Laboratory 3

Variant 3

Group 3

By Szymon Bedeniczuk and Juan Mallo

Introduction

- *Describe the task.*

The task entails creating a Python program that employs a basic genetic algorithm to optimize a given 2D function, specifically the Bukin function, using Tournament Selection. The Bukin function is defined as

3. Optimize Bukin function using Tournament Selection:

$$f(x, y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10|.$$

Initialize x from range $[-15, 5]$ and y from range $[-3, 3]$.

The genetic algorithm involves implementing mutation using the Gaussian operator and crossover using random interpolation. The crossover formula,

$$(\text{eg: } x_o = \alpha * x_{p1} + (1 - \alpha) * x_{p2},\ y_o = \alpha * y_{p1} + (1 - \alpha) * y_{p2},$$

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

is particularly suitable for real-valued problems.

Furthermore, the program is designed to replace the current population with the newly generated offspring in each iteration.

Experiments

- *Finding genetic algorithm parameters.*

To ensure the reproducibility of our experiments, we set a random seed at the beginning of each execution of the genetic algorithm. In this first task, we fixed the seed to a specific value (e.g., 50), allowing us to reproduce the same results in each run. This is crucial for maintaining consistency in comparing different parameter combinations and ensuring the reliability of our findings.

We conducted an experiment to optimize the parameters of a genetic algorithm aimed at solving the Bukin function. Multiple combinations of algorithm parameters were systematically tested, covering reasonable ranges for each parameter. These ranges were chosen to sufficiently explore the solution space without making the experiments computationally prohibitive. Each parameter combination was evaluated over a fixed number of generations, set at 10 for consistency.

```
TASK 1
Population - Mutation Rate - Mutation Strength - Crossover Rate - Best Solution - Best Fitness
100 - 0.05 - 0.08 - 0.6 - (-5.685876710664152, 0.32317710960777457) - 0.8970788456659513
100 - 0.05 - 0.08 - 0.7 - (-7.013987981988695, 0.49185456192218857) - 0.945157725710924
100 - 0.05 - 0.08 - 0.8 - (-16.184515565820277, 2.61967140737295) - 0.5704834672586473
100 - 0.05 - 0.08 - 0.9 - (-6.173164986268535, 0.3812365263455221) - 0.7747533337761887
100 - 0.05 - 0.1 - 0.6 - (-9.178151214997762, 0.8420535580991906) - 0.5471457339063751
100 - 0.05 - 0.1 - 0.7 - (-6.595413164413346, 0.43455242539844496) - 0.4679035076566136
100 - 0.05 - 0.1 - 0.8 - (-13.194463653659687, 1.7421398643730774) - 0.2859013067688686
100 - 0.05 - 0.1 - 0.9 - (-13.409277355803578, 1.7982406325498483) - 0.7856670434828594
100 - 0.05 - 0.12 - 0.6 - (-7.545244135249882, 0.5694618636602452) - 0.7882545334898986
100 - 0.05 - 0.12 - 0.7 - (-6.575045079275851, 0.4325270890084736) - 0.6665626489721569
```

• • •

```
500 - 0.2 - 0.14 - 0.8 - (-11.46995813789525, 1.315523056360386) - 1.1255810386991634
500 - 0.2 - 0.14 - 0.9 - (-7.874318724916125, 0.6200297625558008) - 2.1770597727760617
500 - 0.2 - 0.16 - 0.6 - (-8.17755352419641, 0.6686625353666364) - 1.2483668759379272
500 - 0.2 - 0.16 - 0.7 - (-9.237179054231085, 0.8534529399032269) - 0.7065336285241233
500 - 0.2 - 0.16 - 0.8 - (-6.392017216540136, 0.4086746998092967) - 0.9850711235378479
500 - 0.2 - 0.16 - 0.9 - (-8.933738296316154, 0.798623328670854) - 0.4422268289458529
```

The fitness value obtained for each parameter combination served as the metric for performance evaluation. By analyzing these fitness values, we identified the parameter set that yielded the most promising results for solving the Bukin function using the genetic algorithm:

POPULATION	MUTATION RATE	MUTATION STRENGTH	CROSSOVER RATE	BEST SOLUTION	BEST FITNESS
200	0.1	0.08	0.7	(-6.13632042866655, 0.3765443131887304)	17.9494764174537

These parameters strike a balance between exploration and exploitation, allowing the genetic algorithm to effectively navigate the solution space and converge to a promising solution. The relatively moderate mutation rate and strength ensure sufficient exploration, while the crossover rate facilitates the exchange of genetic information among individuals. Overall, this parameter combination demonstrates robust performance in optimizing the Bukin function.

- *Randomness in genetic algorithm.*

In this second task, we investigated the impact of randomness and population size on the performance of the genetic algorithm. First, we ran the algorithm with the best parameters identified in Task 1 using five different random seeds. This allowed us to assess the robustness of the algorithm and observe how different initializations affect the solutions obtained. We then calculated the average fitness value across all seeds, along with its standard deviation, to quantify the variability in performance.

```
TASK 2
Seeds - Best Solution - Best Fitness
20 - (-10.744379835910447, 1.154585681087639) - 0.7655259876061095
40 - (-10.87859194460724, 1.1834422096372552) - 4.48717212710859
60 - (-11.189288615247134, 1.2519365356431738) - 1.2199007066225154
80 - (-7.473832598040866, 0.5585177727727912) - 1.2120649663895733
100 - (-7.928584781673301, 0.6279527989683961) - 0.3827660621346371
Average values of fitness is: 0.7982904650372662
Standard derivation of fitness is: 0.0
```

Next, we reran the algorithm with decreasing population sizes, specifically around 50%, 25%, and 10% of the original population size. By doing so, we aimed to evaluate how reducing the population size affects the quality of solutions generated by the algorithm.

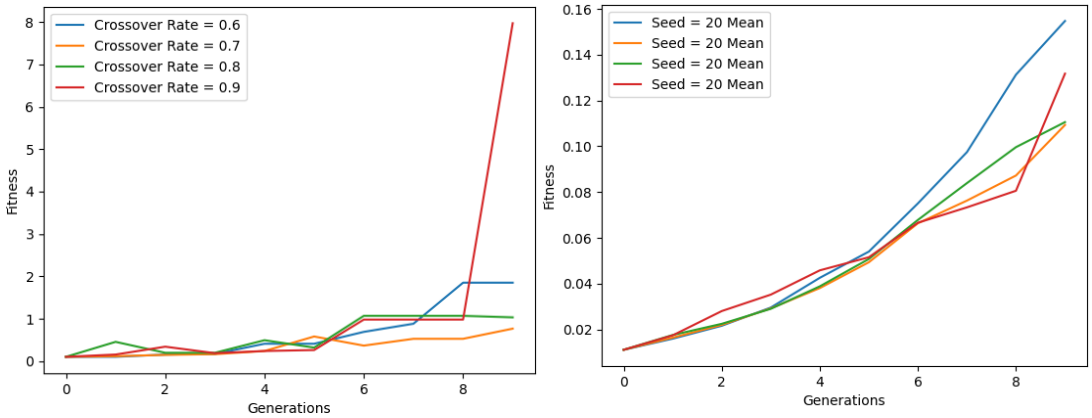
```
New population sizes - Best Solution - Best Fitness
0.5 - (-10.74019517949409, 1.153579141778783) - 1.266121076911359
0.25 - (-6.024426853567226, 0.3637452868311276) - 0.3469256916560346
0.1 - (-10.191244716197865, 1.0350091013147382) - 0.16648445613726728
```

The results showed variations in the optimal solutions and their associated fitness values across different random seeds, indicating the influence of randomness on algorithm performance. Additionally, reducing the population size led to notable changes in the quality of solutions obtained, underscoring the importance of population size in determining algorithm effectiveness.

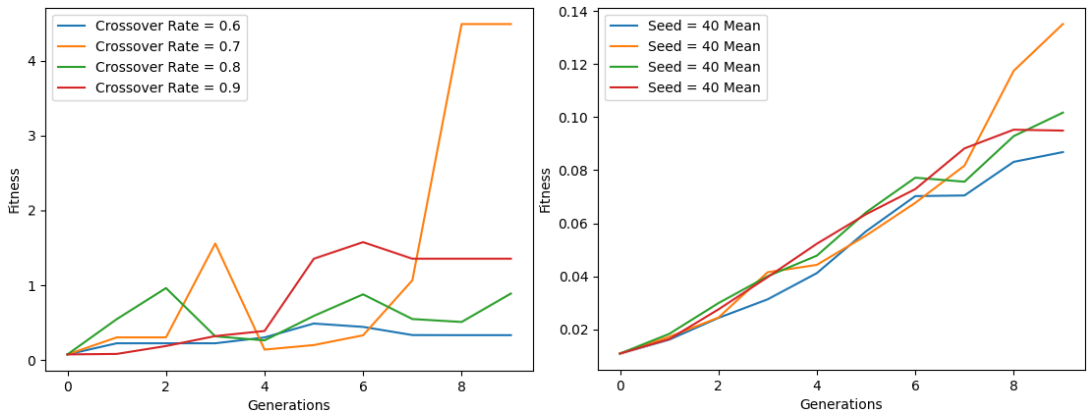
- *Crossover impact.*

In Task 3, we explore the impact of varying the crossover rate on the performance of the genetic algorithm. By running the algorithm multiple times with different crossover rates and plotting the best fitness values across generations, we aim to understand how this parameter affects the algorithm's ability to find optimal solutions. This analysis provides valuable insights into the role of crossover in the genetic algorithm and its influence on solution quality and convergence dynamics.

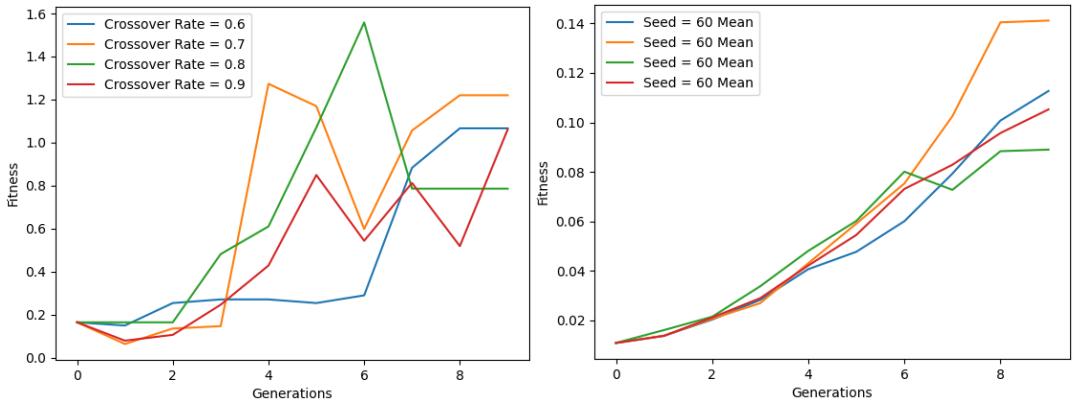
seed = 20



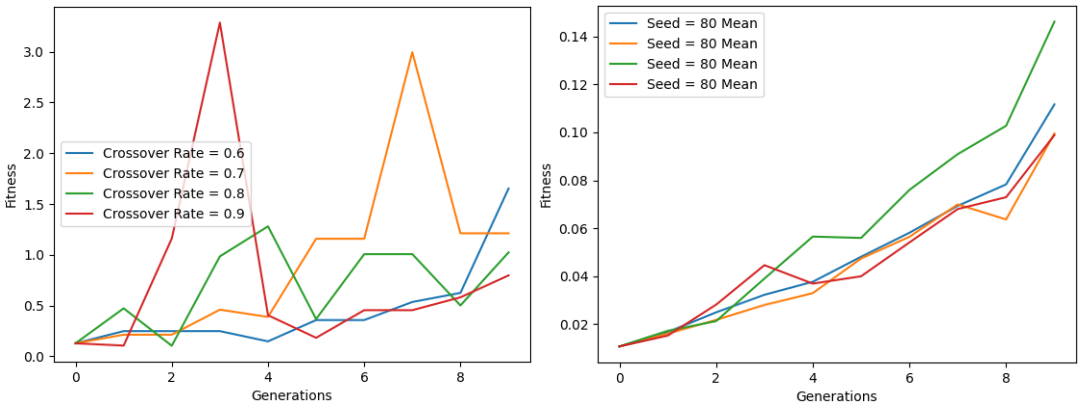
seed = 40



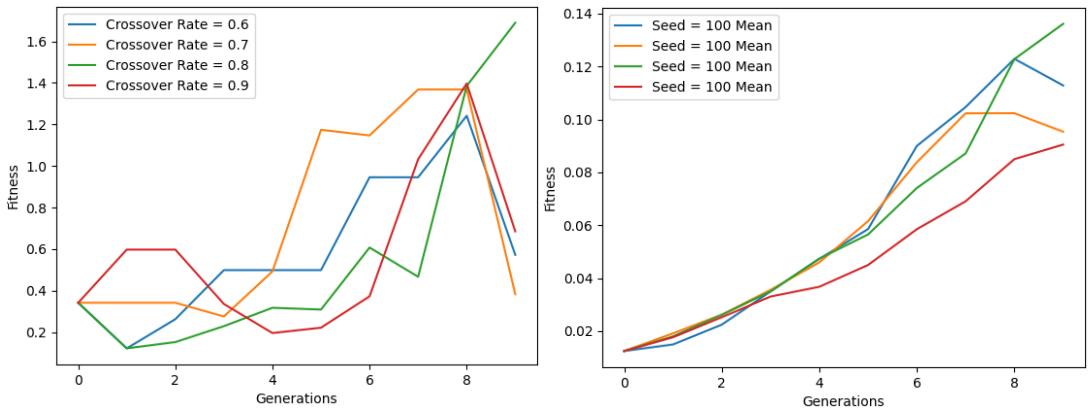
seed = 60



seed = 80



seed = 100



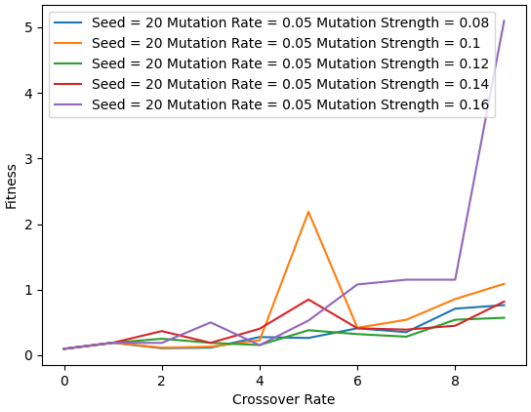
The first graph allows to observe how the best individuals in the population evolve as the algorithm progresses through generations, giving an idea of the quality of the optimal solutions found based on the crossover rate. On the other hand, the second graph shows how the average performance of the population changes as the algorithm unfolds, providing insights into the convergence and stability of the algorithm with different crossover rates.

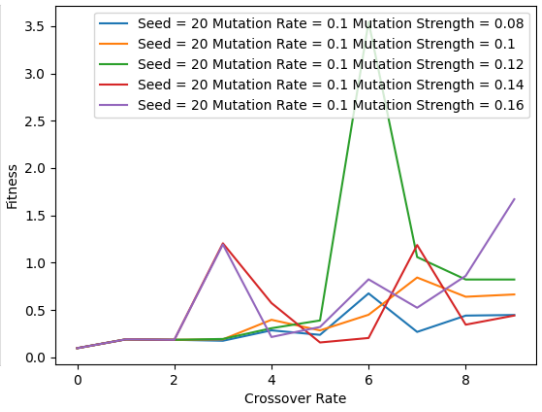
By observing how the fitness values change over generations for each crossover rate, we can gain insights into how the choice of crossover rate influences the algorithm's ability to converge towards optimal solutions.

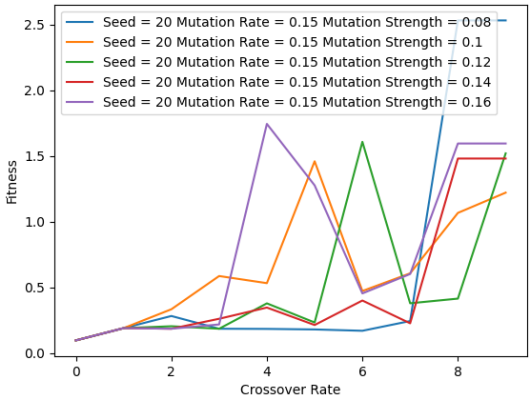
- *Mutation and the convergence.*

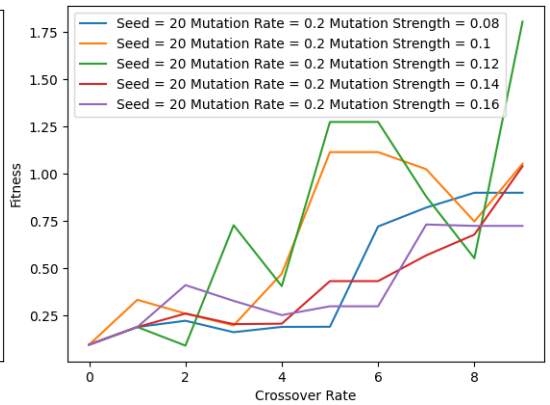
In this final task, we analyze the impact of varying the mutation rate and strength on the performance of the genetic algorithm. In each plot, we can observe the difference mutation strength makes. Then, between plots of the same seed, we can observe the importance of the mutation rate.

seed = 20

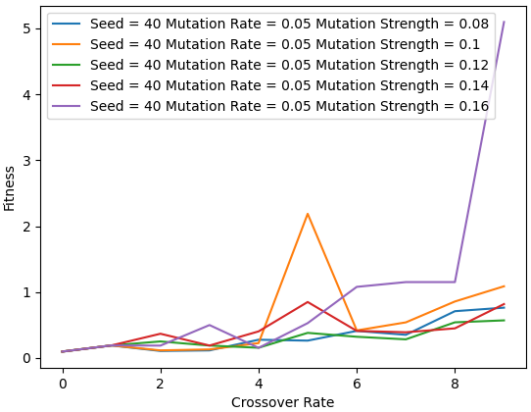


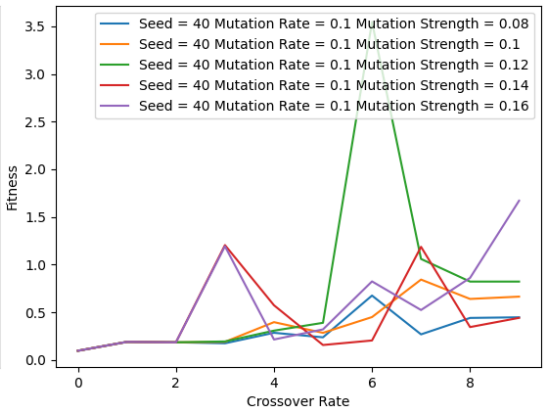


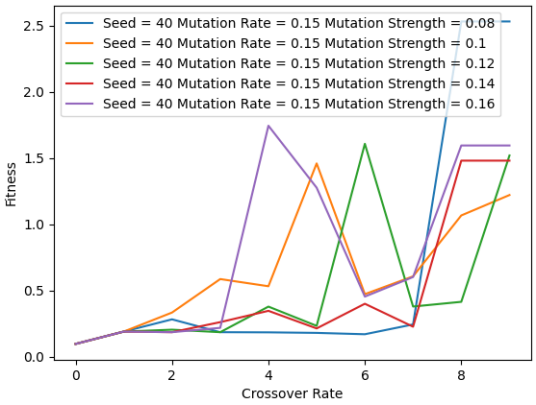


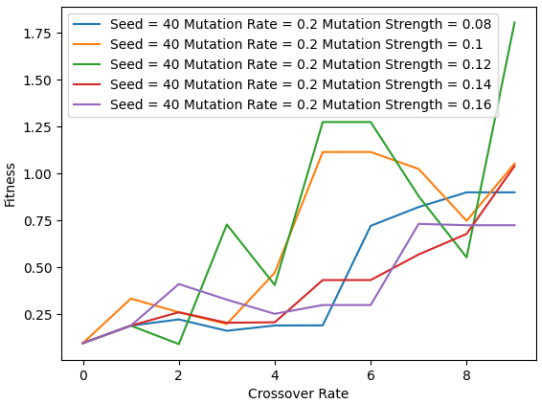


seed = 40



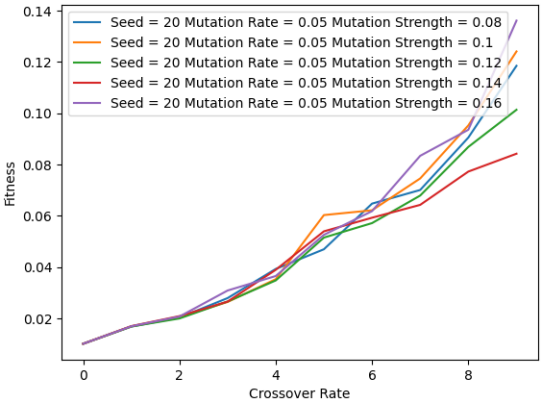


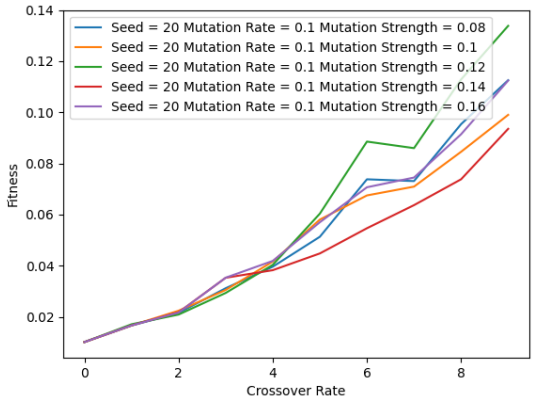


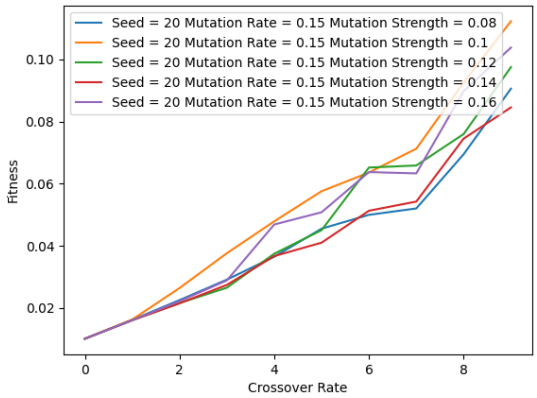


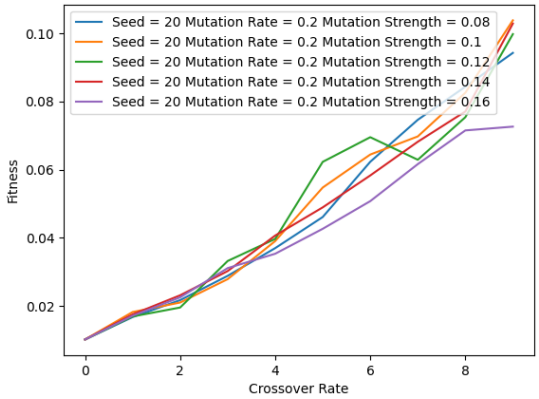
• • •

seed = 20

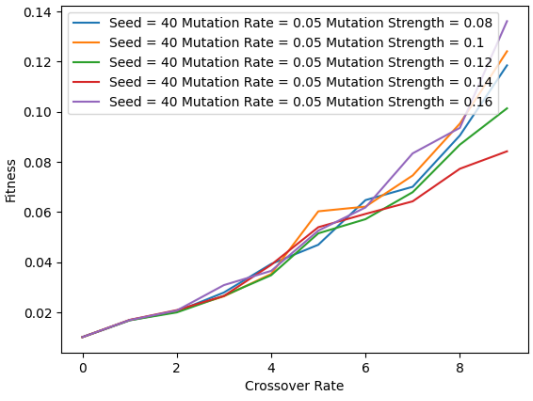


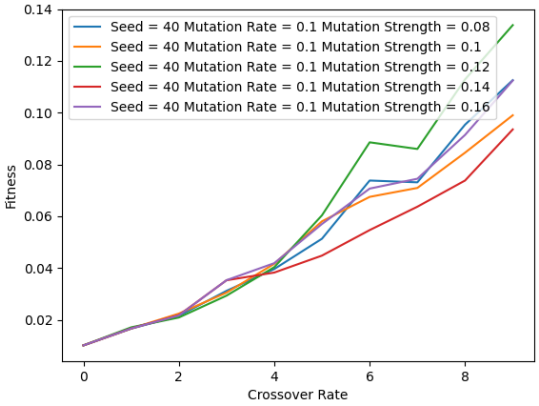


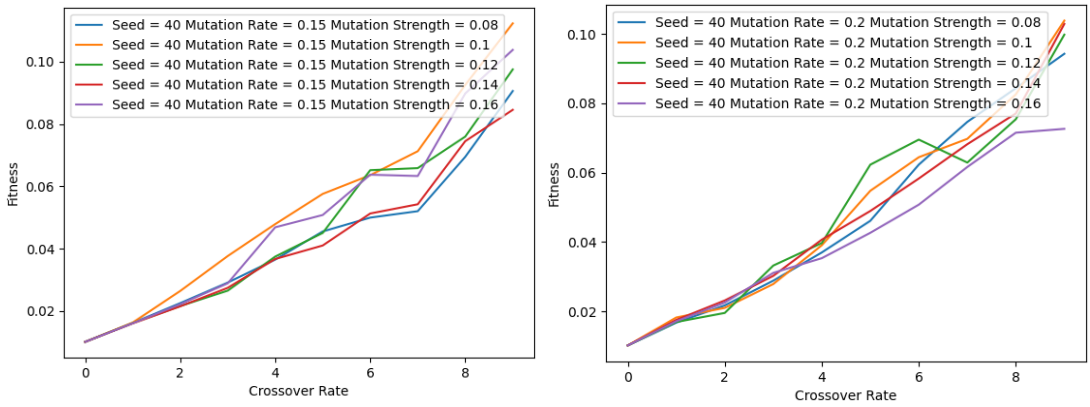




seed = 40







• • •

These last plots represent the average fitness across the generations, the previous ones where the best fitness across the generations.

Conclusion

- *Summary of the findings from experiments 1-4.*

In conclusion, the experiments conducted with the genetic algorithm provided valuable insights into its behavior and performance across various parameter configurations. The choice of algorithm parameters significantly affects the performance and convergence behavior of the genetic algorithm. Randomness, as introduced by random seeds, plays a crucial role in solution variation and algorithm stability.

Crossover rate influences the exploration-exploitation trade-off, affecting both solution quality and convergence speed. Mutation rate and strength impact the algorithm's ability to explore the search space and converge to optimal solutions.

Overall, these experiments underscored the complexity of the genetic algorithm and the importance of parameter tuning for achieving effective optimization. Further research could focus on more sophisticated parameter optimization techniques and their application to real-world problems.