize. For evaluating a candidate solution (calling the optimization function), it is necessary to decode every parameter represented as a bit string to a real number, using the formula:

$$X_{real} = a + \frac{decimal(x_{bits}) \times (b - a)}{(2^x - 1)}$$

A candidate solution is made by random generating a string of bits with the length of x. The neighboring solutions are generated by flipping the bits one by one from the candidate solution.

Tested Functions

- **De Jong 1 Function:** it's a continous, convex and unimodal function, also known as sphere model. The definition of the function is: $f : [-5.12, 5.12]^n \to R$ where the image of the function is calculated with the following formula:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2, \quad where \quad x = (x_1, x_2, \ldots, x_n)$$

.

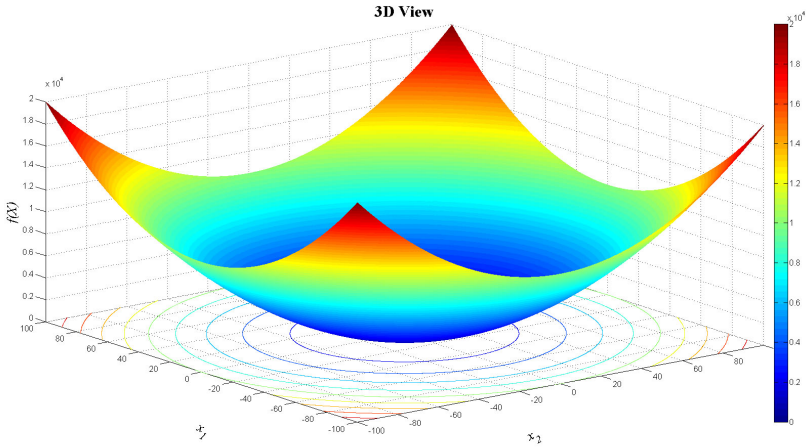

Figure 1: Illustration of the De Jong function. Global minimum = 0. [2]

- **Schewefel's Function:** is a complex function, having many local minima. The definition of the function is $f : [-500, 500]^n \to R$ where the image of the function is calculated with the following formula:

$$f(\mathbf{x}) = 418.9829 \times n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}), \quad where \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)$$

.

- **Rastrigin's Function:** is a non-convex, non-linear multimodal function. The definition of the function is: $f : [-5.12, 5.12]^n \to R$ where the image of the function is calculated with the following formula:

$$f(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right], \quad where \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)$$

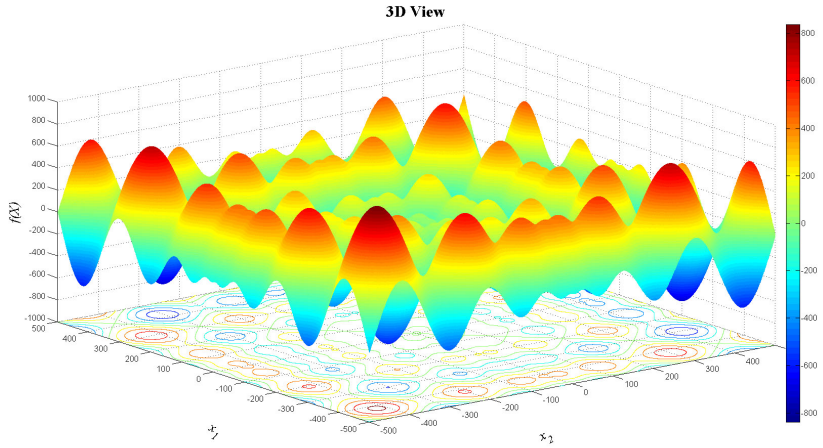

Figure 2
Illustration of the Schwefel's function. Global minimum = -12569.487 [2]

.

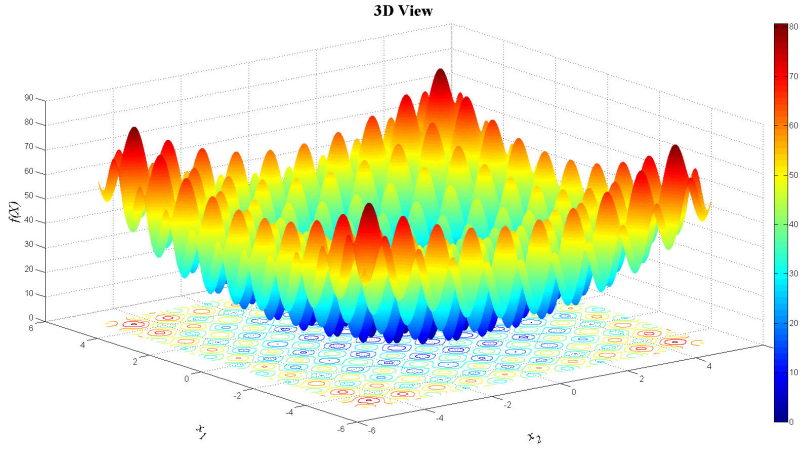

Figure 3
Illustration of the Ratrigin's function. Global minimum = 0 [2]

- **Michalewicz's Function:** is a multimodal function. The parameter m influences the "steepness" of the valleys or edges. The definition of the function is $f : [0, \pi]^n \rightarrow R$ where the image of the function is calculated with the following formula:

$$f(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \left(\sin\left(\frac{ix_i^2}{\pi} \right) \right)^{2m} \quad where \quad \mathbf{x} = (x_1, x_2, \ldots, x_n) \quad and \quad m \quad is \quad typically \quad 10$$

.

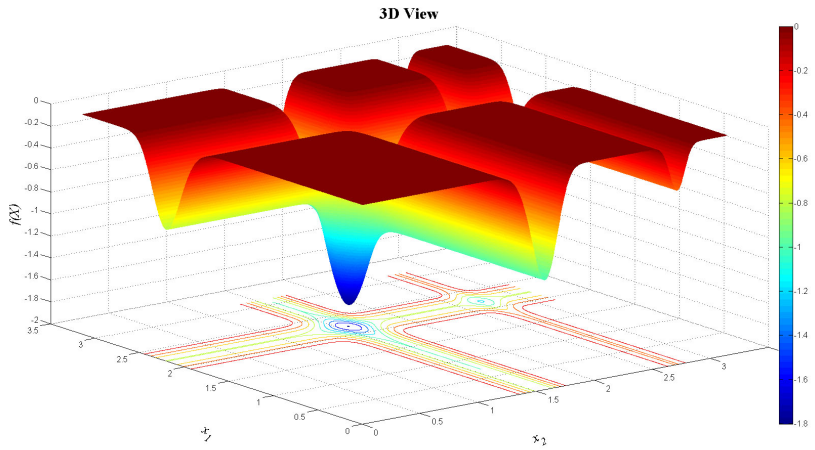

Figure 4
Illustration of the Michalewicz's function. Global minimum = -29.63088 [2]

Experiments description

Every function will be optimized 30 times with the following parameters and values:

- Number of dimensions: 5, 10 and 30;
- Precision: 5;
- Population size: 100 (DeJong, Rastrigin, Michalewicz), 50 (Schwefel);
- Crossover rate: 0.3 (DeJong), 0.65 (Schwefel), 0.7 (Rastrigin, Michalewicz):
- Mutation rate: 0.01 (DeJong), 0.09 (Schwefel), 0.3(Rastrigin, Michalewicz);
- Chromosomes selected in elitism: 0.02 (DeJong, Schewefel), 0.09 (Rastrigin), 0.1 (Michalewicz)
- Maximum generations: 1000 (DeJong), 10.000(Schwefel, Rastrigin, Michalewicz);

Experimental results:

DeJong

Genetic Algorithm DeJong1 Function			
	n=5	n=10	n=30
Mean	0.00000	0.00000	0.00000
Standard deviation	0.00000	0.00000	0.00000
Min value	0.00000	0.00000	0.00000
Max value	0.00000	0.00000	0.00000
Mean Exec Time	0.09425	0.147405	0.49874

Table 1: Statistical Metadata for DeJong1 Function

First Improvement DeJong1 Function			
	n=5	n=10	n=30
Mean	0.00000	0.00000	0.00000
Standard deviation	0.00000	0.00000	0.00000
Min value	0.00000	0.00000	0.00000
Max value	0.00000	0.00000	0.00000
Mean Exec Time	0.00222	0.00959	0.16368

Table 2: Statistical Metadata for DeJong1 Function

Best Improvement DeJong1 Function			
	n=5	n=10	n=30
Mean	0.00000	0.00000	0.00000
Standard deviation	0.00000	0.00000	0.00000
Min value	0.00000	0.00000	0.00000
Max value	0.00000	0.00000	0.00000
Mean Exec Time	0.004	0.01741	0.30502

Table 3: Statistical Metadata for DeJong1 Function

Simulated Annealing DeJong1 Function			
	n=5	n=10	n=30
Mean	0.00000	0.00000	0.00000
Standard deviation	0.00000	0.00000	0.00000
Min value	0.00000	0.00000	0.00000
Max value	0.00000	0.00000	0.00000
Mean Exec Time	0.00373	0.00540	0.01264

Table 4: Statistical Metadata for DeJong1 Function

Schwefel

Genetic Algorithm Schwefel Function			
	n=5	n=10	n=30
Mean	-1914.84364	3653.92792	-10344.38101
Standard deviation	195.05802	362.67523	659.94385
Min value	-2094.81013	-4189.61966	-11549.56823
Max value	-1503.80794	-3031.43504	-9219.52647
Mean Exec Time	0.38697	0.70429	1.85837

Table 5: Statistical Metadata for Schwefel's Function

First Improvement Schwefel's Function			
	n=5	n=10	n=30
Mean	-1913.73706	-3618.37679	-9998.97778
Standard deviation	54.31144	92.17866	169.62926
Min value	-2026.44337	-3837.4567	-10536.2526
Max value	-1788.87621	-3404.65047	-9557.61741
Mean Exec Time	0.27080	1.94091	42.88199

Table 6: Statistical Metadata for Schwefel's Function

Best Improvement Schwefel's Function			
	n=5	n=10	n=30
Mean	-2074.91584	-3941.81087	-10966.78591
Standard deviation	29.15409	84.34130	190.20069
Min value	-2094.91318	-4162.42753	-11257.81395
Max value	-1976.36625	-3796.357	-10624.33781
Mean Exec Time	0.42874	2.94312	66.25076

Table 7: Statistical Metadata for Schwefel's Function

Simulated Annealing Schwefel's Function			
	n=5	n=10	n=30
Mean	-1813.97824	-3274.584804	-8888.26624
Standard deviation	60.39023	100.41182	268.23234
Min value	-1929.6588	-3529.69716	-9773.03541
Max value	-1707.37893	-3052.12378	-8463.37485
Mean Exec Time	0.08047	0.16456	0.62429

Table 8: Statistical Metadata for Schwefel's Function

Rastrigin

Genetic Algorithm Rastrigin's Function			
	n=5	n=10	n=30
Mean	2.17896	9.95013	54.34161
Standard deviation	1.05875	3.25936	9.76767
Min value	0.99496	5.46663	34.55503
Max value	4.92341	20.71354	67.78512
Mean Exec Time	1.11324	1.81020	4.95785

Table 9: Statistical Metadata for Rastrigin's Function

First Improvement Rastrigin's Function			
	n=5	n=10	n=30
Mean	0.03333	3.60602	29.39506
Standard deviation	0.17951	1.40487	3.22901
Min value	0.00000	1.18523	20.61045
Max value	1.00001	6.23586	33.62691
Mean Exec Time	0.27038	1.50082	31.91291

Table 10: Statistical Metadata for Rastrigin's Function

Best Improvement Rastrigin's Function			
	n=5	n=10	n=30
Mean	0.06633	2.51278	23.06865
Standard deviation	0.24819	0.82836	2.88098
Min value	0.00000	1.48665	17.28164
Max value	0.99496	4.99574	28.30786
Mean Exec Time	0.35091	2.76706	60.53943

Table 11: Statistical Metadata for Rastrigin's Function

Simulated Annealing Rastrigin's Function			
	n=5	n=10	n=30
Mean	0.00795	1.31080	28.74744
Standard deviation	0.00495	0.62800	4.26449
Min value	0.00068	0.09158	14.44787
Max value	0.023	2.53979	34.66966
Mean Exec Time	5.19192	11.0387	41.69259

Table 12: Statistical Metadata for Rastrigin's Function

Michalewicz

Genetic Algorithm Michalewicz's Function			
	n=5	n=10	n=30
Mean	-3.57993	-7.943568	-26.01725
Standard deviation	0.45674	0.95445	1.26877
Min value	-3.95565	-8.88528	-27.78358
Max value	-2.33705	-6.18966	-21.45934
Mean Exec Time	1.25729	2.16353	6.18854

Table 13: Statistical Metadata for Michalewicz's Function

First Improvement Michalewicz's Function			
	n=5	n=10	n=30
Mean	-3.69439	-8.18847	-24.723979
Standard deviation	0.00586	0.12416	0.43640
Min value	-3.69886	-8.52691	-25.70863
Max value	-3.68159	-7.91652	-23.89163
Mean Exec Time	0.32288	2.02787	46.65240

Table 14: Statistical Metadata for Michalewicz's Function

Best Improvement Michalewicz's Function			
	n=5	n=10	n=30
Mean	-3.69740	-8.38411	-25.62190
Standard deviation	0.00325	0.12800	0.28674
Min value	-3.69886	-8.65284	-26.30779
Max value	-3.68219	-8.16545	-25.17724
Mean Exec Time	0.48724	3.30066	76.01679

Table 15: Statistical Metadata for Michalewicz's Function

Simulated Annealing Michalewicz's Function			
	n=5	n=10	n=30
Mean	-3.69632	-8.38056	-24.43748
Standard deviation	0.01353	0.10828	0.46230
Min value	-3.69886	-8.64906	-25.77466
Max value	-3.62345	-8.1489	-23.69677
Mean Exec Time	3.10168	6.70004	31.84822

Table 16: Statistical Metadata for Michalewicz's Function

Influence of the parameters

A larger population size promotes diversity and better exploration of the search space, reducing the risk of premature convergence. However, it also increases computational cost. For example, increasing the population size from 50 to 100 improved solution quality in higher dimensions but nearly doubled the runtime.

A high crossover rate (e.g., 0.8–0.9) facilitates the exchange of genetic material between solutions, promoting exploration. However, if set too high, it can disrupt convergence by generating less stable offspring. A moderate rate of 0.7 provided a balance between diversity and stability.

Mutation prevents stagnation by introducing random variations, enabling the algorithm to escape local optima. A low mutation rate (e.g., 0.01) preserved good solutions but occasionally led to stagnation in multimodal problems. Higher mutation rates (e.g., 0.1) improved robustness but increased the variability in results.

Retaining the best solutions from one generation ensures that high-quality candidates are not lost. Without elitism, the algorithm occasionally regressed in solution quality, especially for complex functions.

Differences between algorithms

The experimental results demonstrate significant differences in performance among the optimization algorithms. Genetic Algorithms (GAs) tend to perform well across most tested functions due to their population-based approach, which enables exploration and exploitation of the search space. For unimodal functions like De Jong 1, all algorithms consistently found the global minimum with minimal standard deviation, as expected given the simplicity of the problem. However, GAs exhibit longer execution times due to their iterative and population-dependent nature.

For multimodal functions like Schwefel's and Rastrigin's, GAs showed a higher robustness in avoiding local minima, as evidenced by their superior mean values compared to Hill Climbing variants and Simulated Annealing (SA). The elitism strategy and diverse population help ensure the convergence to better solutions. However, GAs required longer execution times, particularly in higher-dimensional cases.

Conclusions

In conclusion, Genetic Algorithms provide a robust optimization method capable of finding high-quality solutions to complex multimodal problems. While computationally expensive, their ability to balance exploration and exploitation and to avoid local optima makes them suitable for a wide range of applications.

Future work could involve exploring adaptive parameters, operators, Gray coding, meta algorithms to urther enhance efficiency and scalability.

Bibliography

[1] Roulette Wheel Selection. https://cratecode.com/info/roulette-wheel-selection, 2024.

[2] Ali R. Al-Roomi. Unconstrained Single-Objective Benchmark Functions Repository, 2015.

[3] PRATIBHA BAJPAI and M. Kumar. Genetic algorithm - an approach to solve global optimization problems. *Indian Journal of Computer Science and Engineering*, 1:199–206, 10 2010.

[4] Lect. Dr. Eugen Croitoru. Genethic Algorithms. https://profs.info.uaic.ro/eugen.croitoru/teaching/ga/#Notions02, 2024.