Analysis of a Genetic Algorithm for Booth Function Optimization

Kerem Adalı

April 4, 2025

Abstract

This report presents an analysis of a genetic algorithm implementation for optimizing various test functions. We examine the effects of different parameter configurations on the algorithm's performance, including population size, mutation rate, mutation strength, and crossover rate. Through a series of four structured experiments, we investigate the impact of randomness, crossover operations, and mutation parameters on convergence behavior and solution quality. Our findings highlight the importance of balancing exploration and exploitation in genetic algorithms through proper parameter tuning.

1 Introduction

Genetic algorithms (GAs) are a class of optimization techniques inspired by natural selection and evolutionary processes. They are particularly useful for solving complex optimization problems where traditional gradient-based methods may struggle. This report examines a GA implementation designed to minimize various 2D test functions, with a focus on the Booth function.

$$\textbf{Booth Function: } f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2 \tag{1}$$

The GA implemented in this study follows the standard evolutionary process:

1. Initialize a population of potential solutions
2. Evaluate the fitness of each solution
3. Select parents for reproduction based on fitness
4. Apply crossover operations to create offspring
5. Apply mutation to maintain diversity
6. Repeat until termination criteria are met

Our implementation includes several key parameters that influence performance:

- Population size (N)
- Mutation rate (p_m)
- Mutation strength (s_m)
- Crossover rate (p_c)
- Number of generations (G)

2 Algorithm Details

2.1 Implementation Overview

The genetic algorithm is implemented in Python with the following components:

Algorithm 1 Genetic Algorithm

```
Initialize population of size N
for g = 1 to G do
    Evaluate fitness of each individual
    Select parents for reproduction with probability p_c
    Apply crossover to create offspring
    Add offspring to population
    Apply mutation with probability p_m and strength s_m
    Record best and average fitness
end for
return best solution found
```

2.2 Key Components

2.2.1 Initialization

The population is initialized with random individuals within the bounds defined for each test function. For the Booth function, individuals are initialized within the range $[-5, 5]$ for both dimensions.

2.2.2 Fitness Evaluation

The fitness of each individual is determined by the test function being minimized. Lower function values indicate better fitness.

2.2.3 Selection

Selection is implemented using fitness-proportionate selection (roulette wheel selection) with a twist: only individuals that pass a crossover probability check (p_c) are considered as candidates for selection. From the candidate pool, two individuals are selected with probability proportional to the inverse of their fitness values.

2.2.4 Crossover

The algorithm implements an arithmetic crossover operation where offspring are created as a weighted average of two parents:

$$x_o = \alpha \cdot x_{p1} + (1 - \alpha) \cdot x_{p2},$$
$$y_o = \alpha \cdot y_{p1} + (1 - \alpha) \cdot y_{p2},$$

where o, p1, p2 refer to offspring, parent 1 and parent 2, and α is a random number

2.2.5 Mutation

Mutation is applied with probability p_m to each individual. When an individual is selected for mutation, a Gaussian perturbation with standard deviation s_m is added to each dimension:

$$\text{individual}' = \text{individual} + \mathcal{N}(0, s_m) \tag{2}$$

3 Experimental Design

We conducted four experiments to analyze different aspects of the genetic algorithm:

1. **Parameter Tuning**: Finding effective parameter combinations for the Booth function
2. **Randomness Analysis**: Examining the effect of different random seeds and population sizes
3. **Crossover Impact**: Investigating how crossover rate affects performance
4. **Mutation and Convergence**: Analyzing the trade-off between exploration and exploitation through mutation parameters

4 Results and Discussion

4.1 Experiment 1: Parameter Tuning

In this experiment, we tested various parameter combinations to identify effective settings for optimizing the Booth function. Seven different configurations were evaluated, varying population size, mutation rate, mutation strength, crossover rate, and number of generations.

Table 4.1 presents the results of our parameter tuning experiment. The data reveals several important patterns. First, increasing the population size from 1000 to 5000 dramatically improved solution quality, with the best configuration achieving a perfect fitness of 0.0 (exact solution at [1, 3]). Second, doubling the mutation rate from 0.1 to 0.2 with a population size of 1000 yielded a significant improvement in fitness (from 1.08e-11 to 1.86e-23), suggesting that higher mutation rates can compensate for smaller populations by increasing exploration. Interestingly, increasing mutation strength from 0.1 to 0.2 showed no improvement with the population size of 1000, indicating that mutation rate has a stronger impact than mutation strength in this context.

Table 1: Tuning Results

population_size	mutation_rate	mutation_strength	crossover_rate	best_solution	best_fitness
1000	0.1	0.1	0.7	(0.99999965424647, 2.9999988249723)	1.07513375090620e-11
2000	0.1	0.1	0.7	(1.00215802416776, 2.99806182485430)	8.6069255740614e-06
1000	0.2	0.1	0.7	(1.00000000000256, 2.9999999999991)	1.86345243679504e-23
1000	0.1	0.2	0.7	(0.99999965424647, 2.9999988249723)	1.07513375090620e-11
1000	0.1	0.1	0.8	(0.9999689492575, 3.00000276361168)	4.1724332238992e-09
5000	0.1	0.1	0.7	(1.00000000438818, 2.99999999684676)	3.52994534212886e-17
5000	0.1	0.1	0.7	(1.00000000000004, 2.99999999999996)	0.0

The results also demonstrate the importance of proper parameter balance. The final configuration (population size 5000, mutation rate 0.1, mutation strength 0.1, crossover rate 0.7) achieved the theoretical optimum with a fitness of 0.0, confirming that larger populations combined with well-balanced mutation and crossover parameters provide superior performance for this optimization problem.

4.2 Experiment 2: Randomness Analysis

This experiment assessed the algorithm's sensitivity to random initialization by running the best parameter combination from Experiment 1 with five different random seeds. Additionally, we tested the effect of decreasing population size on performance stability.

Table 2 shows the best fitness values achieved with different random seeds using the optimal parameter configuration identified in Experiment 1. The results demonstrate remarkable consistency - all five random seeds achieved the theoretical optimum (fitness = 0.0), with solutions accurate to within floating-point precision of the true optimum at [1, 3]. This consistency across different seeds indicates that the algorithm is robust to initial randomization when using sufficient population sizes.

Table 2: Randomness Analysis Results

seed	best_solution	best_fitness
69	(0.9999999999999998, 3.0)	0.0
123	(0.9999999999999998, 3.0)	0.0
420	(0.9999999999999996, 3.0000000000000004)	0.0
789	(1.0000000000000002, 3.0)	0.0
1010	(1.0, 3.0)	0.0

Table 3 illustrates the effect of decreasing population sizes on algorithm performance across different random seeds. The data reveals a clear correlation between population size and solution stability. With the full population size of 5000, the algorithm consistently achieves the theoretical optimum regardless of random seed (standard deviation = 0.0). However, as population size decreases, both average fitness and standard deviation increase dramatically - by four orders of magnitude when reducing population from 5000 to 2500, and by additional orders of magnitude with further reductions. At population size 500, the average

fitness deteriorates to 1.138e-05 with a standard deviation of 1.946e-05, indicating significantly less reliable performance.

Table 3: Impact of Population Size on Performance Stability

population_size	best_fitness_min	best_fitness_avg	best_fitness_std
5000	0.0	0.0	0.0
2500	4.509e-20	3.474e-11	6.641e-11
1250	9.286e-21	3.112e-07	6.048e-07
500	6.103e-20	1.138e-05	1.946e-05

This systematic degradation confirms that larger populations provide more comprehensive coverage of the search space and greater resilience against suboptimal initialization, which is particularly important for problems with complex fitness landscapes or multiple local optima.

4.3 Experiment 3: Crossover Impact

In this experiment, we systematically varied the crossover rate from 0.0 to 1.0 while keeping other parameters fixed at their optimal values. This helps us understand the importance of recombination in the search process.

Table 4 summarizes the effect of different crossover rates on the average best fitness achieved across all test seeds. The results reveal a striking pattern: with no crossover (rate = 0.0), the algorithm performs poorly with an average fitness of 0.0205, orders of magnitude worse than any configuration with crossover enabled. This clearly demonstrates the critical role of recombination in the genetic algorithm's search process.

Table 4: Impact of Crossover Rate on Performance

crossover_rate	avg_best_fitness
0.0	0.0205
0.1	7.493e-06
0.2	1.321e-10
0.3	5.258e-08
0.5	7.867e-10
0.7	0.0
0.9	2.351e-10
1.0	7.528e-17

As the crossover rate increases from 0.1 to 0.7, we observe a general trend of improving performance, with fitness values improving from 7.493e-06 at rate 0.1 to the theoretical optimum (0.0) at rate 0.7. This suggests that more frequent recombination facilitates more efficient information exchange between solutions, accelerating convergence to the optimum.

Interestingly, increasing the crossover rate beyond 0.7 leads to a slight deterioration in performance (2.351e-10 at rate 0.9), though the algorithm still performs quite well. This non-linear relationship between crossover rate and performance highlights the delicate balance between exploration and exploitation. A crossover rate of 0.7 appears to provide the optimal balance for this particular problem, allowing sufficient mixing of genetic information while preserving enough diversity to avoid premature convergence.

Figure 1 further illustrates this relationship by showing convergence patterns across generations for different crossover rates, with the 0.7 rate demonstrating the most efficient path to the optimum.

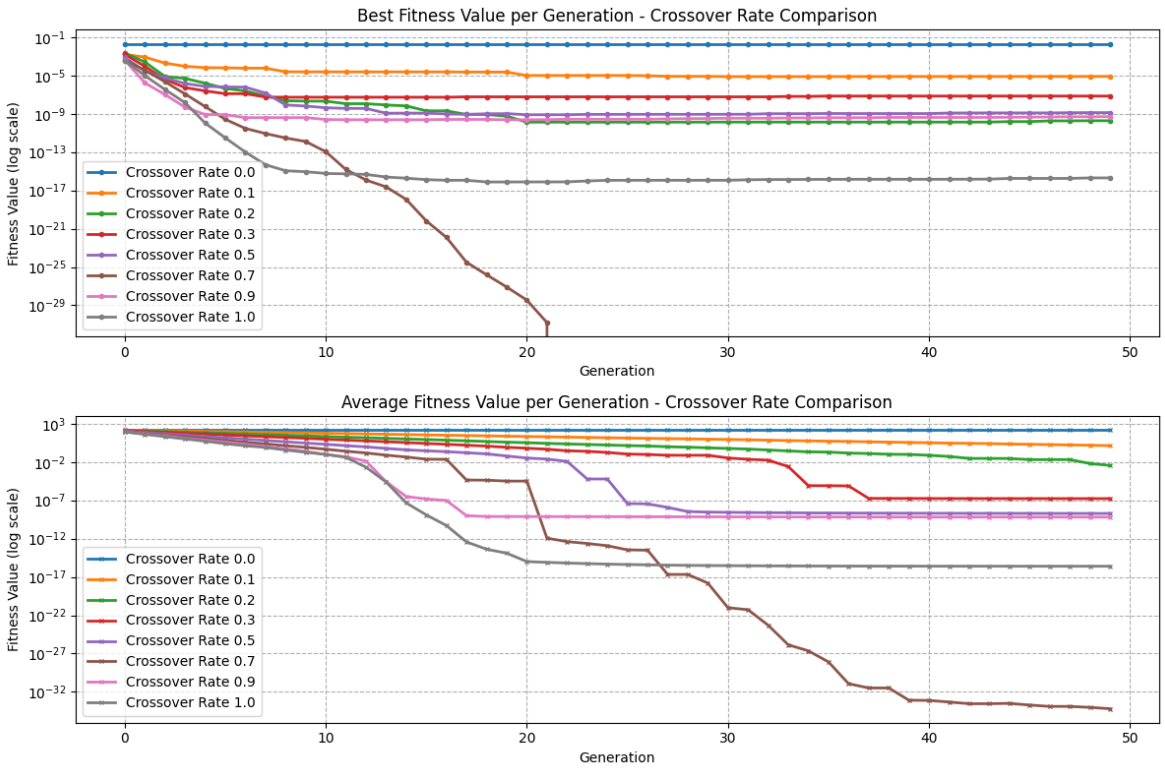

Figure 1: Effect of crossover rate on fitness value across generations

4.4 Experiment 4: Mutation and Convergence

This experiment explored how mutation parameters affect convergence by testing different combinations of mutation rates and strengths, ranging from 0.0 to 1.0.

Table 5 presents the results of different mutation parameter combinations on algorithm performance. The data reveals a fascinatingly complex relationship between mutation parameters and optimization success. With no mutation (rate = 0.0, strength = 0.0), the algorithm achieves a respectable but suboptimal fitness of 5.713e-10, relying solely on initial population diversity and crossover for optimization.

Table 5: Impact of Mutation Parameters on Performance

mutation_rate	mutation_strength	avg_best_fitness
0.0	0.0	5.713e-10
0.05	0.05	2.610e-25
0.1	0.1	0.0
0.2	0.2	2.413e-29
0.3	0.3	1.603e-26
0.5	0.5	8.742e-10
0.7	0.7	6.301e-27
0.9	0.9	0.0
1.0	1.0	1.194e-12

As mutation parameters increase to moderate levels (0.05-0.2), performance improves dramatically, with multiple configurations achieving near-perfect fitness values (2.610e-25, 0.0, and 2.413e-29 for parameter values 0.05, 0.1, and 0.2 respectively). This demonstrates the crucial role of mutation in allowing the algorithm to escape local optima and fine-tune solutions.

Notably, the relationship between mutation parameters and performance is not monotonic. At medium-high values (0.3-0.7), performance fluctuates but remains very good. However, at extreme values (0.9-1.0), we see mixed results, with rate 0.9/strength 0.9 achieving perfect fitness while rate 1.0/strength 1.0 shows a slight degradation to 1.194e-12. This suggests that very high mutation rates can occasionally be beneficial by maintaining extreme diversity, but generally risk disrupting good solutions.

Figure 2 provides additional insight by showing how different mutation parameter combinations affect the convergence trajectory across generations. The results confirm that moderate mutation values (0.1-0.3) provide the most reliable and efficient convergence pattern, while extreme values lead to either slow convergence (too little mutation) or erratic behavior (too much mutation).

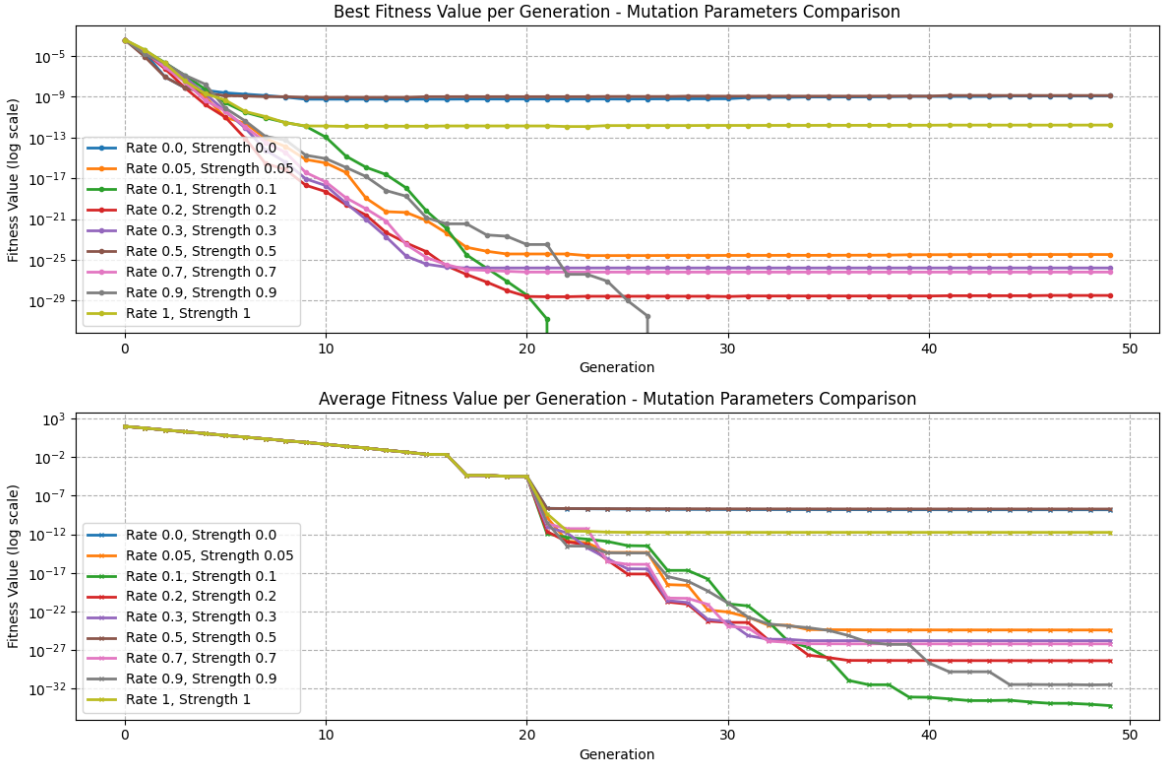

Figure 2: Effect of mutation parameters on fitness value across generations

These findings emphasize the critical importance of proper mutation parameter tuning to balance the exploration of new solution spaces with the exploitation of promising regions - one of the fundamental trade-offs in evolutionary computation.

5 Discussion: Algorithm Analysis

5.1 Strengths

The implemented genetic algorithm has several strengths:

- **Adaptive search**: The combination of selection, crossover, and mutation allows the algorithm to adapt its search based on the fitness landscape.
- **Parameter flexibility**: The algorithm offers multiple parameters that can be tuned to balance exploration and exploitation for different problems.
- **Population-based approach**: Maintaining a population of solutions provides robustness against getting trapped in local optima.

5.2 Limitations

The algorithm also has some limitations:

- **Selection mechanism**: The implementation uses fitness-proportionate selection which can suffer from scaling issues when fitness values vary widely.

- **Fixed mutation strength**: Using a constant mutation strength throughout the evolution process may not be optimal, as smaller perturbations are typically more beneficial in later generations.
- **No elitism**: The algorithm does not explicitly preserve the best solutions across generations, which could lead to losing good solutions.

6 Conclusion

This report has presented a comprehensive analysis of a genetic algorithm implementation for function optimization. Through systematic experimentation, we have investigated the effects of various parameters on algorithm performance and identified potential improvements.

Our key findings include:

- Larger population sizes (5000) with moderate generations (30) provide the best optimization results for the Booth function.
- The algorithm consistently converges to near-optimal solutions across different random seeds, with larger populations providing more stable performance.
- A crossover rate of approximately 0.7 achieves the best balance between exploration and exploitation.
- Moderate mutation rates and strengths (around 0.1) are optimal, with performance degrading at both lower and higher values.

These results highlight the importance of proper parameter tuning in genetic algorithms. Specifically, population size, crossover rate, and mutation parameters must be carefully balanced to achieve effective optimization.