07.04.2025

**Laboratory 3**

Variant 4 – Optimize the Booth function using Roulette Wheel Selection

Lab group 105

*Group 9 – Aditya Kandula, Martyna Wielgopolan*

1. **Introduction**

In this laboratory, we implemented a Genetic Algorithm (GA) to solve a continuous optimization problem defined by the **Booth function**, a well-known benchmark in mathematical optimization. Our task was to use the GA to find the global minimum of the function in a two-dimensional space.

The Booth function is defined as:

**f(x,y) = (x+2y−7)2 + (2x+y−5)2**

It has a known global minimum at (x,y) = (1,3), where the function value is

f(x,y) = 0.

To solve this problem, we used a basic real-valued genetic algorithm with **Roulette Wheel Selection** for choosing parents, **random interpolation for crossover**, and **Gaussian mutation**. The main goal of the project was to study how the genetic algorithm behaves under different parameters and how randomness, crossover, and mutation influence the convergence.

1. **Implementation**

The genetic algorithm was implemented in Python using the provided template structure. The entire solution is modular, with individual files responsible for different components: function definitions, genetic algorithm logic, execution, and experiments.

**Core Algorithm Implementation**

The algorithm itself is implemented in the **GeneticAlgorithm class**, located in **genetic\_algorithm.py**. It consists of the following components:

* Population Initialization: Randomly generates individuals (real-valued pairs *x,y*) within the function’s valid domain.
* Fitness Evaluation: The Booth function is used to evaluate each individual. Lower values indicate better fitness.
* Selection: Roulette Wheel Selection is used. Individuals with better fitness (i.e., lower function values) have a higher chance of being selected to produce offspring.
* Crossover: Real-valued crossover is performed using random interpolation. Two parents generate offspring by combining their genes weighted by a random factor.
* Mutation: Gaussian noise is added to the genes of selected individuals based on a mutation rate and mutation strength. The resulting values are clamped to remain within the valid range.
* Evolution Loop: The GA runs for a fixed number of generations. In each generation, the algorithm evaluates the current population, selects parents, applies crossover and mutation, and forms a new generation.

During execution, we track:

* Best fitness (minimum function value found in each generation)
* Average fitness (mean function value of the whole population)

These metrics are used later for analysis and plotting.

**Experiment Execution and Automation**

To fulfill the experimental requirements from the lab, several scripts were created to automate testing and data collection:

**1. main.py**

Used for testing individual runs with fixed parameters. It generates a plot showing convergence (best and average fitness over time, in log scale) and prints the best solution found.

**2. experiment1.py**

This script tests various combinations of population size, mutation rate, mutation strength, crossover rate, and number of generations. The results (best solution and fitness for each combination) are saved in a text file.

**3. experiment2.py**

This script evaluates the impact of randomness on convergence by:

* Running the GA with the same parameters and different random seeds (5 in total)
* Calculating the average fitness and standard deviation
* Repeating the same runs with reduced population sizes (50%, 25%, 10%) All results are saved in experiment\_2\_randomness.txt.

**4. experiment3.py**

This script varies the crossover rate (e.g. 0.5, 0.7, 0.9, 1.0) and for each rate:

* Runs the algorithm with 3 seeds
* Averages the results
* Saves convergence plots (.png) to visualize how crossover affects optimization

**5. experiment4.py**

This script varies both the mutation rate and mutation strength across several combinations. Each setup is averaged over 3 seeds and plotted as:

* Best fitness over time
* Average fitness over time The resulting plots are used to evaluate how mutation settings influence convergence behaviour.

All experiments use the same base structure of the GA, and only the parameters (and seed) are changed between runs. This makes it easier to isolate the effect of each parameter.

1. **Discussion**

This section presents and analyzes the results obtained from a series of experiments designed to evaluate the performance of a genetic algorithm applied to the minimization of the Booth function. The experiments examined the effects of parameter settings, randomness, crossover rate, and mutation characteristics on the convergence behavior and final solution quality of the algorithm.

**Experiment 1: Parameter Tuning**

The objective of this experiment was to identify a suitable configuration of parameters that leads to reliable convergence to the global minimum of the Booth function. The following parameters were varied:

* Population size: 10, 25, 50
* Mutation rate: 0.1, 0.3
* Mutation strength: 0.05, 0.1
* Crossover rate: 0.7, 0.9
* Random seed: 1, 42, 123
* Generations: 100

Each combination was tested using the script experiment\_runner.py, and the final best fitness values were recorded. The results showed that higher population sizes (particularly 50) generally led to better convergence. Mutation rates of 0.3 and strengths of 0.1 performed well across most combinations, and a crossover rate of 0.9 yielded the most consistent improvements.

The best-performing configuration was:

* Population size: 50
* Mutation rate: 0.3
* Mutation strength: 0.1
* Crossover rate: 0.9
* Generations: 100

This configuration consistently produced results close to the global minimum of the Booth function, with best fitness values on the order of 10−4 to 10−5.

**Experiment 2: Randomness in the Genetic Algorithm**

Using the best configuration from Experiment 1, the algorithm was executed five times with different random seeds (1, 42, 100, 2024, 999). The results showed some variability in performance depending on the seed. While several runs converged well with final fitness values below 10−3, others showed significantly worse results. This confirms that random initialization and stochastic operations (selection, crossover, mutation) can affect performance, especially in runs with lower population diversity.

The average final fitness across seeds was 0.244, with a standard deviation of 0.444, indicating moderate variability.

To assess the effect of population size on convergence stability, the same configuration was rerun with reduced populations:

|  |  |  |
| --- | --- | --- |
| **Population Size** | **Mean Fitness** | **Standard Deviation** |
| 25 (50%) | 0.371 | 0.724 |
| 12 (25%) | 1.275 | 1.696 |
| 5 (10%) | 25.833 | 48.310 |

As the population size decreased, both the average fitness and the standard deviation increased substantially. This highlights the importance of sufficient population size in maintaining diversity and ensuring robust convergence.

**Experiment 3: Crossover Rate Impact**

This experiment investigated the impact of varying the crossover rate on the algorithm's performance. The algorithm was executed with crossover rates of 0.5, 0.7, 0.9, and 1.0. For each rate, three runs with different seeds were averaged, and the evolution of both best and average fitness was recorded.

The results indicated that a crossover rate of 0.5 resulted in slow convergence and poor final fitness values. Increasing the rate to 0.7 improved performance, but convergence remained relatively slow. The crossover rate of 0.9 showed the most balanced behavior: fast convergence, stable population dynamics, and low final fitness. A rate of 1.0 also performed well but introduced slightly more variability between generations, likely due to excessive recombination.

These results suggest that a high but not maximal crossover rate (e.g., 0.9) is optimal, supporting both exploration and exploitation during the evolutionary process.

**Experiment 4: Mutation Rate and Strength**

In this experiment, the influence of mutation rate and mutation strength on convergence behavior was studied. Nine combinations were tested:

* Mutation rates: 0.1, 0.3, 0.5
* Mutation strengths: 0.05, 0.1, 0.2

Each combination was averaged over three random seeds. The following table summarizes the approximate final best fitness values and observed stability:

|  |  |  |  |
| --- | --- | --- | --- |
| **Mutation Rate** | **Strength** | **Final Best Fitness (approx.)** | **Stability** |
| 0.1 | 0.05 | ~1.1 | Very stable |
| 0.1 | 0.1 | ~0.9 | Stable |
| 0.1 | 0.2 | ~0.15 | Stable |
| 0.3 | 0.05 | ~0.5 | Stable |
| 0.3 | 0.1 | ~0.3 | Stable |
| 0.3 | 0.2 | ~0.001 | Best overall result |
| 0.5 | 0.05 | ~0.4 | Semi-stable |
| 0.5 | 0.1 | ~0.02 | Slight instability |
| 0.5 | 0.2 | ~0.001 | Unstable (oscillating) |

Low mutation rates (e.g., 0.1) with low strengths were stable but showed limited exploration ability and slow convergence. High mutation rates (0.5) with high strengths allowed the population to escape local optima but led to instability in the later stages.

The best results were achieved with a mutation rate of 0.3 and strength of 0.2. This configuration produced the fastest convergence and lowest final fitness values while maintaining acceptable stability. It represents a good balance between exploration and exploitation.

**General Observations**

Across all experiments, the genetic algorithm consistently found solutions very close to the known global minimum of the Booth function, provided that the parameters were properly tuned. Key takeaways include:

* Parameter tuning is essential: population size, mutation, and crossover settings must be balanced to achieve good results.
* Randomness introduces variability, but sufficient population size mitigates its negative effects.
* Mutation and crossover both strongly affect convergence, and moderate values generally perform best.
* The use of averaged plots and results over multiple seeds provides more reliable insights into parameter behavior.

These findings are consistent with the theoretical expectations of evolutionary algorithms and highlight the importance of empirical evaluation and parameter sensitivity analysis.

1. **Conclusions**

In this project, a genetic algorithm was implemented to minimize the Booth function, using a real-valued representation and Roulette Wheel Selection. The implementation included core evolutionary mechanisms such as random interpolation for crossover and Gaussian mutation, allowing for continuous optimization in two-dimensional space.

Through a structured set of experiments, we examined how the algorithm's behavior is influenced by different parameter configurations, including population size, mutation rate, mutation strength, and crossover rate. Additionally, we evaluated the impact of randomness by repeating runs with multiple seeds and varying population sizes.

The experiments demonstrated that:

* The algorithm is capable of reliably converging toward the global minimum of the Booth function, provided that parameters are carefully tuned.
* A population size of 50, crossover rate of 0.9, mutation rate of 0.3, and mutation strength of 0.2 yielded the most effective and stable results.
* Low mutation rates or small populations led to premature convergence and poor exploration, while overly aggressive mutation introduced instability.
* Averaging results over multiple seeds was necessary to ensure robustness and interpretability of results.

Overall, the findings highlight the importance of empirical parameter analysis when applying genetic algorithms to continuous optimization problems. The modular implementation structure also allows for further experimentation with different objective functions or selection strategies, making it a flexible basis for future work.