Atefe Rajabi (s40230563)

Index:

1. Functionality

This Python code is an implementation of a genetic algorithm to solve the classic N-Queens problem. The N-Queens problem is a combinatorial puzzle that asks for all arrangements of N queens on an N×N chessboard such that no two queens threaten each other. This code aims to find a solution to the problem using a genetic algorithm.

Here's a brief overview of the code:

1. **Initialization (initialization function)**: It starts by generating a population of possible solutions (chessboard configurations) with N queens randomly placed on the board. Each configuration is represented as a permutation of the numbers from 1 to N (indicating the row in which the queen is placed in each column). Following the initialization step, the function proceeds to calculate the fitness of the individuals and stores these fitness values.

2. **Parent Selection (parent selection function)**: In each generation, this function randomly selects 5 solutions from the population, sorts them based on their fitness values in ascending order, and then selects the two best solutions as potential parents for generating offspring in the genetic algorithm. The lower the fitness value, the better.

3. **Fitness Calculation (calculate fitness function)**: The fitness of a chessboard configuration is calculated by counting the number of conflicts (queens threatening each other). A fitness of 0 indicates a valid solution with no conflicts.

4. **Recombination (Recombination function)**: The selected parents are used to create two offspring (new chessboard configurations) through a recombination process. This process involves cutting the parents' configurations at a random point and then exchanging the segments between the parents to create two new configurations (cut phase). The code combines genetic information from both parents to create two offspring: offspring1 combines the left part of parent1 with the right part of parent2, ensuring that all genes are **unique** (fill phase). offspring2

combines the left part of parent2 with the right part of parent1, also ensuring that all genes are unique. This introduces genetic diversity. This process is done with a probability of one. If the probability assumption (pc) changes, the parents have a chance to be selected as offspring, and new offspring are not created by recombination.

5. **Mutation (mutation function)**: The mutation function randomly selects two positions (queens) on the chessboard and swaps their positions with a certain probability (pm). This introduces genetic diversity into the population and can help explore new solutions during the search for a valid N-Queens arrangement.

6. **Survival Selection (survival selection function)**: The function starts by sorting the population based on the fitness values of the individuals. It sorts the individuals in descending order because, in this context, lower fitness values are better. It then removes the two individuals with the highest fitness values from the sorted population.

7. **Check Solution (check solution function)**: After each generation, the code checks if a valid solution (fitness = 0) has been found. If a solution is found, it is displayed on the chessboard.

8. **Fitness Statistics (calculate fitness statistics function and plot fitness statistics function)**: The code records the maximum and average fitness values in each generation and plots these values over generations to visualize the progress of the genetic algorithm. It creates a line plot to visualize the evolution of fitness values across different generations. It is typically used when analyzing the performance of genetic algorithms or other optimization algorithms.

9. **Display chessboard (display chess board function)**: This function is used to display a chessboard with queens placed on it. The chromosome parameter represents a solution to the N-Queens problem, where each element of the chromosome is an integer representing the column in which a queen is placed in the corresponding row.

10. **Main (main function)**: The main function coordinates the execution of the genetic algorithm. It iteratively evolves the population and checks for valid solutions. If a solution is found, it is displayed, and the fitness statistics are plotted. This code provides a demonstration of

how a genetic algorithm can be applied to solve a combinatorial problem like the N-Queens puzzle.

Let's break down what this function does step by step:

2. Initial Population

3. Fitness Evaluation:

   It calculates the fitness of each individual in the population using the `calculate fitness` function. The fitness represents the number of collisions or conflicts on the chessboard.

4. Initial Fitness Statistics:

   The initial fitness values are used to calculate and record statistics. These statistics include the maximum and average fitness of the initial population, and they are stored in the `fitness statistics` list.

5. Check for Solutions in Initial Population:

   It checks if any solutions in the initial population are already optimal solutions (satisfy the N-Queens constraints). If any are found, it prints these solutions and exits the algorithm.

6. Main Loop:

   The core of the genetic algorithm is in the loop that follows:

   - Parent Selection

   - Crossover and Mutation

   - Fitness Evaluation for Offspring:

     Within the loop, the number of fitness evaluations is increased by 2 with each iteration of the loop. The algorithm **has already calculated the population fitness** before, and this calculation is **not repeated** in this part of the loop, which aligns with the context of the genetic algorithm implementation.

   - Population Update:

     The offspring solutions are added to the population.

   - Survival Selection:

     A selection process (survival selection) is applied to choose which individuals will survive and be part of the next generation.

   - Fitness Statistics Update:

It calculates new fitness statistics for the population and appends them to the `fitness statistics` list.

- Solution Check:

It checks if any optimal solutions have been found. If so, it breaks out of the loop.

7. Display Solutions:

If optimal solutions are found, it prints and displays them, including chessboard representations of the solutions.

8. Plotting:

Finally, it uses the `plot fitness statistics` function to create a plot of the fitness statistics over time, showing how the maximum and average fitness values change over the course of the algorithm's execution.

In summary, this `main` function implements a genetic algorithm to solve the N-Queens problem and provides information about the evolutionary process and the quality of solutions found.

2. Libraries:

This code uses the following libraries:

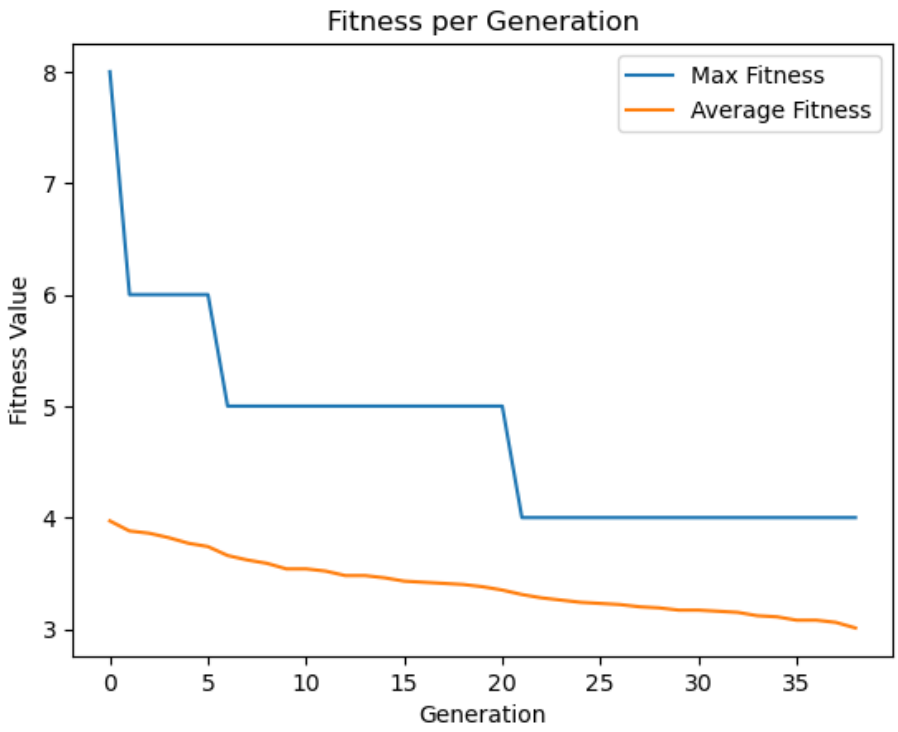
1. **itertools**: It is used to generate all permutations of the queens on the chessboard. This is used during the initialization of the population to create different initial solutions.

2. **random**: The `random` module is used for various randomization tasks, such as shuffling the permutations of queens and for mutation operations.

3. **matplotlib.pyplot**: This library is used for creating plots to visualize the fitness statistics over generations.

These libraries are used to support the implementation of a genetic algorithm to solve the N-Queens problem and visualize the algorithm's progress.

3. Plot fitness function per generation:



The data revealed a consistent decrease in average fitness over generations. This trend suggests that the algorithm was converging towards better solutions.

Similarly, the maximum fitness also exhibited a decreasing pattern, indicating that the algorithm continuously found improved solutions.

The fitness per generation data supports the algorithm's effectiveness in minimizing the objective function.

The decreasing trends in both average and maximum fitness values demonstrate that the algorithm successfully converged towards improved solutions.

4. One possible solution for the problem which algorithm has found: