Atefe Rajabi (40230563)

1.Libreries

2.Functions

3.Setups

4.Plots

4.Comparison

5.Other Map

1.Libraries

1. numpy (import numpy as np):

- In this code, numpy is used for various array operations, such as creating and manipulating the surface roughness map and performing mathematical operations in the genetic algorithm.

2. random (import random):

- In this code, it is used for various random operations, such as initializing the population with random individuals, selecting individuals for crossover and mutation, and determining crossover points.

3. matplotlib.pyplot (import matplotlib.pyplot as plt):

- In this code, matplotlib.pyplot is used for visualizing the surface roughness map, plotting the fitness performance over generations, and visualizing the best solution found by the genetic algorithm.

4. RoadAlignment (from RoadAlignment import Road_Alignment):

- This class likely contains the genetic algorithm implementation for solving the road alignment problem.

Genetic Algorithm

1. Initialization of Population:
    - The initial population is created with individuals represented as lists of coordinates.
2. Fitness Function:
    - The fitness function evaluates the cost of a road alignment based on the surface roughness of the terrain and the cost of building roads or bridges.
3. Parent Selection:
    - Two parent selection methods are implemented: tournament selection and Boltzmann selection.
4. Crossover:
    - Three crossover methods are implemented: uniform crossover, arithmetic crossover, and one-point crossover.
5. Mutation:
    - Two mutation methods are implemented: uniform mutation and inversion mutation.
6. Survival Selection:
    - Elitism is used to select a certain percentage of the best individuals to pass unchanged to the next generation.
7. Termination Criteria:
    - The algorithm terminates if either convergence is detected or if there's no improvement in the best fitness values for a certain number of generations.
8. Visualization:
    - The map roughness and the road alignment represented by a chromosome can be visualized.

2.Functions

1. __init__

- Input: Various parameters for the genetic algorithm, such as population size, crossover and mutation probabilities, etc.
- Output: None

- Functionality: Initializes the parameters and settings for the genetic algorithm, allowing for customization of various aspects such as population size, crossover and mutation probabilities, and termination criteria.

2. initialize_population

- Functionality: Creates an initial population of individuals, where each individual represents a potential road alignment. The population is generated based on the dimensions of the given surface roughness map.

3. calculate_fitness

- Functionality: Evaluates the fitness of an individual (road alignment) by considering the cost associated with constructing roads. The cost is determined by the surface roughness values and the distances between consecutive coordinates.

4. parent_boltzmann_selection

- Functionality: Implements parent selection using the Boltzmann selection method. The probability of selecting each individual as a parent is proportional to the exponential of the negative normalized fitness.

5. parent_tournament_selection

- Functionality: Implements parent selection using the tournament selection method. A set of individuals is randomly sampled, and the one with the lowest fitness is selected as a parent. This process is repeated to select a second parent.

6. uniform_crossover

- Functionality: Performs uniform crossover between two parents to produce two offspring. For each gene (coordinate), there is a probability of exchanging values between parents.

7. arithmetic_crossover

- Functionality: Performs arithmetic crossover between two parents. A weighted average of the y-coordinates is calculated for each gene to generate offspring.

8. one_point_crossover

- Functionality: Performs one-point crossover between two parents. A random crossover point is chosen, and the genes beyond that point are exchanged between parents to produce offspring.

9. uniform_mutate

- Functionality: Applies uniform mutation to a chromosome. There is a probability of mutating each gene, where the y-coordinate may be adjusted randomly within a certain range.

10. inversion_mutation

- Functionality: Applies inversion mutation to a chromosome. Randomly selects two points and inverts the order of the corresponding genes, potentially introducing structural changes in the road alignment.

11. generate_offspring

- Input: Number of offspring to generate

- Output: List of generated offspring

- Functionality: Generates offspring using the selected parent individuals and the specified crossover and mutation methods. The number of offspring is determined by the population size.

12. elitism_survival_selection

- Input: Fitness values, generated offspring, number of elite individuals

- Output: New generation

- Functionality: Selects individuals for the next generation using elitism, where a certain percentage of the fittest individuals from the current generation are preserved. The remaining individuals come from the generated offspring.

13. terminate

- Functionality: Checks if the termination criteria are met. The algorithm terminates if either convergence is detected or there has been no improvement in the best fitness values for a specified number of generations.

14. has_converged

- Functionality: Checks if the algorithm has converged by analyzing the average changes in best fitness values over recent generations. Convergence is declared if the average change falls below a specified threshold.

15. no_improvement

- Functionality: Checks if there has been no improvement in the best fitness values for a specified number of generations, indicating a potential stagnation in the optimization process.

16. extract_solution

- Functionality: Identifies the best individual (chromosome) from the population based on the lowest fitness value. This best individual represents the optimal road alignment solution found by the genetic algorithm.

17. visualize_map

- Functionality: Visualizes the given surface roughness map using a heatmap, where darker colors represent higher levels of roughness.

18. visualize_chromosome

- Functionality: Visualizes a given chromosome (road alignment) on a plot. Roads are depicted in black, and tunnels or bridges are represented in red.

19. genetic_algorithm

- Functionality: Executes the main genetic algorithm loop. It initializes the population, evolves it over multiple generations, and terminates based on convergence or lack of improvement criteria. It returns the last generation, average fitness values, and best fitness values.

20. visualize_solution

- Functionality: Visualizes the best solution found by the genetic algorithm, including the corresponding road alignment on the surface roughness map.

21. plot_fitness_performance

- Functionality: Plots the fitness performance over generations, showing the trend of average and best fitness values. This helps in analyzing the optimization progress throughout the algorithm's execution.

---

3. Setups:

Differences between Setups:

- Crossover Method:
  - Setup 1 uses Uniform Crossover.
  - Setup 2 uses Arithmetic Crossover.
  - Setup 3 uses One-Point Crossover.
  - Setup 4 uses Arithmetic Crossover.
  - ….

- Mutation Method:
  - Setup 1 uses Uniform Mutation.
  - Setup 2 uses Uniform Mutation.
  - Setup 3 uses Uniform Mutation.
  - Setup 4 uses Inversion Mutation.
  - ……

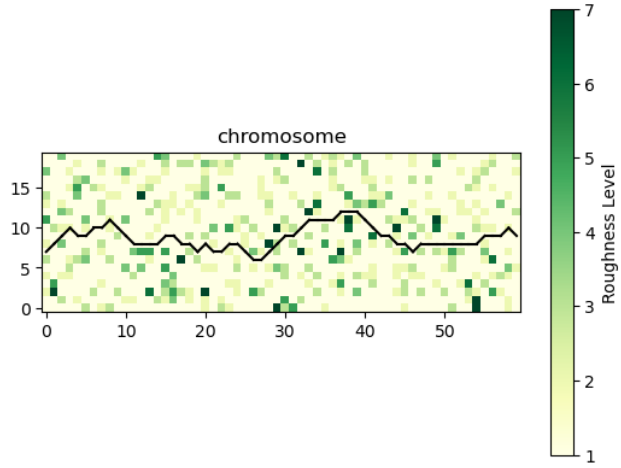
- Parent Selection Method:
  - All setups use Tournament Parent Selection.
  - The Boltzmann is implemented but not used in the provided setups.

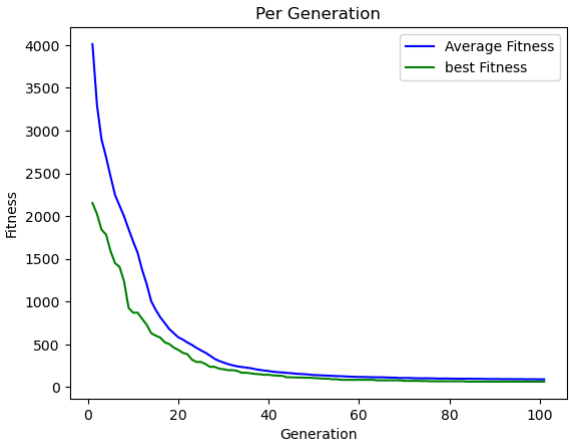
---

4. Plots

1.Setup 1: uniform crossover, uniform mutation, tournament parent_selection

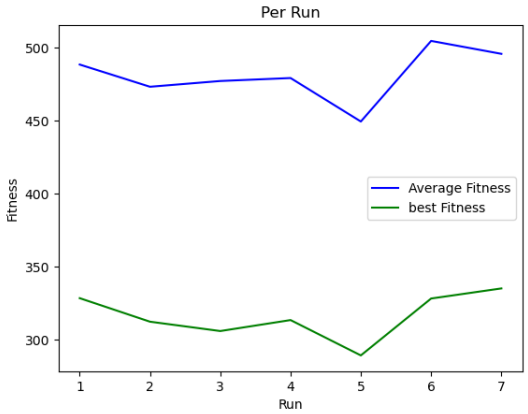
best result in 7 different runs:



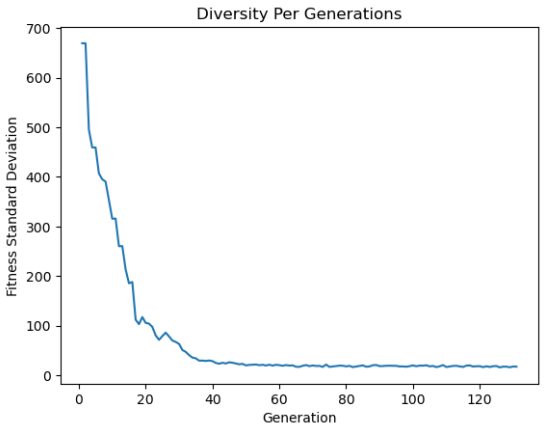


Fitness of this solution is: 64

Fitness Performance per run:

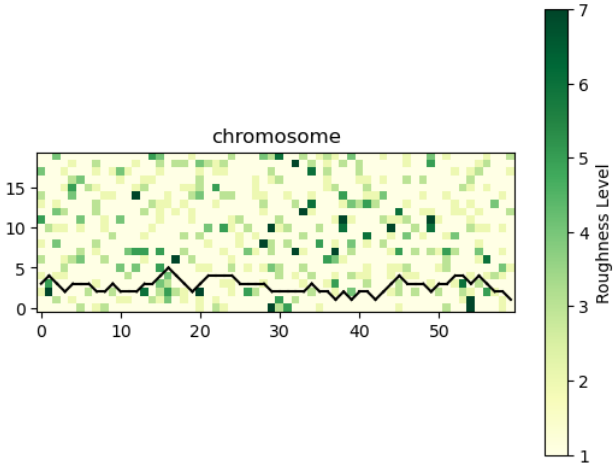


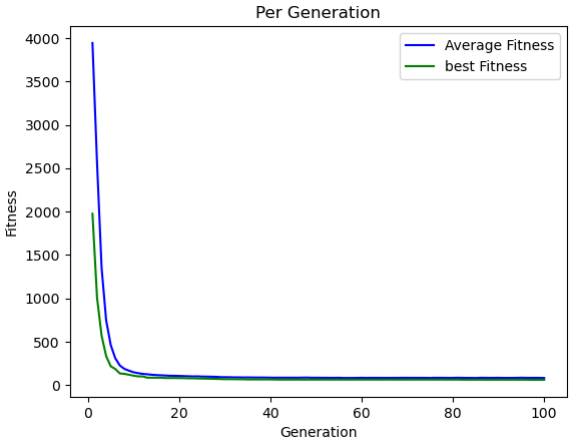
Fitness standard deviation per generation:



2.Setup 2: arithmetic crossover - uniform mutation – tournament parent_selection

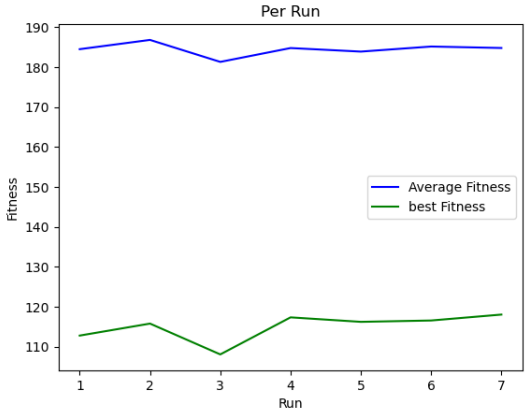
best result in 7 different runs:



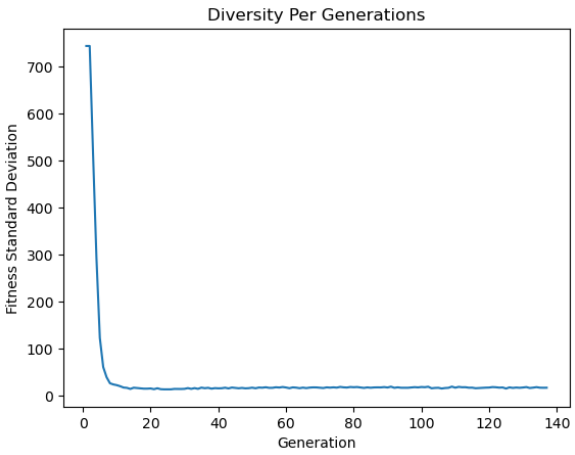


Fitness of this solution is: 61

Fitness Performance per run:

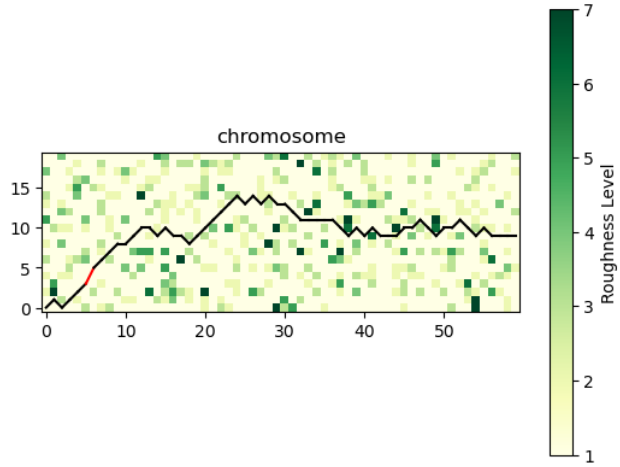


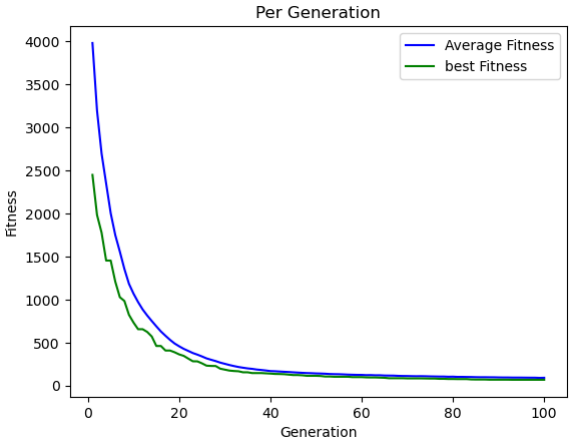
Fitness standard deviation per generation:



3.Setup 3: one-point crossover - uniform mutation – tournament parent_selection

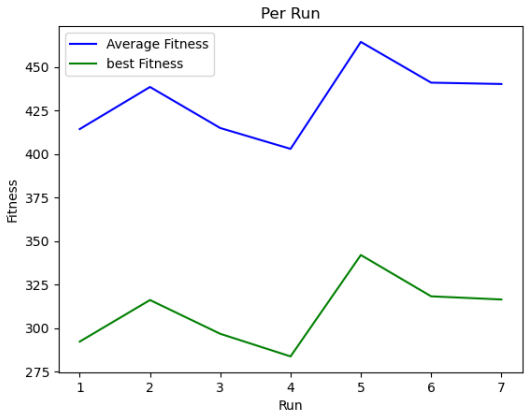
best result in 7 different runs:



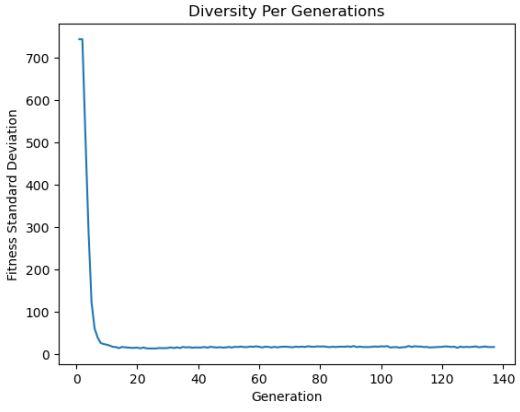


Fitness of this solution is: 68

Fitness Performance per run:

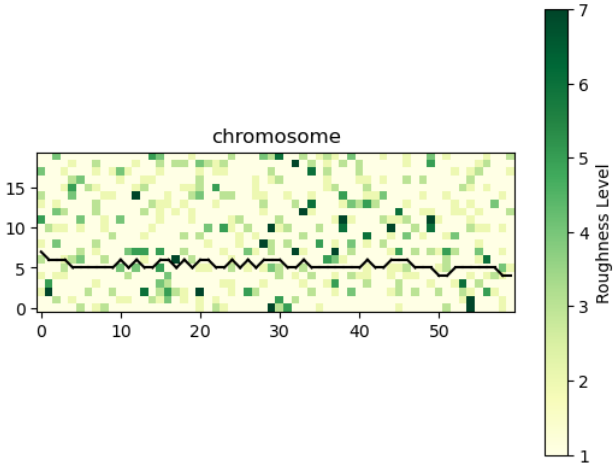


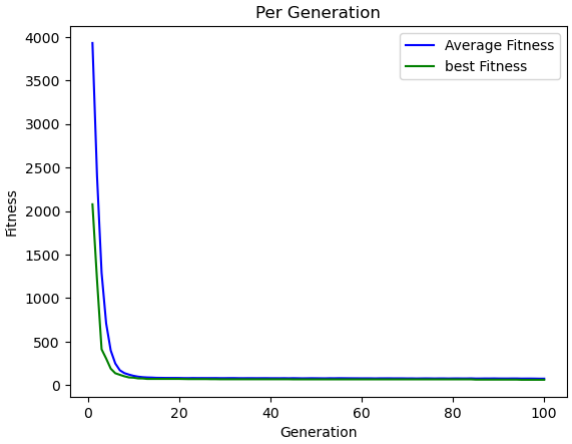
Fitness standard deviation per generation:



4.Setup 4: arithmetic crossover - inversion mutation – tournament

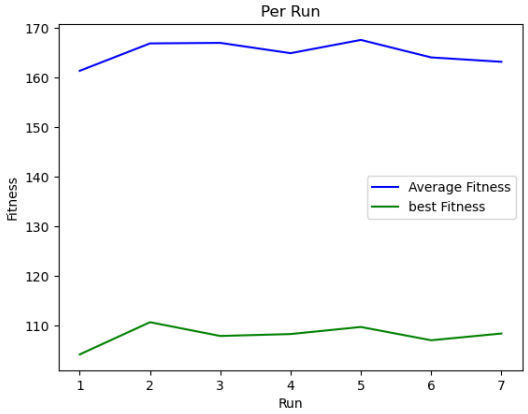
best result in 7 different runs:



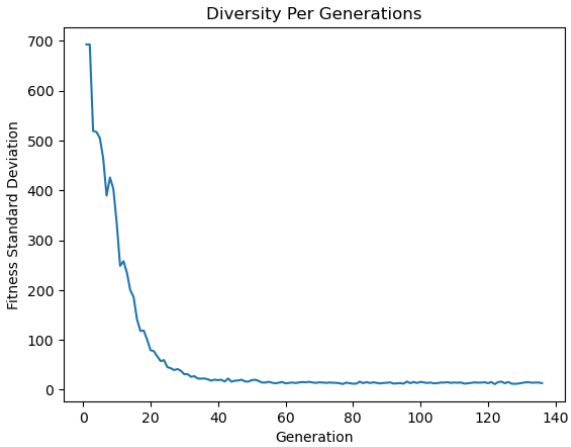


Fitness of this solution is: 62

Fitness Performance per run:



Fitness standard deviation per generation:



5.Comparison:

Here are some observations and comparisons based on the different setups you implemented for the road alignment problem:

Setup 1: Uniform Crossover, Uniform Mutation, Tournament Parent Selection

- Parameters:
  - Population Size (`pop_size`): 150
  - Max Generations (`max_generations`): 130
  - Tournament Size (`tournament_size`): 5
  - Crossover Probability (`crossover_probability`): 1
  - Uniform Crossover Probability (`uniform_crossover_probability`): 0.7
  - Mutation Probability (`mutation_probability`): 1
  - Uniform Mutation Probability (`uniform_mutation_probability`): 0.1
  - Elite Percentage (`elite_percentage`): 0.1

- Results:
  - Found solutions with varying fitness values (e.g., 67.0, 69.0, 74.0).
  - Converged within the specified convergence threshold.

Setup 2: Arithmetic Crossover, Uniform Mutation, Tournament Parent Selection

- Parameters:
  - Population Size: 200
  - Max Generations: 200

  - Tournament Size: 5
  - Crossover Probability: 1
  - Uniform Crossover Probability: 0.6
  - Mutation Probability: 1
  - Uniform Mutation Probability: 0.1
  - Elite Percentage: 0.1

- Results:
  - Starting point was the same for different runs.
  - Found solutions with varying fitness values (e.g., 64.0, 68.0, 63.0).
  - Converged within the specified convergence threshold.

Setup 3: One-Point Crossover, Uniform Mutation, Tournament Parent Selection

- Parameters:
  - Population Size: 150
  - Max Generations: 250
  - Tournament Size: 5
  - Crossover Probability: 0.8
  - Uniform Crossover Probability: 0.6
  - Mutation Probability: 1
  - Uniform Mutation Probability: 0.1
  - Elite Percentage: 0.2

- Results:

  - Found solutions with varying fitness values (e.g., 68.0, 72.0, 78.0).
  - Converged within the specified convergence threshold.

Setup 4: Uniform Crossover, Inversion Mutation, Tournament Parent Selection

- Parameters:
  - Population Size: 150
  - Max Generations: 150
  - Tournament Size: 5
  - Crossover Probability: 1
  - Uniform Crossover Probability: 0.6
  - Mutation Probability: 0.1
  - Uniform Mutation Probability: 0.1
  - Elite Percentage: 0.05

- Results:
  - Found solutions with varying fitness values (e.g., 66.0, 60.0, 68.0).
  - Converged within the specified convergence threshold.

Setup 5: Arithmetic Crossover, Inversion Mutation, Tournament Parent Selection

- Parameters:
  - Population Size: 200
  - Max Generations: 160
  - Tournament Size: 5

  - Crossover Probability: 0.8
  - Uniform Crossover Probability: 0.6
  - Mutation Probability: 0.1
  - Uniform Mutation Probability: 0.1
  - Elite Percentage: 0.1

- Results:
  - Found solutions with varying fitness values (e.g., 64.0, 62.0, 63.0).
  - Converged within the specified convergence threshold.

Setup 6: One-Point Crossover, Inversion Mutation, Tournament Parent Selection

- Parameters:
  - Population Size: 150
  - Max Generations: 250
  - Tournament Size: 5
  - Crossover Probability: 0.8
  - Uniform Crossover Probability: 0.6
  - Mutation Probability: 0.1
  - Uniform Mutation Probability: 0.1
  - Elite Percentage: 0.2

- Results:
  - Found solutions with varying fitness values (e.g., 73.0, 66.0, 67.0).
  - Converged within the specified convergence threshold.

General Observations:

- The choice of crossover and mutation operators seems to have a significant impact on the performance.

- Different setups yield solutions with varying fitness values, indicating the sensitivity of the algorithm to these parameters.

- It's evident that the nature of the crossover and mutation operations affects the convergence behavior and the diversity of solutions.

The result for the map (30, 75) is available in the Jupyter notebook.